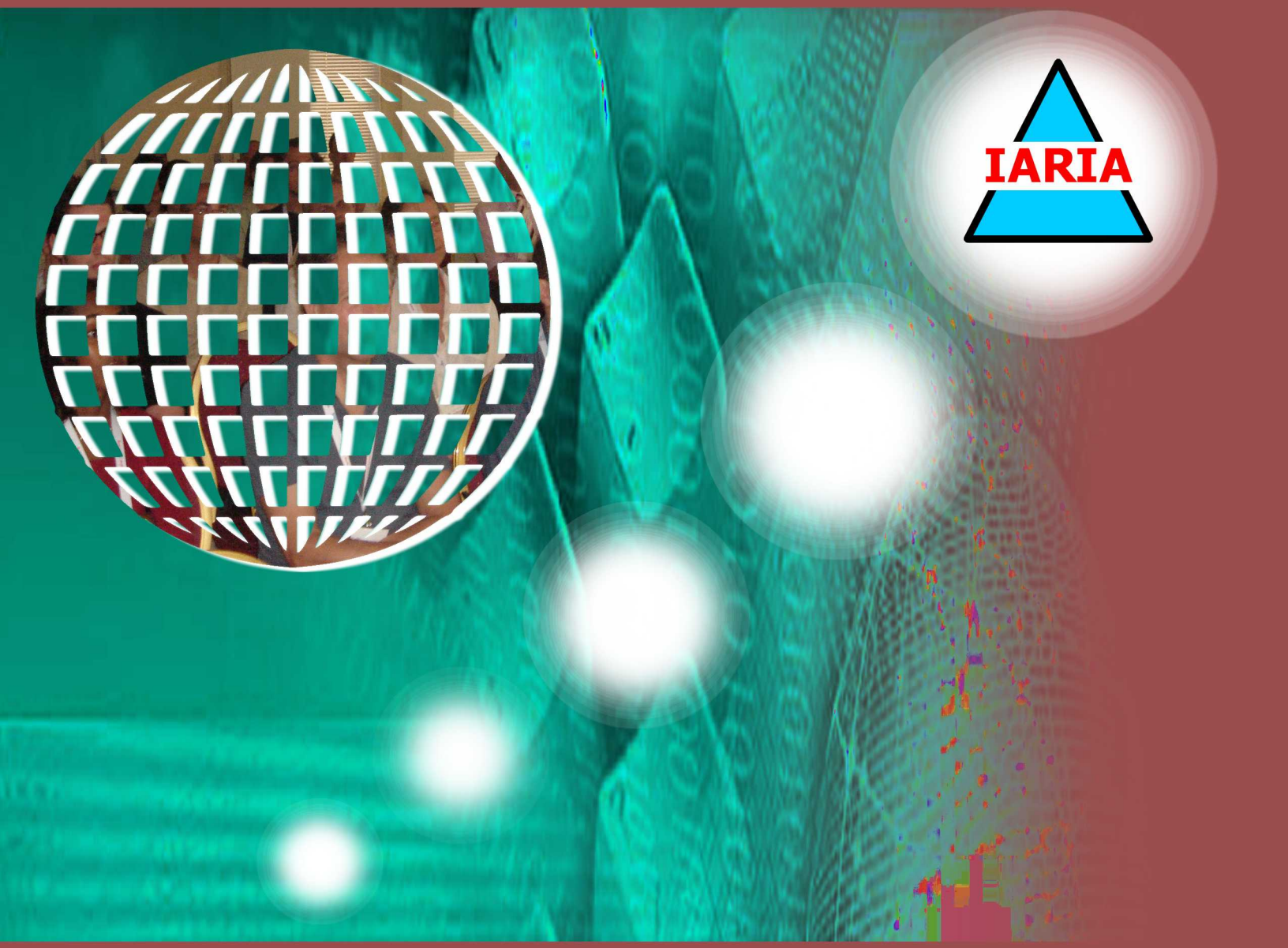
International Journal on Advances in Software



2011 vol. 4 nr. 1&2

The *International Journal on Advances in Software* is published by IARIA.
ISSN: 1942-2628
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Software
Volume 4, Number 1 & 2, 2011

- Constantin Paleologu, University 'Politehnica' of Bucharest, Romania
- Liviu Panait, Google Inc., USA
- Kenji Saito, Keio University, Japan
- Ashok Sharma, Satyam Computer Services Ltd – Hyderabad, India
- Marcin Solarski, IBM-Software Labs, Germany

Advanced Computing

- Matthieu Geist, Supelec / ArcelorMittal, France
- Jameleddine Hassine, Cisco Systems, Inc., Canada
- Sascha Opletal, Universitat Stuttgart, Germany
- Flavio Oquendo, European University of Brittany - UBS/VALORIA, France
- Meikel Poess, Oracle, USA
- Kurt Rohloff, BBN Technologies, USA
- Said Tazi, LAAS-CNRS, Universite de Toulouse / Universite Toulouse1, France
- Simon Tsang, Telcordia Technologies, Inc. - Piscataway, USA

Geographic Information Systems

- Christophe Claramunt, Naval Academy Research Institute, France
- Dumitru Roman, Semantic Technology Institute Innsbruck, Austria
- Emmanuel Stefanakis, Harokopio University, Greece

Databases and Data

- Peter Baumann, Jacobs University Bremen / Rasdaman GmbH Bremen, Germany
- Qiming Chen, HP Labs – Palo Alto, USA
- Ela Hunt, University of Strathclyde - Glasgow, UK
- Claudia Roncancio INPG / ENSIMAG - Grenoble, France

Intensive Applications

- Fernando Boronat, Integrated Management Coastal Research Institute, Spain
- Chih-Cheng Hung, Southern Polytechnic State University, USA
- Jianhua Ma, Hosei University, Japan
- Milena Radenkovic, University of Nottingham, UK
- DJamel H. Sadok, Universidade Federal de Pernambuco, Brazil
- Marius Slavescu, IBM Toronto Lab, Canada
- Cristian Ungureanu, NEC Labs America - Princeton, USA

Testing and Validation

- Michael Browne, IBM, USA
- Cecilia Metra, DEIS-ARCES-University of Bologna, Italy
- Krzysztof Rogoz, Motorola, USA
- Sergio Soares, Federal University of Pernambuco, Brazil
- Alin Stefanescu, University of Pitesti, Romania

- Massimo Tivoli, Universita degli Studi dell'Aquila, Italy

Simulations

- Robert de Souza, The Logistics Institute - Asia Pacific, Singapore
- Ann Dunkin, Hewlett-Packard, USA
- Tejas R. Gandhi, Virtua Health-Marlton, USA
- Lars Moench, University of Hagen, Germany
- Michael J. North, Argonne National Laboratory, USA
- Michal Pioro, Warsaw University of Technology, Poland and Lund University, Sweden
- Edward Williams, PMC-Dearborn, USA

International Journal on Advances in Software
Volume 4, Numbers 1 & 2, 2011

CONTENTS

Stochastic Greedy Algorithms

A Learning-Based Approach to Combinatorial Optimization

Viswa Viswanathan
Stillman School of Business
Seton Hall University
South Orange, NJ, 07079
viswa.viswanathan@shu.edu

Anup K Sen
Management Information Systems
Indian Institute of Management
Calcutta
D. H. Road, Kolkata 700104, India
sen@iimcal.ac.in

Soumyakanti Chakraborty
Information Systems Area
XLRI School of Business and HR
Jamshedpur, India
soumyakc@xlri.ac.in

Abstract **- Research in combinatorial optimization initially focused on finding optimal solutions to various problems. Researchers realized the importance of alternative approaches when faced with large practical problems that took too long to solve optimally and this led to approaches like simulated annealing and genetic algorithms which could not guarantee optimality, but yielded good solutions within a reasonable amount of computing time. In this paper we report on our experiments with stochastic greedy algorithms (SGA) – perturbed versions of standard greedy algorithms. SGA incorporates the novel idea of *learning from optimal solutions*, inspired by data-mining and other learning approaches. SGA learns some characteristics of optimal solutions and then applies them while generating its solutions. We report results based on applying this approach to three different problems – knapsack, combinatorial auctions and single-machine job sequencing. Overall, the method consistently produces solutions significantly closer to optimal than standard greedy approaches. SGA can be seen in the space of approximate algorithms as falling between the very quick greedy approaches and the relatively slower soft computing approaches like genetic algorithms and simulated annealing. SGA is easy to understand and implement -- once a greedy solution approach is known for a problem, it becomes possible to very quickly rig up a SGA for the problem. SGA has explored only one aspect of learning from optimal solutions. We believe that there is a lot of scope for variations on the theme, and the broad idea of learning from optimal solutions opens up possibilities for new streams of research.**

Keywords- greedy algorithms; stochastic approaches; approximate solutions; knapsack problem; combinatorial auctions; single-machine scheduling; machine learning

I. INTRODUCTION

"Greedy" solutions are commonplace in the field of combinatorial optimization for obtaining very quick solutions to complex problems. For example, the unconstrained knapsack problem (UKP) is known to be NP-complete, but there exists a greedy algorithm with $O(N^2)$ time complexity that yields very good solutions in practice. In general, greedy algorithms do not guarantee optimal solutions. In this paper, we elaborate on the idea of stochastic greedy algorithms first presented in [31].

In general terms, a greedy algorithm tackles a problem in several steps. At each step, the algorithm chooses the locally most attractive option with no concern for its effect on global optimality. Greedy algorithms are usually very simple and intuitive. In the Traveling Salesperson Problem (TSP) [12], the problem is to start at a city and visit n-1 other cities and return to the original city while traversing the minimal distance. A greedy algorithm for the TSP is straightforward – at each stage, simply travel to the closest unvisited city and continue this process till the tour is complete. In the Transportation Problem (TP) [17], we are given a set of requirements for goods to be satisfied from stocks available in various warehouses. The unit transportation cost from each warehouse to each demand point is also given and the problem is to satisfy the demands while incurring minimal cost. Vogel's Approximation Method [17] is a greedy algorithm that first finds the warehouse-demand point combination with the lowest unit transportation cost, satisfies the demand to the extent possible, and continues in similar vein till all demands are satisfied (or all supplies are exhausted).

Greedy algorithms are useful when the time available to solve a problem is severely limited. In the space of solution approaches to combinatorial optimization problems, greedy approaches can be seen as lying at one end of the spectrum with optimal algorithms lying at the other extreme. In the middle are approximate algorithms like genetic algorithms and simulated annealing. As we move from the greedy algorithms to optimal algorithms, the solution quality increases with a concomitant increase in the solution time.

In this paper, we elaborate on the results we presented in [31] on *stochastic greedy algorithms* (SGA). Whereas greedy algorithms choose the next step deterministically based solely on what is locally best, SGA, a variant of greedy algorithms, selects it stochastically. In other words, rather than the probability of the best available option being selected being 1, the algorithm uses a probability distribution to select the next step. It selects the next step as the n^{th} best available option with probability p(n). SGA generates many solutions and returns the best one as the output of the algorithm.

How do we determine the probability distribution that specifies p(n)? In seeking quick and good, but not necessarily optimal, solutions to combinatorial optimization problems, researchers have thus far hardly adopted the idea of learning from optimal solutions. We introduce the idea, and showcase the use of a data-mining inspired approach to learn from optimal solutions the probability distribution to use in SGA.

Learning the probability distribution involves solving many problem instances up front to optimality -- which imposes a large fixed cost. SGA is thus only good in situations where this fixed cost can be amortized over many problem instances. Therefore, SGA only makes sense when many problem instances have to be solved on an ongoing basis and the time allowed for solving each instance is small. This could arise for example, when solving the problem is part of a larger business process in a real-time transaction processing system where the on-line user cannot be kept waiting for too long and yet the response to the user's request involves solving a moderately large non-trivial combinatorial optimization problem. With the proliferation of complex web-based transactional processing systems, this scenario is only likely to become increasingly common.

We have experimented extensively with three combinatorial optimization problems – the Unbounded Knapsack Problem (UKP), Single Unit Combinatorial auction (CA) and Single machine sequencing with quadratic penalties (QPSD) , and obtained very encouraging results. These problems represent a good range because the greedy approach is extremely effective for the UKP and very ineffective for the SQP. The combinatorial auction problem falls in between.

On the UKP, we have obtained solutions consistently within 0.02% of the optimal. Our results on the other two problems are also very encouraging and establish SGA as a viable alternative in the pool of soft computing approaches. More research is definitely needed to understand the nuances and to establish performance parameters more rigorously. Nevertheless, the results we present prove conclusively the viability of the approach.

Learning from optimal solutions is a novel, useful and generic idea that opens up exciting new unions between statistics and combinatorial optimization. Whereas we have demonstrated in this paper only a small aspect of learning from optimal solutions, there is clearly unlimited scope to exploit this idea in the search for good quick solutions to combinatorial optimization problems.

In section II we discuss prior work in related areas and present, in section III a generic domain-independent description of SGA. In subsequent sections we discuss the specifics of our application of SGA to the UKP, CA and QPSD problems and the corresponding empirical findings. We conclude the paper with a summary and a discussion of the scope for further work.

II. RELATED WORK

Greedy algorithms [14] represent natural ways of quickly finding good solutions to combinatorial optimization problems [19]. In rare cases, [14], greedy approaches can even guarantee optimal solutions. Greedy algorithms use deterministic steps in that they select the next course of action by choosing the locally best option available. Stochastic algorithms ([10], [11]), on the other hand, use probabilistic elements to alter the steps of the algorithm. Blending the two approaches lies at the heart of SGA. Although researchers have looked at stochastic local search approaches ([4], [10], [11] and [13]), prior research has not explored the pros and cons of stochastic perturbations of known and new greedy approaches.

We use the knapsack problem, single unit combinatorial auctions and a class of single machine sequencing problems to demonstrate the utility of SGA. Knapsack problems have been widely studied in ([16], [21]). The Unbounded Knapsack Problem (UKP) is known to be NP-hard. Greedy approaches to knapsack problems have been discussed in [16]. A new algorithm for finding exact solutions to UKP can be found in [22].

Auctions have been in use since antiquity. The commonest format has been the ascending auction, also known as the 'English' auction. The first major work on auction theory is that of Vickrey [30] who recommended the adoption of second price sealed bid auctions (later called Vickrey auctions). His ideas were extended to combinatorial auctions by Clarke and Groves ([3],[8]). In their scheme, bidders submit their valuations of packages, and the seller solves the revenue maximization problem, known as Winner Determination Problem (WDP) and allocates the bundles. Solving WDP with dynamic programming was proposed by [25]. Two approaches, CASS [5] and CABOB [26] are the prominent heuristic search techniques to solve large instances of WDP optimally for the single unit case. Both these approaches employ Depth-First Branch-and-Bound (DFBB) but they differ in the formulation of their search space. Both these algorithms may take a long time for solving large instances optimally. For the methodical evaluation and comparison of algorithms for solving WDP, Kevin Leyton-Brown et al. [15] designed a suite of distribution families called CATS 2.0 (http://cats.stanford.edu) for generating realistic, economically motivated combinatorial bids in a number of broad real world applications. With the proliferation of on-line auction situations, it is conceivable that there will be an increasing need to obtain reasonably good solutions quickly to CA and related problems.

Single machine sequencing problems [20] are generally known to be NP-hard [23]. The presence of sequence-dependent setup times makes the sequencing problem with quadratic penalties ([28], [29]) very difficult to solve [27]. Greedy approaches to the single machine sequencing problem with quadratic penalties and setup times (QPSD) are not popular yet. The best exact approach reported thus far [18] can solve problems that have only up to 22 jobs. Therefore, providing good solutions to larger instances serves to extend the envelope for this problem.

Machine learning through neural networks has been applied to optimization problems [1]. However, machine learning based on the analysis of optimal solutions to learn their characteristics and then augmenting the process of generating solutions with the resultant knowledge has not been effectively tried before. This paper shows clearly that the approach has promise.

III. GENERIC DESCRIPTION OF SGA

An instance of an *optimization problem* [19] is a pair *(F, c)* where *F* is any set, the domain of feasible points and *c* is the cost function, a mapping: $c: F \rightarrow R^1$. The problem is to

find an $f \in F$ for which $c(f) \leq c(y)$ for all $y \in F$ for a minimization problem (or to find an $f \in F$ for which $c(f) \geq c(y)$ for all $y \in F$ for a maximization problem).

When the set F has a finite number of points, the problem becomes a *combinatorial optimization problem*. A solution procedure that guarantees the best f in the above sense is an *exact* procedure; other procedures are *approximate*.

EXAMPLE 1: In the Unbounded Knapsack Problem (UKP), we are given a knapsack with weight-capacity K, and N items, with item i having weight w_i and value v_i, with each item available in unlimited quantity. The objective is to fill the knapsack with q_i units of the i^{th} item in such a way that the value of the items in the knapsack is maximized. Here

$$F = \{(q_1, q_2, \ldots, q_N) : \sum_{i=1}^{i=N} q_i w_i \leq K\} \quad (1)$$

and

$$c(q_1, q_2, \ldots, q_N) = \sum_{i=1}^{i=N} q_i v_i \quad (2)$$

EXAMPLE 2: In single unit combinatorial auctions, there is only one unit of each item. Bidders place bids on the items or the combination of items they desire, and the auctioneer determines the winning allocation, *i.e.* the set of winning bids. In determining the winning bids the objective is to select a feasible set of bids such that no item is allocated to more than one bid (no overlapping items) and revenue is maximized. Let there be M distinct items and N bids, and let bid B_i has quoted price v_i on a non-empty bundle $S \subseteq M$ of items. In this case: F = {*feasible set of bids with no overlapping items*} and

$$c(x : x \in F) = \sum_{i=1}^{N} u_i \quad (3)$$

where $u_i = v_i$ if bid $B_i \in x$ and 0 otherwise.

EXAMPLE 3: In the Single Machine Sequencing with Quadratic penalties on job completion times and Sequence Dependent Setup times (QPSD) problem [27], there are N jobs, J_i , $i = 1..N$ with J_i having processing time a_i , penalty coefficient q_i and setup times $s_{i,j}$ (being the setup time for J_j when it is immediately preceded by J_i, and $s_{0,j}$ is the setup time for J_i when it is the first job in the sequence). The objective is to find the schedule that minimizes the total cost. Each feasible schedule is a permutation of 1..N and therefore, in this case F = *{all permutations of 1 ... N} and*

$c(x) = \sum_{i=1}^{N} q_i t_i^{\,2}$ where t_i is the completion time for J_i as per permutation x.

It is common to view the solution procedure for a general optimization problem as starting from a given point in F and then moving step by step towards the final solution (optimal or otherwise). For combinatorial optimization problems, a point in the set F is usually determined through a systematic process of construction involving several stages. For example:

- In UKP, each member of F represents one feasible way of filling the knapsack. Constructing one feasible solution involves selecting items one by one and determining how many pieces of each to take. Here, we could see a feasible solution as being constructed through steps with each step involving the selection of an item and a quantity such that the weight added by this item, when combined with the weights of items already added in prior steps, does not exceed the capacity of the knapsack..
- In CA, a member of F represents a feasible set of bids with non-overlapping items. Here, constructing an element of F can be seen as involving a series of steps with each step selecting a bid which does not have any overlapping items with any bid already selected.
- In QPSD, a member of F is any valid permutation of jobs, and creating one could be seen as a series of steps with each step involving the selection of a job which has not already been selected.

Having laid down the fact that creating a member of F involves a process of constructions having several steps, it is now possible to describe abstractly both the greedy approach and SGA. In the greedy approach, we first identify an intuitive measure of attractiveness of each possible step. This measure varies from domain to domain and we will describe the actual measure used for each problem domain when we discuss the domain separately in later sections. At each step in the process of constructing a feasible solution, we choose the step that seems most attractive according to this intuitive estimate. Thus, for UKP, we first choose the item that seems most attractive and take as many units of it as will fit. We then choose as many units of the next best item as will fit and take as many units as possible and so on till no more items will fit. For CA, we first choose the most attractive bid and then choose the most attractive bid from those that remain which do not have an overlap with bids already selected. We go on like this till no more bids are available. For QPSD, we simply order the jobs by their attractiveness with the most attractive job as the first. The generic version of the greedy algorithm is shown below. It is written from the perspective of a maximization problem and can be easily modified for a minimization problem.

We use the following notation:

P	A combinatorial optimization problem
F	The set of feasible solutions to P
a_i	$1 \leq i \leq N$, all possible actions which can be used to construct any feasible solution in F. Each action can be used at most once in building one element of F
r_i	A measure of attractiveness of action a_i , $1 \leq i \leq N$ *(higher is better)*
E_S	The set of eligible actions, given that the actions contained in set S have already been chosen

p_i Probability of choosing the i^{th} most attractive action from E_S (this is used only in SGA)

L Number of trials for SGA

```
Algorithm Greedy {
    S =empty set
    Initialize Es to set of eligible actions
    While Es is not empty {
        From the actions in Es select action ai that
        corresponds to the maximum ri
        Add ai to S
        Remove ai and all ineligible actions from ES
    }
    Output the actions in S
}
```

Figure 1. Algorithm Greedy

In the greedy approach we choose the next step deterministically as the best available step at that point. In SGA, we perform this step stochastically, by selecting at each stage the i^{th} best available step with probability p_i .We generate many solutions in this process and select the best of these as the output. SGA is driven by a probability distribution. The details of how the probability distribution is arrived at are specific to each problem domain and we will describe those when we look at each problem domain separately.

```
Algorithm SGA {
    best_sol = 0
    best_s = empty set
    Repeat the following L times {
        S =empty set
        Initialize Es to set of eligible actions
        While Es is not empty {
            Select a random i using probability distribution
            pi
            From the actions in Es select action ai that
            corresponds to the i^th highest ri
            Add ai to S
            Remove ai and all ineligible actions from ES
        }
        sol = objective function value corresponding to the
        actions in S
        If sol > best_sol {
                best_sol = sol
                best_s = S
        }
    }
    Output the actions in best_s
}
```

Figure 2. Algorithm SGA

IV. SGA APPLICATION TO THE UNBOUNDED KNAPSACK PROBLEM

The problem statement for UKP appears in section III. To implement the greedy approach for UKP, we need a specification of the *attractiveness* r_i, $1 \leq i \leq N$. Intuitively, the "bang for the buck" ratio of v_i/w_i looks like a good measure of the attractiveness of an item and in fact leads to good greedy solutions.

The greedy approach is to order the items in non-increasing order of the ratio v_i / w_i and then to fill the knapsack with as many units of the first item as can fit, and then as many units of the next lowest numbered item that will fit, and so on, till the knapsack is full (that is, the residual weight capacity is less than the weight of the lightest item). In doing this, at each stage we are taking the locally most attractive step, without considering its global effects. It could turn out, for example, that the greedy approach is unable to fill the knapsack completely, but that taking one less unit of one of the items currently in the knapsack would enable us to fill the knapsack completely, albeit with more units of a lower valued item, but with a larger total value. It is for this reason that the greedy solution cannot guarantee optimality.

In UKP there are N items and therefore a maximum of N possible actions at each step. In order to implement SGA for UKP, we need to specify the probability distribution, p_i , $1 \leq i \leq N$ which gives the probability with which the i^{th} most attractive action available is to be chosen. The logic of SGA is that whereas the greedy approach always picks the most attractive step available while constructing a solution, SGA determines this stochastically. Instead of always picking the most attractive item, we select the next item based on a probability distribution. Having selected the item to be used, we next need to decide on how many units of the item should be picked. It is not necessary to fill the knapsack with the maximum number of units possible for the chosen item. Once again we choose this probabilistically. Items are chosen in this fashion till no more can be added to the knapsack. This concludes a single trial. Several trials are performed and the best solution is chosen.

At the stage of selecting the next item, it seems reasonable to assume that the probability of picking items with higher attractiveness should be higher because it is expected that higher the attractiveness, higher is the chance of striking an optimal solution. Likewise, at the stage of choosing the quantity for the selected item, the chance of picking the maximum possible quantity should be highest.

We now describe the procedure we adopted for introducing stochasticity into the greedy approach for the knapsack problem. In the standard greedy approach where the next item to be allocated is chosen strictly according to the best value-to-weight or v_i / w_i ratios, and the maximum possible quantity of the selected item is used. In SGA, we make both of these choices, namely the choice of item and the quantity of the chosen item probabilistically.

We derived the probability distribution empirically by solving many problems to optimality and then learning from these optimal solutions. The dynamic programming solution

procedure for optimally solving the knapsack problem (Gilmore and Gomory [7]) exploits the Bellman Optimality principle.

As explained in the introduction, we introduce the novel idea of learning from optimal solution and using the knowledge thus derived in a stochastic process of generating solutions. This approach can be seen to be inspired by "learning" as applied to data mining. We describe the learning process in detail in the next paragraph. Broadly, the approach relies on generating optimal solutions to a large number of instances of UKP. Once we have optimal solutions to a large number of instances, we seek patterns in these. In this paper we rely on the large body of optimal solutions to calculate the probability with which the best available piece is selected, the probability of the second best available piece, and so on. We also calculate the probability of the optimal solution containing the maximum number of units of the selected piece, 1 less than the maximum and so on. Once we have these, we can then use these probabilities to generate a large number of random solutions and choose the best among them. We based the calculations on optimal solutions to a total of 500 problem instances with N varying from 50 to 250. We lumped problems with different values of N together because we did not find any significant differences in the probabilities when we calculated them separately for different values of N.

We now describe the procedure for learning the probability distributions. Consider a knapsack problem with capacity 20, and 5 items with weights w_i = {8, 3, 10, 5 and 2} in non-increasing order of their value-to-weight ratios (we ignore the actual values for this discussion). Suppose the optimal solution x_i = {1, 0, 1, 0 1} (one unit each of items 1, 3 and 5). Note that this differs from the greedy solution which would be {2, 1, 0, 0, 0}. Looking at this optimal solution, we find that initially when the knapsack is empty, all of the items are eligible for consideration and the optimal solution actually used the best available item, namely the first, although it does not use the maximum quantity possible – two units would have fitted into the knapsack, but the optimal solution uses only one unit. At the next stage, the residual knapsack capacity is 12 (having allocated one unit of item 1). Even at this stage, the residual capacity is sufficient for all the remaining items to be eligible for consideration – it can hold at least one unit of each of them. However, we see that the optimal solution for the sub-problem did not choose the best item and instead chose only the third best item (namely item 3). Only one unit of this item could fit and hence the maximum allowable number of units were used. The residual knapsack capacity now is 2 and the optimal solution now chose the best item available (only item 5 is eligible for consideration now because only it can fit) and the maximum allowable quantity, namely 1, was used.

We did the above analysis for each optimal solution and calculated the probability of the j^{th} eligible item being actually chosen, and also noted the probability of the number of units of the chosen item used in the optimal solution deviating by an amount *d*, *d* = 1, 2, 3, … from the maximum amount that would fit into the residual capacity.

In this way we calculated the probability p_j of the item with the j^{th} highest ratio being chosen as the next item. Similarly we also calculated the probability $q_{j,k}$ of the number of units of the selected item *j* being less than the maximum possible number by k units.

Figure 3 shows the algorithm for applying SGA to UKP.

Algorithm SGA_UKP

Re-order the *N* items such that $v_1/w_1 \geq v_2/w_2 \geq .... \geq v_N/w_N$

$best\_val \leftarrow 0$

$best_k \leftarrow 0, k = 1, 2, ... N$

Repeat *numtrials* times {

  $capacity \leftarrow K$

  $curpos \leftarrow 1$

  $sga\_value \leftarrow 0$

  while $(capacity \geq \min(w_i), i = 1..N))$ {

    Randomly select a position *j* according to the chosen probability distribution for the position of the next item relative to *curpos*

    Starting from *curpos* skip the first *j* items whose weights are not greater than *capacity*. Let *k* be the index of the next item whose weight is not greater than *capacity*. If this causes a spillover beyond *N*, then search backwards for the first item whose weight is not greater than *capacity*

    $maxunits \leftarrow floor(capacity/w_k)$

    Randomly select a number *m* according to the chosen probability distribution for the quantity of the next item relative to *maxunits*

    Set sol_k the number of units of the k^{th} item in the solution to *max(1, maxunits - m)*

    $curpos \leftarrow k - 1$

    $capacity \leftarrow capacity - sol_k * w_k$

    $sga\_value \leftarrow sga\_value + sol_k * v_i$

  }

  if (*sga_value* > *best*_val)

    $best\_val = sga\_value$

    $best_k \leftarrow sol_k, k = 1..N$

}

Figure 3. Algorithm SGA-UKP

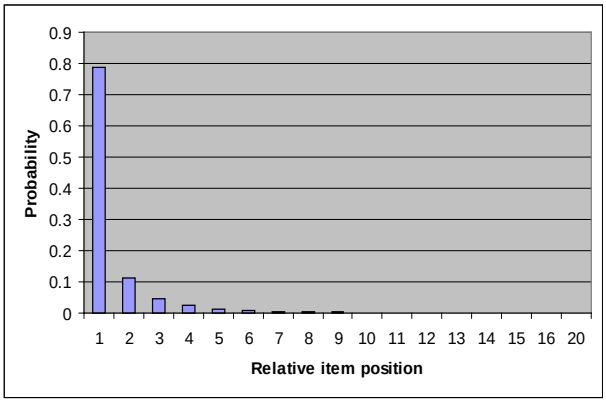


Figure 4. Example of learned probability distribution for position of next item relative to current item (based on 500 random instances)

Figures 4 and 5 show the probability distributions we obtained experimentally for p_j and q_{jk} respectively. The first bar on Figure 4 shows for example that almost 80% of the time the optimal solution chooses the next item as the one with the highest value-to-weight ratio. The second bar shows that there is a close to 10% chance that this is the second best item. Similarly, the first bar in Figure 5 shows that about 36% of the time the optimal solution will utilize the maximum number of units of the selected item. The second bar shows that about 18% of the time, the optimal solution will use one unit less than the maximum possible and so on. Given the probability distributions being used, and an optimal solution, it is easy to calculate the probability that SGA will generate the given optimal solution.

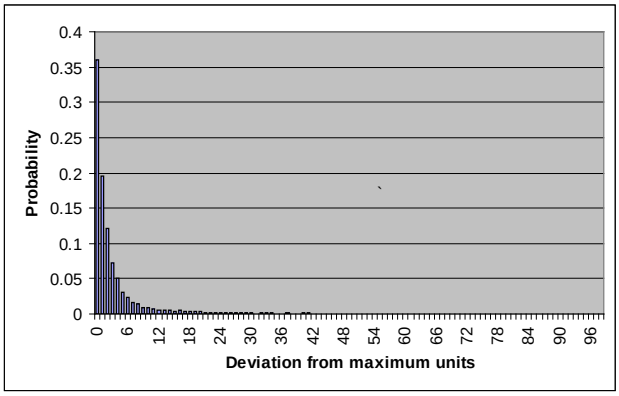


Figure 5. Example of learned probability distribution for extent of deviation of quantity used from maximum possible (based on 500 random instances)

Suppose the probability that SGA will generate an optimal solution in a single trial is *p*, then the probability that it will generate a non-optimal solution in a single trial is *(1-p)*. If there are *L* trials, the probability that each trial generates a non-optimal solution is $(1-p)^L$. Therefore the probability that at least one of the trials generates an optimal solution is

$$1-(1-p)^L$$

As is well known, this number can be surprisingly close to 1 for even fairly low values of *p*. This probability estimate is somewhat lower than then real value, as a problem could have multiple optimal solutions. Also, it is possible for a given solution to be generated in more than one way by our algorithm.

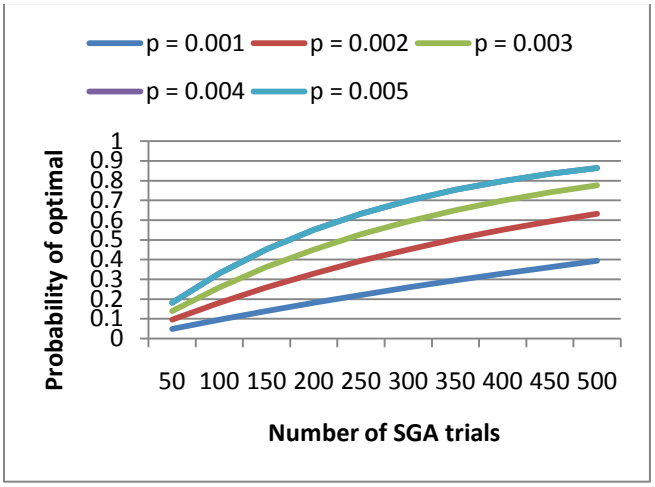


Figure 6. Probability of finding optimal solutions in SGA as number of trials increses

To demonstrate the probability calculation, we revert to the example used earlier. Suppose we have a knapsack problem with capacity 20 and 5 items with weights w_i = {8, 3, 10, 5 and 2} in non-increasing order of their value-to-weight ratios (we ignore the values for this discussion). Suppose the probability of distribution for item position is {0.6, 0.3, 0.1}. This means that the best item available was chosen 60% of the time, the second best 30% of the time and the third best 10% of the time. Suppose the probability distribution for the deviation from the maximum is {0.7, 0.25, 0.05}. This means that the maximum number of units possible would be used 70% of the time, one less than the maximum would be used 25% of the time and two less than the maximum would be used 5% of the time.

Suppose the optimal solution x_i = {1, 0, 1, 0 1} (one unit each of items 1, 3 and 5). With these numbers, the probability of SGA finding the optimal solution in a single trial is the product of the probability of selecting each of the actual items chosen and the probabilities of the correct quantities being chosen. The first element of the optimal solution is a choice of one unit of the best item. The probability of this happening is the probability of the first item being chosen – which is 0.6 times the probability that the deviation from the maximum number of units being 1 (since 2 units will fit, but only one unit is represented in the optimal solution) which is 0.25. Calculating in this way we find the probability as (0.6*0.25)*(0.1*0.7)*(0.6*0.7) = 0.0041. Therefore the probability of generating an optimal solution in 250 trials will be about 0.63. Since the computation and the results are similar for other problem

domains, we have not shown this probability for CA and QPSD problems.

Table I and II show the results obtained on UKP instances with varying numbers of items, as well as different instance types based on [21], where the weights and values are weakly correlated (easy) and strongly correlated (harder),

TABLE I. AVERAGES OF 100 RUNS FOR UKP USING LEARNED PROBABILITY DISTRIBUTIONS IN WEAKLY CORRELATED CASE

N	Weakly correlated (easy)						
	Greedy %dev	*SGA_UKP*					
		50 trials		*100 trials*		*200 trials*	
		%dev	*Prob*	*%dev*	*Prob*	*%dev*	*Prob*
50	0.410	0.097	0.321	0.061	0.367	0.012	0.564
100	0.401	0.112	0.243	0.079	0.310	0.013	0.545
150	0.425	0.100	0.266	0.073	0.300	0.015	0.496
200	0.435	0.100	0.263	0.063	0.290	0.018	0.439
250	0.458	0.103	0.277	0.063	0.275	0.019	0.418

TABLE II. AVERAGES OF 100 RUNS FOR UKP USING LEARNED PROBABILITY DISTRIBUTIONS IN STRONGLY CORRELATED CASE

N	Strongly correlated (hard)						
	Greedy %dev	*SGA_UKP*					
		50 trials		*100 trials*		*200 trials*	
		%dev	*Prob*	*%dev*	*Prob*	*%dev*	*Prob*
50	0.430	0.172	0.180	0.110	0.171	0.061	0.256
100	0.490	0.200	0.151	0.146	0.141	0.090	0.251
150	0.521	0.210	0.128	0.171	0.130	0.113	0.183
200	0.541	0.225	0.113	0.183	0.222	0.142	0.121
250	0.580	0.251	0.107	0.175	0.104	0.150	0.100

All the data are based on an average over 100 problem instances. For each instance, we also calculated the probability of SGA obtaining the optimal solution, and the tables show these as well. Across all the figures in Tables I-II, the deviation from optimal for the greedy solution is, at the minimum, 2.5 times the SGA deviation and the maximum is 35 times.

V. APPLICATION TO COMBINATORIAL AUCTIONS

In single unit combinatorial auctions, there is only one unit of each item. Bidders place bids on the items or the combination of items they desire, and the auctioneer determines the winning allocation, *i.e.* the set of winning bids. In determining the winning bids the objective is to select a feasible set of bids such that no item is allocated to more than one bid (no overlapping items) and revenue is maximized. The formal description of the problem is given in section III.

Individual items have no prices associated with them. Prices are only associated with bids and each bid can be for many items. Accordingly a useful measure of attractiveness of a bid is its price per item. Thus suppose a bid has price 200 and is for four different items. The price per item for this bid is 50. Suppose there is another bid whose price is 80, but is for just a single item. Then the second bid is in some sense preferable to the first because its price per item is higher. Table III below shows an example of CA with 10 items and 5 bids.

TABLE III. SINGLE UNIT COMBINATORIAL AUCTION WITH 10 ITEMS AND 5 BIDS

Bid no	Price	Items in bid	Attractiveness
1	100	{8, 9, 10}	33.33
2	125	{6, 9, 2, 1}	31.25
3	75	{4, 6}	37.5
4	80	{5, 7, 1}	26.66
5	30	{6}	30

The greedy approach for CA therefore is very straightforward. Simply pick the most attractive bid first and then continue to pick the most attractive remaining bid which has no overlapping items with any bids already chosen. In the above example, first we would choose bid 3. Then we can choose bid 1. Now, since items 4, 6, 8, 9 and 10 have already been chosen, only bid 4 can be chosen because of item overlap considerations. The greedy solution is 255, which also happens to be the optimal solution,

For learning the probabilities, we ran CA to optimality using CASS [5]. We then analyzed the optimal solutions generated by CASS. For each optimal solution generated by CASS, we first considered the bids in the optimal solution in their order of their attractiveness. We then tallied the number of times the optimal solution picked the best admissible bid, the second best admissible bid and so on. We calculated the probability with which CASS chose the most attractive bid at each stage. Using the above problem as an example, we would see that the optimal solution selected the best available bid at each stage. It is important to note that while analyzing the optimal solutions, we consider only the admissible bids at any stage. For example, it is possible that at some stage the optimal solution uses the fifth best bid overall. However, if at that stage this bid happens to be the best among the *admissible* bids at that stage based on overlaps with bids already selected, then we will consider that the best bid has been chosen. Suppose we perform this analysis over a large number of problems and see that the i^{th} best available bid was chosen n_i times across all the problem instances. Then the probability of SGA choosing the i^{th} best available bid at any point is (N being the number of bids)

$$p_i = \frac{n_i}{\sum_{i=1}^{N} n_i}$$

```
Algorithm SGA_CA {
    best_val = 0;
    best_bids = empty set
    repeat num_trials times {
        selected_bids = empty set
        S = set of all bids
        val = 0;
```

```
        while the set S of bids is not empty {
            randomly select i based on the learned
                probability distribution p_i
            select the i^th most attractive bid b_i from S
            add b_i to selected_bids
            remove b_i and all bids overlapping with b_i from
                S
            val = val + p_i
        }
        If val > best_val {
            best_val = val
            best_bids = selected_bids
        }
    }
    Output best_sol and best_bids
}
```

Figure 7. Algorithm SGA_CA

The probability distribution that we gleaned from optimal solutions is shown in Figure 8.

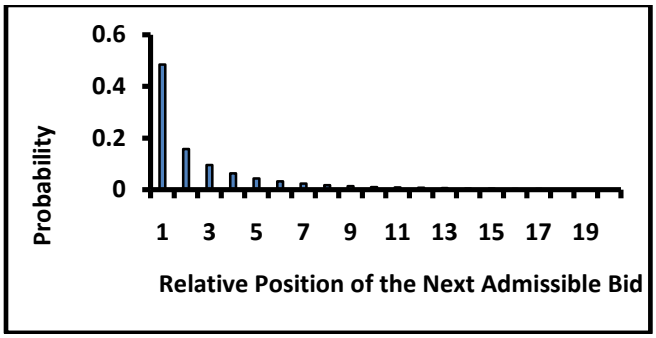


Figure 8. Example of learned probability distribution for relative position of next admissible bid (based on 500 random instances with number of goods varying from 10 to 40 and number of bids between 50 and 200)

The results of running SGA on CA are shown in Table IV. Each row shows the average of 100 random problem instances generated using the standard problem suite generator CATS 2.0 [15]. The results indicate very clearly that even with 50 trials, SGA is able to drastically improve on the greedy solution.

We wanted to see if this impressive performance of SGA was merely the result of the fact that the greedy solution was quite poor in the problem instances tested. We wanted to create a situation where the greedy solution is a lot closer to the optimal solution and then see if SGA can provide benefits even under this scenario that tests SGA more rigorously. We hypothesized that if the number of bids in relation to the number of items is drastically increased, then the greedy solution is likely to come a lot closer to the optimal solution on the average. We expected this because the drastically increased number of bids will make available many more attractive bids than would have been possible with fewer bids. Accordingly we generated random problem instances with a significantly larger number of bids. The results on running SGA on this set are shown in Table V. As we expected, the greedy solution was indeed a lot closer to the optimal. Encouragingly, SGA still managed to improve significantly upon the greedy solution.

TABLE IV. AVERAGES OF 100 RUNS FOR CA USING LEARNED PROBABILITY DISTRIBUTIONS IN CASE OF WEAK GREEDY SOLUTIONS

No. of Goods	No. of Bids	Greedy % dev	SGA			
			50 Trials	100 Trials	200 Trials	500 Trials
			% dev	% dev	% dev	% dev
10	50	19.97	2.39	0.86	0.70	0.29
10	200	11.18	2.28	1.43	0.95	0.29
10	500	6.96	2.01	1.50	1.07	0.49
10	1000	6.75	3.08	2.45	2.14	1.68
12	50	21.50	2.18	1.08	0.41	0.23
12	200	12.58	2.70	1.79	1.09	0.55
12	500	7.34	2.56	1.68	1.19	0.82
12	1000	8.49	4.05	3.50	2.86	2.50
15	50	18.45	2.12	1.22	0.63	0.27
15	200	11.80	2.51	1.83	1.25	0.60
15	500	6.65	2.63	1.95	1.55	1.06
15	1000	9.24	4.82	4.36	3.83	3.25
20	50	27.73	4.08	2.49	1.32	0.46
20	200	15.62	4.56	3.03	2.15	1.46
20	500	10.98	4.54	3.75	3.08	2.43
20	1000	13.38	6.92	6.30	5.58	4.89
26	50	27.07	5.19	4.28	2.85	2.40
26	200	19.09	5.44	4.06	3.15	2.01
26	500	15.25	6.65	5.43	4.54	3.32
26	1000	20.79	11.08	9.98	9.05	8.00
30	50	27.92	6.13	4.59	2.81	1.91
30	200	19.11	6.53	4.85	3.84	2.92
30	500	13.22	5.85	5.16	4.08	3.31
30	1000	19.68	11.39	10.70	9.57	8.81
40	50	31.74	7.71	5.75	3.96	2.67
40	200	20.93	8.99	7.49	5.91	4.61
40	500	15.34	7.65	6.42	5.47	4.68
40	1000	21.18	13.07	12.13	11.28	10.41

We were curious to see if the probability distributions for the problems with lower number of bids and those for the problems with a huge number of bids would be significantly different. It turned out that they were very stable. The probability distribution is shown in Figure 9. Thus, while it might be a good idea to re-learn the probability distributions when the problem parameters change a lot, this finding indicates that in a time crunch nothing much would be lost in using a probability distribution obtained from a set of problem instances with different characteristics.

VI. APPLICATION TO SINGLE MACHINE SEQUENCING

We also studied the performance of SGA on a very hard single machine sequencing problem with quadratic penalties on job completion times and sequence dependent setup times (QPSD) [27]. This is also described in section III. In QPSD, there are N jobs , J_i i = 1..N with J_i having processing time a_i, penalty coefficient q_i and setup times $s_{i,j}$ (being the setup time for J_j when it is immediately preceded by J_i, and $s_{0,j}$ is the setup time for J_j when it is the first job in the sequence). We assume that all values are non-negative integers.

Figure 9. Example of learned probability distribution for relative position of next admissible bid on problems with large nunmber of bids (500 and 1000)

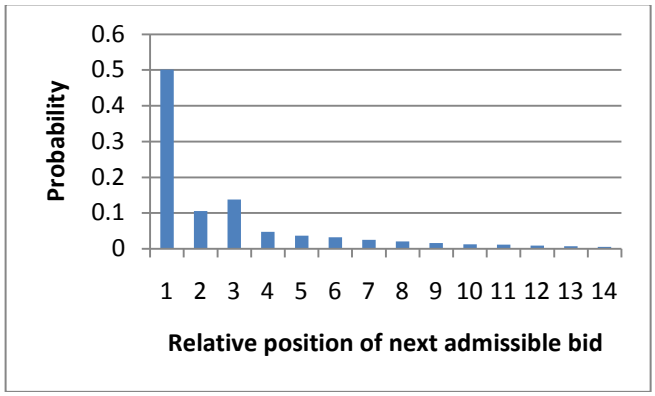


TABLE V. AVERAGES OF 100 RUNS FOR CA USING LEARNED PROBABILITY DISTRIBUTIONS IN CASE OF STRONG GREEDY SOLUTIONS

No. of Goods	No. of Bids	Greedy % dev	SGA			
			50 Trials	100 Trials	200 Trials	500 Trials
			% dev	% dev	% dev	% dev
10	3000	1.27	0.72	0.59	0.44	0.31
10	5000	0.67	0.53	0.43	0.35	0.26
10	8000	0.54	0.38	0.31	0.26	0.18
10	10000	0.37	0.25	0.20	0.16	0.14
12	5000	0.76	0.53	0.46	0.37	0.27
12	8000	0.62	0.47	0.40	0.33	0.23
12	10000	0.31	0.23	0.21	0.19	0.13
15	8000	0.64	0.55	0.50	0.39	0.33
20	10000	1.34	1.10	0.99	0.88	0.78
26	10000	2.15	1.79	1.67	1.59	1.39
26	12000	1.67	1.55	1.47	1.36	1.20
26	15000	1.56	1.36	1.28	1.17	1.01
26	20000	0.91	0.80	0.77	0.70	0.64
30	8000	5.78	3.70	3.44	3.00	2.60
30	10000	3.49	2.76	2.51	2.26	2.03
30	12000	2.31	1.99	1.88	1.77	1.55
30	15000	1.76	1.62	1.53	1.42	1.20
40	4000	6.55	4.61	4.33	3.99	3.63

Let

$$e_{i,j} = s_{i,j} + a_j \forall i \in \{0..N\}, j \in \{1..N\} \quad (4)$$

be the effective processing time for J_j when it is immediately preceded by J_i. Let M be a number such that

$$M > \sum_{j=1}^{N} \max(e_{i,j}), i \in \{0..N\}. \quad (5)$$

The objective is to minimize the total penalty across all jobs, that is, to minimize the weighted sum of the square of completion times. When the setup times are sequence-dependent, the quadratic penalty problem becomes extremely difficult to solve. Drawing from Balas [2], the problem of minimizing the total penalty was formulated by [31] as:

$$Min \sum_{j=1}^{N} q_j t_j^2$$

Subject to:

$$t_j \geq \min\{e_{0,j}, j \in \{1..N\}\} \quad (6)$$

$$t_j + e_{j,k} \leq t_k + M(1 - x_{j,k}),\ j < k,\ \ j,k \in \{1..N\} \quad (7)$$

$$t_k + e_{k,j} \leq t_j + M x_{j,k},\ \ j < k,\ j,k \in \{1..N\} \quad (8)$$

$$t_j \in \left\{0..\sum_{j=1}^{N} \max(e_{i,j}), i \in \{0..N\}\right\} \quad (9)$$

$$x_{j,k} \in \{0,1\},\ j < k,\ \ j,k \in \{1..N\} \quad (10)$$

Constraint 1 addresses the completion time for the first job in the sequence. Constraints 7 and 8 ensure that for any pair of jobs j and k, either j precedes k or k precedes j. We use "j < k" in constraints 7, 8 and 10 to reduce the number of x-variables by half. As in the case of Traveling Salesman Problem, such a formulation may not be efficient to solve in practice using IP solvers.

In [27], it has been shown that the search space for sequencing problems can be modeled as a tree, or as a graph, and those algorithms using the graph search space run faster. For the QPSD problem under the tree formulation, two nodes with the same set of jobs but in different orders and having the same last job will generally not have the same cost because the setup times for the jobs could differ. Nevertheless, the sub trees below them are identical in terms of the structure. Algorithms using the tree search space cannot take advantage of this fact and might wastefully traverse these identical sub-trees more than once. The graph search space has far fewer nodes and offers the potential for faster search. The node count reduction results from the fact that unlike in the tree search space, there could be multiple paths from the root node to any given node, and this helps to avoid replicating the identical sub trees. However, sequence-dependent setup times complicate traditional graph search because the identical sub trees may not have the same costs.

The main feature of graph search algorithms like the graph version of A* [9] is that when these reach the same node through different paths, they retain the path having the lowest cost, discarding any other paths from the root to the node. This approach works fine when the incremental cost from a given node to a goal node is independent of the path by which the node was reached. This is the same as the principle of optimality on which the dynamic programming formulations [12] are based. However, this does not hold for sequence-dependent setup times[18]. For example, consider the following 4 job problem given in Table VI below.

TABLE VI. 4 JOB QPSD PROBLEM

Job	Setup Times				Proc. Times	Penalty Coeff
	1	2	3	4		
1	-	1	1	3	1	2
2	1	-	3	2	4	1
3	5	4	-	10	3	1
4	3	6	9	-	10	1

In this example, it is assumed that the setup time for a job is zero if it is the first in the sequence. Consider the ordered sequence of jobs (1, 2, 3) and (2, 1, 3). Under the graph formulation, a node is represented by the set of completed jobs without regard to the ordering, except for the last job in the sequence. Because the set of jobs and the last job in the two ordered sequences in question are the same, the two are represented by a single node $(\{1,2\},3)$, where the first two jobs form an (unordered) set and the last job is shown separately. The cost when the node is reached through the sequence 1, 2, 3 is 182 and through the sequence 2, 1, 3 is 188. If a traditional graph search algorithm reaches the node through the two different paths considered, it would simply discard the higher cost path 2, 1, 3. However, if we look below this node, we see that the sequence 1, 2, 3, 4 has a cost of 1206, which is higher than the cost of the sequence 2, 1, 3, 4 which is 1088. A traditional graph search algorithm thus runs the risk of missing the optimal solution.

In [18], solutions for QPSD only up to 22-job problems using a memory constrained graph search algorithm have been reported. Increasing the memory limit to 512K nodes, we could solve 30-job problems using PC running Windows XP. We wanted to study how SGA performs on this hard problem. For a simpler problem not involving setup times, Townsend [29] had proposed two sufficient conditions for a given sequence of jobs to be optimal. The first of these involves ordering the jobs by non-ascending order of their p_i/a_i ratios. Being only one of two sufficient conditions for optimality, this ordering cannot guarantee optimal solutions for the simpler problem, but it does provide the basis for very good greedy solutions for that problem. In the absence of any other known greedy approaches to QPSD, we chose to adopt Townsend's heuristic.

Our SGA application to QPSD orders the jobs as above, and at each stage, chooses the job with the highest p_i/a_i ratio. Since UKP and CA have already established the benefit of learning from optimal solutions, we wanted to check and see how a standard discrete probability distribution with the right shape would perform for SGA. The benefit of doing this is that the up-front cost of solving many problem instances to optimality can then be avoided. Accordingly, in our experiments with QPSD, instead of learning the probability distribution from the solutions to optimal solutions, we experimented with both the Geometric and the Binomial distributions (since they can have the proper shape with suitably chosen parameters) and found that the Binomial distribution with a low value for its parameter performed better. The results are given in Table VII. It shows that the results for QPSD are good, but not as impressive as for UKP. It is intuitively clear that SGA can give good results only when the underlying greedy algorithm is reasonably good. Results of SGA application to QPSD - based on 100 trials and averaged over 100 random problem instances for each value of N.

TABLE VII. RESULTS OF SGA APPLICATION TO QPSD (BASED ON 100 TRIALS AND AVERAGED OVER 100 RANDOM PROBLEM INSTANCES FOR EACH VALUE OF N)

N	% deviation from optimal		
	Greedy	SGA – Binomial (p = 0.025)	SGA – Geometric (p=0.8)
10	7.70	1.67	2.07
12	10.02	2.81	3.48
14	11.13	3.70	4.65
16	11.05	4.01	5.14
18	12.50	4.93	6.20
20	13.91	6.10	7.61
22	14.45	7.11	8.34
24	14.41	7.43	8.93
26	15.64	8.31	9.92
28	15.54	8.53	9.75
30	15.71	9.22	10.56

For QPSD we based the greedy approach on a result obtained for a far simpler problem, and its performance was not very good. Nevertheless, we find that SGA is able to improve upon the solution significantly. We need to experiment with learned distributions in this domain too.

VII. CONCLUSIONS

We have proposed a new approximate approach called the Stochastic Greedy Algorithm and presented the results of its application to the Unbounded Knapsack Problem, Combinatorial Auctions and a hard Single Machine Sequencing Problem.

The two major contributions of SGA are

- its combining greedy approaches with stochastic approaches
- its introduction of the idea of learning from the characteristics of optimal solutions to incorporate in a generative approach

In all three domains, SGA provides significant improvements over the greedy solution. Of the three, the results for the single machine sequencing problem are perhaps relatively weak, and one reason for this is that no good greedy approach is known for the problem as of now. One important finding is that standard discrete probability distributions perform quite well and that, if necessary, the costly step to learn the underlying probability distribution can be avoided on occasion. Furthermore, our findings seem to hint that probability distributions are pretty stable and need not necessarily be re-learned when the problem characteristics change.

Our results explore the potential for learning patterns from optimal solutions and applying this learning in the process of generating solutions. There is obviously much more scope to extend this "supervised learning" approach for combinatorial optimization. While analyzing optimal

solutions to learn characteristics, it is possible to assign several other descriptors to each decision point. For example, in the knapsack problem, we could attach the percentage difference between the best available option and the next best one as a descriptor. Once the decisions made in the optimal solution are thus tagged, we effectively have different probability distributions for different states and SGA could sample from a more fine-grained and situation-specific probability distribution. Another approach would be to study numerous optimal solutions and impute decision rules and see how solutions based on such rules perform. Broadly speaking, this learning metaphor can be exploited in numerous ways and certainly opens up new avenues for further work.

REFERENCES

[1] A. Cochocki, R. Unbehauen, Neural Networks for Optimization and Signal Processing, New York: John Wiley, 1993,

[2] Balas, E., 1985. On the facial structure of scheduling polyhedra, *Mathematical Programming*, 24, 179-218

[3] E. H. Clarke, "Multipart pricing of public goods," *Public Choice* (11) 1971, pp 17 - 33.

[4] A. Feldman, G. Provan and A. V. Gemund, Computing minimal diagnoses by greedy stochastic search**,** In *Proc. AAAI* 2008, pp. 911-918.

[5] Y. Fujishima, K. Leyton-Brown, and Y. Shoham, "Taming the Computational Complexity of Combinatorial Auctions: Optimal and Approximate Approaches," in: *International Joint Conference on Artificial Intelligence*, Stockholm, 1999, pp. 548 - 553.

[6] E. C. Freuder, R. Dechter, B. Ginsberg, B. Selman. and E. P. K. Tsang, 1995. Systematic versus stochastic constraint satisfaction. In *Proc. IJCAI 95*, volume 2.

[7] P. C. Gilmore and R. E. Gomory, 1966, The theory and computation of knapsack functions, Operations Research, 14(6), pp 1045-1074

[8] T. Groves, "Incentives in Teams," *Econometrica* (41) 1973, pp 617 - 631.

[9] P. Hart, N. Nilsson and B. Raphael, 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Trans. Syst. Science and Cybernetics*, SSC4(2):100-107

[10] H. H. Hoos and T. Stutzle, *Stochastic Local Search Foundations and Applications*, 2004, Elsevier.

[11] J. Hromkovic, R. Královič, M. Nunkesser, and P. Widmayer (Eds.), *Stochastic Algorithms: Foundations and Applications*, Proceedings of 4th International Symposium, SAGA 2007, Lecture Notes in Computer Science, Zurich, Switzerland, Sept 13-14, 2007.

[12] D. S. Johnson and L. A. McGeoch, The Traveling Salesman Problem: A Case Study in Local Optimization, In *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J.K. Lenstra (Eds), John Wiley and Sons Ltd, 215-310, 1997.

[13] K. Kask, and R. Dechter, 1999, Stochastic local search for Bayesian networks. In *Proc. AISTAT'99*, 113–122.

[14] J. Kruskal, Greedy algorithm for the minimum spanning tree problem, *Proceedings of the American Mathematical Society*, 48-50, 1956.

[15] K. Leyton-Brown, M. Pearson, and Y. Shoham, Y. "Towards a Universal Test Suite for Combinatorial Auction Algorithms," in: *ACM Conference on Electronic Commerce*, 2000a, pp. 66 -76.

[16] S. Martello and P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, 1990.

[17] M. Mathirajan, and B. Meenakshi, Experimental Analysis of some Variants of Vogel's Approximation Method, *Asia-Pacific Journal of Operational Research* 21(4), 447-462, 2004.

[18] S. A. Mondal and A. K. Sen, 2000. TCBB scheme: Applications to single machine sequencing problems, *Proc AAAI-2000*, pp. 792-797.

[19] K. Papadimitriou and K. Steiglitz K. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[20] M. Pinedo, Scheduling: Theory, Algorithms and Systems. Prentice Hall. 1995.

[21] D. Pisinger, *Algorithms for Knapsack Problems*, Ph. D. Thesis, Department of Computer Science, University of Copenhagen, Denmark, 1995.

[22] V. Poirriez, N. Yanev and R. Andonov, "A hybrid algorithm for the unbounded knapsack problem", Discrete Optimization, volume 6, 2009, pp. 110-124.

[23] A. H. G. Rinooy Kan, Machine Complexity Problems: Classification Complexity and Computations. Nijhoff, The Hague, 1976.

[24] F. Rossi, P. v. Beek and T. Walsh, Constraint Programming, In *Handbook of Knowledge Representation*, Edited by B. Porter, V. Lifschitz and F. van Harmelen, 2008 , Elsevier B.V.

[25] M. H. Rothkopf, A. Pekec and R. M Harstad. Computationally Manageable Combinatorial Auctions. Management Science, 44(8):1131 – 1147, 1998.

[26] T. Sandholm, "Algorithm for Optimal Winner Determination in Combinatorial Auctions," *Artificial Intelligence* (135) 2002, pp 1 - 54.

[27] A. K. Sen, and A. Bagchi, Graph Search Methods for Non-order-preserving Evaluation Functions: Applications to Job Sequencing Problems, *Artificial Intelligence*, 86(1), 43-73, 1996.

[28] W. Szwarc, M. E. Posner and J. J. Liu, "The single machine scheduling problem with quadratic penalty function of completion times", Management Science. Volume 34, no 2, 1988, pp. 1480-1488.

[29] W. Townsend, The Single Machine Scheduling Problem with Quadratic Penalty Function of Completion Times: A Branch-and-bound Solution, *Management Science*, 24(5), 530-534, 1978.

[30] W. Vickrey, "Counterspeculation, Auctions, and Competitive Sealed Tenders," *Journal of Finance* (16) 1961, pp 8 - 37.

[31] K. V. Viswanathan and A. K. Sen, Greedy by Chance – Stochastic Greedy Algorithms, Proceedings of the Sixth International Conference on Autonomic and Autonomous Systems (ICAS 2010), March 7-13, 2010, Cancun, Mexico, published by IEEE CPS, pp. 182-187.

An Adaptive Multimedia Presentation System

Philip Davies
Faculty of Technology
Bournemouth and Poole College
Bournemouth, UK
pdavies@bpc.ac.uk

David Newell
Software Systems Research Group
Bournemouth University
Bournemouth, UK
dnewell@bournemouth.ac.uk

Nick Rowe
Faculty of Technology
Bournemouth and Poole College
Bournemouth, UK
nrowe@bpc.ac.uk

Suzy Atfield-Cutts
Software Systems Research Group
Bournemouth University
Bournemouth, UK
satfieldcutts@bournemouth.ac.uk

***Abstract* - Requirements elicitation for a multimedia presentation system for e-learning led the writers to propose a video segmentation process that adapts learning materials through online interventions between the student and tutor. The tutor tailors audio/visual segments by dynamically inserting new fragments that provide supplementary updates in response to questions from students. A survey of advanced adaptive approaches revealed that processing of manually or automatically generated metadata would provide better adaptation. Automated use of metadata requires storage and processing of context dependent ontology hierarchies that describe the semantics of the curriculum. Data and semantic models needed to adaptively process multimedia presentations in real-time are derived. The design models are implemented using HTML, XML and Flash. The authors conclude that the use of context-based rules that process meta-level descriptions of segmented multimedia components stored according to a bounded ontology can produce a system that dynamically adapts learning materials.**

Keywords – e-learning, adaption, metadata, semantic, ontology.

I. INTRODUCTION

Traditional lectures and seminars are being supplemented or replaced by multimedia presentation systems. However, this movement towards on-line learning suffers from a number of drawbacks, such as reductions in contact with real tutors and changes to the traditional teaching-learning feedback loop. The Adaptive Multimedia Presentation System (AMPS) is an attempt to overcomes some of these drawbacks [1].

A brief survey of prevailing approaches to adaptive multimedia learning [2],[3],[4] has shown that systems with personalisation requirements have begun to be designed and developed. For example, Yang and Yang discuss the development of SMILAuthor [5] a tool based on the Synchronised Multimedia Integration Language, SMIL. SMILAuthor generates SMIL code to spatially place objects on a presentation panel using a drag-and-drop interface. It claims benefits over other multimedia authoring tools because the use of visual representation of a timeline for the placement of events making generation of SMIL referring to temporal events much simpler and less error prone than the alternative manual coding of an SMIL document. Reducing the complexity of the content creation process helps reduce the incidence of coding errors. The novel approach introduced by this paper provides features of the dynamic fragmentation of learning materials, which the SMILAuthor does not. Fragmentation facilitates the formation of better multimedia materials because the tutor supplements materials when responding to online questions from students. It also provides a future platform for a multimedia presentation system that is adaptive in real-time. The future development of HTML 5 may address some of these shortcomings [5].

Evaluation of an initial prototype provided evidence for the need to add efficient navigation for student users, so that they can access relevant learning at any point in the audio/video segment. This requires user controls and the structure of the presentation to be manifest to the student in the form of a table of contents. An evaluation is made to determine how the student users' experience is genuinely improved by using adaptation, what models are needed theoretically and what are the best practical tools to generate executable models to achieve dynamic adaptation - for example, the ontology and the student/tutor model - what form do the input and output files need to take, what is the nature of adaptation, to what extent can the current prototype interface be considered adaptive and how can the adaptations be evaluated and improved. The structure of the paper is as follows: Section 2 gives brief requirements specification for the proposed adaptive multimedia presentation system, Section 3 introduces the prototype AMPS while Section 4 looks in more detail at the media segmentation process used within AMPS. Section 5 looks at the adaptive authoring tool and its architecture. Section 6 discusses the prototype AMPS interface evaluation findings in a pilot study with degree level students and their implications. Section 7 discusses the question of automating AMPS and presents a staged implementation plan. Section 8 looks at the issues surrounding the use of ontology and develops a particular instance of network ontology and its application to AMPS. Finally, section 9 is a conclusion and discussion of future work.

II. REQUIREMENTS FOR INTERFACE DESIGN

An initial use case diagram in Figure 1 shows essential requirements for the tutor and the student. The tutor requires the minimum amount of time and effort to input learning material. Initially, this is limited to producing and uploading the audio/video segments and being able to put them into an appropriate order. An adaptive engine within the system could extract appropriate text and timeline data from these and

distribute this to the display panes of the interface to present the table of contents and supplementary text.

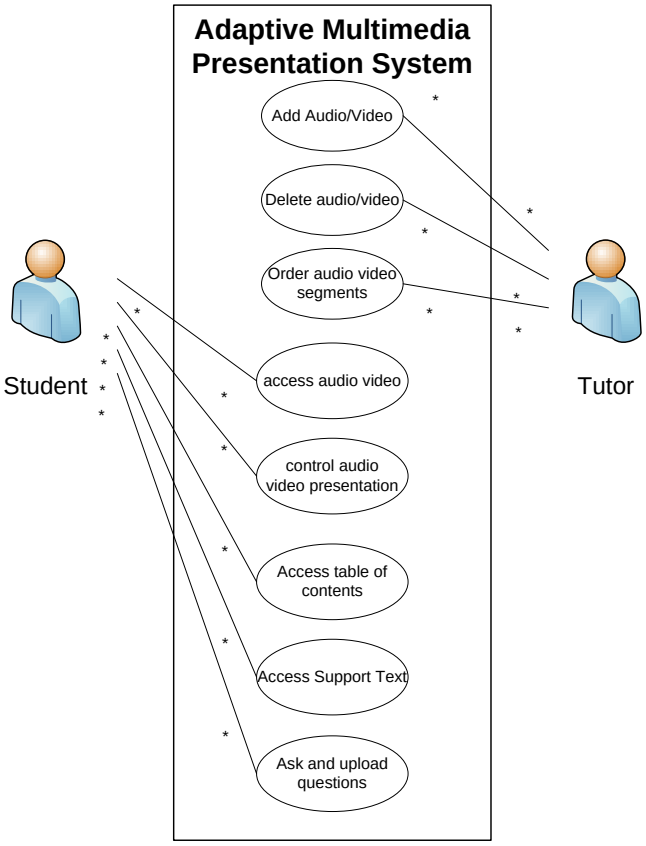


Figure 1: Use Case Diagram

The student requires access to the audio/video segments and some measure of control over their delivery. Being able to select and re-run segments is important for learning at the student's own pace. To enable this, an intuitive navigation system is required, which sequences and orders the significant points in the presentation and displays them in a table of contents with the associated supporting text. The ability to gain clarification on points not understood is also an essential requirement to effective learning. It is intended to fulfil this requirement with supplemental text and by providing access to other materials at any point in time during the presentation, as well as the ability to stop, start and jump to other points on the presentation timeline.

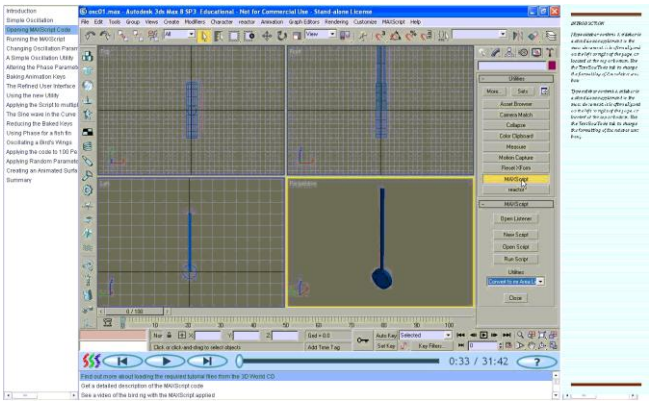


Figure 2: A proposed prototype system

A proposed prototype system shown in Figure 2 is composed of five principal parts: the main presentation panel, the table of contents panel, the supplementary text panel, the questions panel and submit button, and timeline controls for the running of the audio/video presentations.

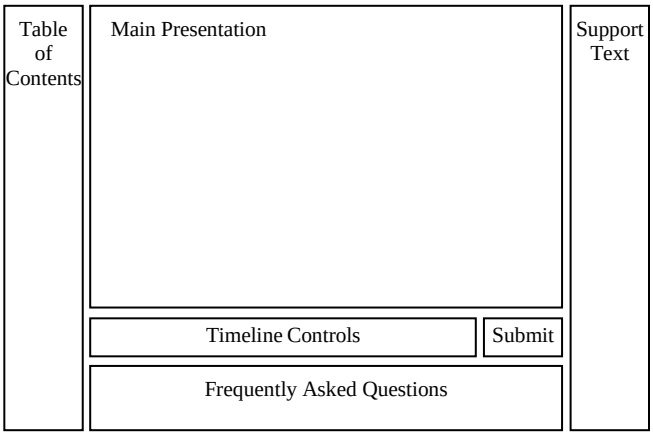


Figure 3: Schematic of the prototype system

A. *Main Presentation Area*

This contains the multimedia document which may display any combination of text, graphic, image, audio and video. It is also the primary data display area from which all supplemental information will be retrieved.

B. *Table of Contents*

The information displayed in the table of contents is automatically retrieved from the support text pane. This will require the use of intelligent knowledge storage and retrieval techniques that can structure, select and display the most useful learning material. The table of contents is presented in a hierarchical structure with a breakdown of sections. Each section title is a link to a position on the timeline, so that it is possible to jump between places within the same video/animation, or sequence of them. In later developments, additional supplementary information may be provided from the main presentation area using a variety of knowledge engineering techniques including text-based retrieval, image retrieval, video retrieval, and audio retrieval to construct a more adaptable multimedia presentation. Content-based retrieval techniques vary from one element of multimedia to another, ranging from keywords for texts, colour and texture for images and spoken words for audio, for example.

C. *Supporting Text*

Additional supporting notes will appear in this portion of the screen. This is intended to be text that assists the user's accessibility of the learning material. It may contain links to other timelines, e.g. open a new window with a duplicate set of components and its own timeline. The text displayed here may be a simple transcription of the audio part of the presentation displayed in the main area which could be retrieved by voice recognition techniques but at present are manually produced by the multimedia author.

D. FAQ and Submit Button

A facility is needed to answer with questions raised by students during a class or lecture. This external interaction requires the tutor to respond to questions put by students using the system. A proposed solution is to allow the user to invoke a text dialogue with a tutor triggered by a button.

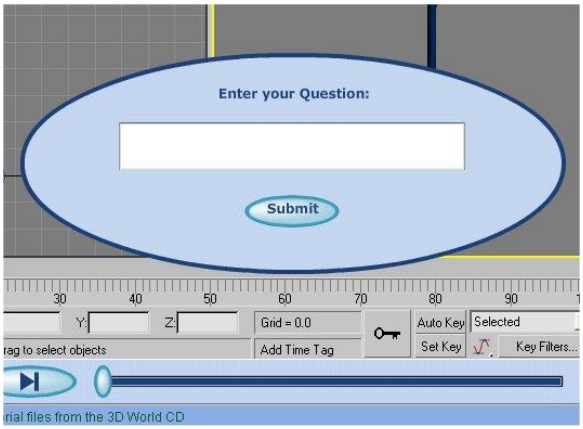


Figure 4: The submit question dialogue box

Questions are typed into the text area and submitted to the tutor. From this, an e-mail might be generated, additional automated data is added including a unique identifier for the presentation module and a timestamp. The timestamp isolates the precise time in the timeline when the question was asked, allowing the tutor to track into the presentation to see the context of the question.

The user's specific question forms the basis of feedback to alert the tutor of possible clarifications in the presentation that need additional explanation. The student's specific question is normally answered by the tutor through the creation of new video segments designed to provide clarification, which is made available to all students by insertion into the original presentation. The text question is displayed in a FAQ region when the presentation timeline reaches the point when it was asked. The audio/video segment containing the answer can then be optionally activated by selecting the question, and pausing the main presentation until the supplementary segment has been played. As more students view the modules, ask questions and gain answers, the presentation evolves by dynamically enhancing the learning resources.

E. Media Time Line with Function Buttons

The system offers temporal interaction that allows students to move through the presentation using the time bar, offering the ability to pause a presentation, to select another point in the timeline and restart the presentation, or by clicking on the table of contents to move to a different specific area. The current topic in the table of contents is highlighted in real-time so students can determine the position within the presentation, enabling students to manage their study time effectively. This type of interaction allows students to adjust the delivery of the presentation to suit their own learning style.

A graphical representation of a time line is provided, similar to a media player, representing the temporal state of the currently playing video or animation. A standard set of buttons for controlling playback will be provided. The total duration of the video/animation, or set of videos/animations which run in sequence, determines the maximum duration of the media time line.

III. THE PROTOTYPE AND ARCHITECURE

The first prototype of AMPS was developed based on the authors' understanding about how students would be expected to learn. This was felt to be a valuable initial step in personalisation [7]. The next stage is to develop the personalisation further through a new level of automated adaption and work with student end-users to gain their direct feedback of AMPS.

The prototype system shown in Figure 3 is composed of five principal parts: the main presentation panel (A), the table of contents panel (B), the supplementary text panel, (C) for the running of the audio/video presentations (D) submit button, and timeline controls and (E) the questions panel.

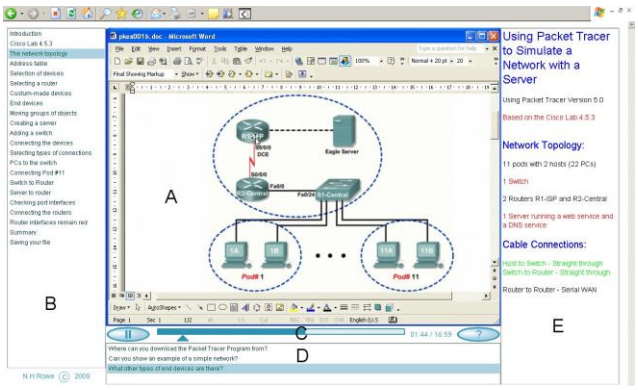


Figure 5: The AMPS prototype showing an adaptive CISCO™ learning object

The tutor builds the e-learning modules by using the segmentation architecture, which provides flexible delivery. The presentation is broken down as required into multiple segments each corresponding to an individual learning object. The selection, arrangement and linking of segments will constitute the delivery of a particular learning obect with a learning approach. In this way many segments could be played one after the other to view different aspects of the content. For example, screen shots within on-line learning materials may be followed by a video of a practical laboratory example.

IV. MEDIA SEGMENTATION

Re-segmentation of the video into smaller sections with each section carrying a single learning objective will be a direct consequence of the new user requirements. Smaller segments will further allow the personalization of the learning packages in a highly customized way and lead towards the better adaptation of AMPS.

Furthermore, in order to respond to the differing needs of students, the linking of the media segments will involve more than just a linear arrangement. The response to student interaction requires branching capabilities within the segmentation architecture [8],[9]. Segmentation allows the selection of material according to learning objectives. Students may choose to view only those segments they need to see. Additionally, the system will have the ability to respond to new students' needs not already met, or even envisioned, by currently available material. Hence the system will record and insert new media segments as required. For example, in response to a student's question for more information on a particular topic, the tutor can record a new segment and make the new segment available to all students.

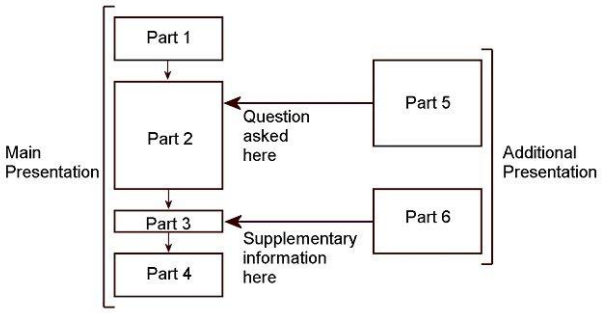


Figure 6: The timing of presentation segments

Figure 6 shows a main presentation sequence of four media segments making up a learning object. Questions asked by students at points in segment 2 and segment 3 have led to the generation of new segments 5 and 6 by the tutor which link to the main sequence at the correct points shown in the diagram.

```
<SEGMENT 1>
<SEGMENT 2>
      <SEGMENT 5>
<SEGMENT 3>
      <SEGMENT 6>
<SEGMENT 4>
```

Figure 7: Multi-level list of media segments with a hierarchical architecture

This is equivalent to a multi-level list with a hierarchical architecture. Each new segment is simply added as a subsection at the appropriate place in the list which is constructed in XML. This is rendered by the system to produce a new table of contents entry and FAQ entry. When either of these is selected, a new window opens containing the video or animation explaining the answer to the query. Each term listed needs to be linked back to a point or points in the video when the term was used and is marked as a point on the timeline. Clicking the link moves the current timeline to the associated video or animation.

B. Media Player Configuration

As a single player is required to play any module, configuration is required to activate the required resources and also to give the temporal information needed to activate the table of contents entries and the FAQs. Figure 8 shows the original XML file used for configuring the system. The file has an outer main tag. The children within this are frame rate, module ID, filename, tocInfo and questions.

The filename tag contains the files to play in sequence in the main presentation area. In this case a small presentation was played before the start, ploadv2.swf. This allowed the main presentation to be preloaded. While this was playing there was no loading delay for the main presentation.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<main>
   <framerate>8</framerate>
   <moduleid>V200134234</moduleid>
   <filename>
      <node name="ploadv2.swf"/>
      <node name="art02.swf"/>
   </filename>
   <tocInfo>
      <node label="Introduction" fileset="0" time="0.00" />
      <node label="Simple Oscillation" fileset="0" time="11.50" />
      <node label="Opening MAXScript Code" fileset="0" time="24.75" />
      <node label="Running the MAXScript" fileset="0" time="64.75" />
      <node label="Changing Oscillation Parameters" fileset="0"
time="109.38" />
      <node label="A Simple Oscillation Utility" fileset="0"
time="183.25" />

      ...

      <node label="Creating an Animated Surface" fileset="0"
time="1563.25" />
      <node label="Summary" fileset="0" time="1802.25" />
   </tocInfo>
   <questions>
      <node name="Find out more... " file="art01.swf" frame="88"/>
      <node name="Get a detailed... " file="art05.swf" frame="552"/>
      <node name="See a video of..." file="art06.swf" frame="10416"/>
      <node name="See a video of..." file="art02c.swf" frame="12416"/>
      <node name="How can this..." file="art03.swf" frame="12560"/>
      <node name="How can the oscillation..." file="art04.swf"
frame="12640"/>
   </questions>
</main>
```

Figure 8 : The XML configuration file

A prototype design architecture satisfying these initial requirements has undergone implementation and evaluation by the writers.

V. ADAPTIVE AUTHORING & RETRIEVAL TOOLS

A. Development Stages

A prototype development with staged design and implementation with increasing levels of adaptation uses two Virtual Learning Environments (VLE). One VLE is at Bournemouth and Poole College, using the open source VLE Moodle. Bournemouth University uses a localised version of the Blackboard VLE. Both VLEs have been in use for a number of years at these institutions to support peer assisted learning [10].

The development stages are:

1. Presentation player to display learning object content from VLEs
2. Authoring integration tool with manually entered meta data to create segmented learning objects
3. Authoring tool with automatic generation of meta data using adaptation/ontology techniques

4. Authoring tool with adaptive retrieval engine to automatically create multimedia content for presentations from generated ontology/metadata
5. Personalised adaptive multimedia presentation system based on students' assessment test results.

B. User Types

The user types we model are student users and academic tutors.

C. Authoring Tool

The authoring tool is shown in Figure 9. This can be evaluated by the widespread use of the system by lecturing staff and students. Success amongst staff will only occur if authoring is easy and will continue where feedback from students is widespread and positive. An authoring tool for multimedia presentations must be easy to use by non-technical teaching staff for speedy development of content [12].

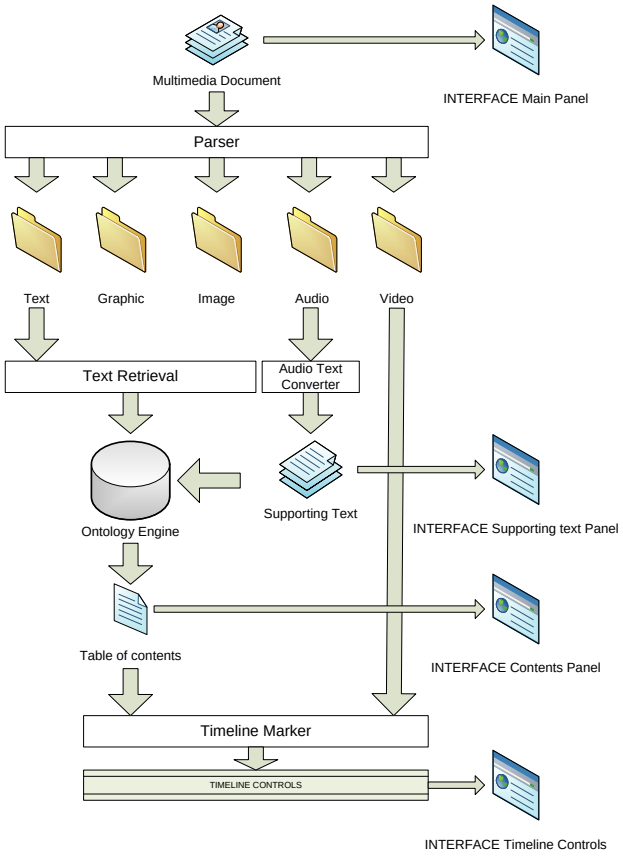


Figure 9: Architecture of an Authoring Tool

VI. INTERFACE EVALUATION FINDINGS

An online survey was used for the evaluation of the AMPS. A simple online training session teaching students how to configure a Cisco wireless router, was set up in the AMPS using the Cisco Packet Tracer [10] network simulation tool Fifty-five first year undergraduates on the honours level computing degree at Bournemouth University were recruited during normal lab classes to undertake the training through the AMPS.

Three areas of examination were covered by the questions. The first is the current level of prior knowledge of online learning environments and the subject area. The second is their experience of using the AMPS with the focus on finding out what users are trying to achieve and whether that could be made easier using new technology. And the third is the level of knowledge attained through the AMPS. Opportunity was provided for additional comments the user wished to confide.

In terms of prior knowledge, the majority of students assessed themselves as have good or excellent knowledge in the following areas:

Computer Networking 53%
Using Visual Training programmes 60%
Using VLEs 57%

Approximately a third of students (34.5%) had prior knowledge of the Cisco Packet Tracer programme and none claimed excellent knowledge.

In the area of interface use, the following features of the AMPS were rated as the most useful:

The ability to pause and rewind the presentation (83.6%)
The index list on the left of the screen (83.3%)
The ability to click on the index link to move along the video (81.4%)
The video panel in the centre (70.9%)
The time line below the video panel (70.9%).

Ease of use of the same features was rated as follows with percentages showing responses rated as very easy or easy:

The index list on the left of the screen (83.7%)
The overall interface (83.6%)
The ability to click on the index link to move along the video (81.8%)
The teaching panel in the centre (77.8%)
The time line below the video panel (76.3%)

The content of the teaching package was rated as good or excellent as follows:

How well explained was the content of the video? (83.3%)
How good was info in the index on the left? (83.4%)
How good was info in the text on the right? (49.1%)
How good was the email response (if used)? (17%) N/A (64.2%)
How good were the FAQs? (15.1%) N/A (49.1%)

Asking students to rate the most important feedback features gave the following results for very important and quite important:

Ask a question during the presentation? (68.5%)
See other student's questions and their replies? (50%)
Create your own FAQ entries? (38.9%)

We also asked what would be an acceptable response rate time for feedback enquiries:

10 minutes 34.0%
1 hour 34.0%
4 hours 8.5%
24 hours 19.1%
2-3 Days 2.1%
1 week 2.1%

In the third section, we asked students how much they actually felt they learned from the experience. The rating for those who

learned a substantial amount and those who learned quite a lot are as follows:

Networking (51%), Wireless (52.9%), Packet Tracer (62.2%)

As a result of this survey a number of findings emerged which have potential impact upon the redesign of the AMPS interface.

First, concerning the layout of the interface, not all users realized that there was a right-hand panel as this was just off the screen for some users. This issue needs to be addressed either by indicating the panel is off the screen, by reorganisation of the interface elements, or by automatic resizing of the application according to the size of the monitor used to view it. These, and possibly other options, need exploring, and user testing completed, to select the most appropriate.

Second, concerning usability, a number of students commented that the audio segment was too long at 30mins and requested shorter teaching modules. User testing will determine the ideal duration of each learning segment. In addition we have to consider if the acceptable duration of learning segments changes as the user becomes more familiar with the interface. Segmenting the video into smaller sections with each section carrying a single learning objective will be a direct consequence of the new user requirements.

VII. AN APPROACH TOWARDS AUTOMATING AMPS

A staged approach to the automation of AMPS is planned as a research programme:

1. The generation of additional video segments interweaved within the original presentation as a response to student feedback
2. The automatic generation of the content in the table of contents pane (B)
3. The automatic generation of the content in the supplementary text pane (E)
4. The segmentation of the video presentation (A) into learning objects
5. The presentation of the learning material adapted to the specific needs of the student and personalized to them.

At present only stage 1 has been realised. Figure 9 shows a model of a theoretical segmentation architecture containing a number of functions, including conversion of speech to text, a parser, the employment of an appropriate ontology engine and time line coordination to drive the AMPS.

The stages are as follows:

Stage 1: the audio component of the video clip will be parsed through a voice to text engine to transliterate the voice content of the presentation into text. This will be fed into the text panel at the right of the interface. While viewing a multimedia segment, for example, the audio of the presentation is separated and passed through a text retrieval engine which uses voice recognition principles to recover and provide text direct to the supporting text panel. Text may then be sent to the ontology engine. It uses a mixture of manual and automatically generated semantic structures that represent the conceptualisations meaningful within the context of the segment contents. The details of operation and application of ontology engines are current research areas [13] however the required outcome is the construction of the table of contents in the form of a hierarchy of terms. In the case of a 3D visualisation tool, a heading 'rendering' might be inserted into the table of contents referring to a combination of multimedia information available in the presentation system. The timeline controls links the term 'rendering' to relevant points in the multimedia content to mark the position on the timeline. The link provides a method to access the timeline of the relevant video segment or animation.

Stage 2: the generated text will be analysed by the ontology engine to construct the time-linked index. This will search the generated text for every token in the networking ontology to create a set of frequency distribution tables. Tables will be constructed for each token level within the ontology hierarchy. Level 1 tokens will form the primary analysis and will be ordered first. Level 2 will be performed within level 1, and so on. The frequency of level 1 tokens will determine how the index is structured. Boundaries of discussion will need to be detected in order to know when the topic has shifted from one domain to another. The frequency of tokens will be sufficient to name and label the domains of discussion but they will not be able to determine the boundaries. This will require a supplementary ontology dealing with concept boundary transitions and searches for the tokens that indicate these transitions.

Stage 3: The index elements will be passed through a timeline marker to set up the timeline controls. In an effort to further reduce authoring complexity, in the simplest case, metadata describing the content of segments could be created and entered manually by a domain expert at the time of media segment creation.

Stages 4 and 5 are more complex and will be considered in more detail in a later paper. However by analysing content dynamically in response to students needs in real time, the authoring tool itself would ideally be made capable of creating ontology information and using metadata. It is anticipated that the most difficult analysis would be looking for objects in videos and determining their type and meaning. However, the sports industry have analysis software for tracking the paths of moving objects such as balls on pitches and organisations involved in photography have workable face recognition systems in cameras already in use.

Beyond stages 4 and 5 we envisage a programme that will encompass the following considerations:

- The presentation system will be made adaptive through stages 2-5 and will attempt to approach real-time implementation.

- The scope of the application domain is the special case of 'Digital Networking' which will be defined through an example ontology
- The knowledge represented in the ontology will be in the form of a class diagram formatted in XML and processed in an ontology engine constructed for the purpose
- Inputs and outputs are used through a fully documented API to control input into the AMPS user interface and to personalise the learning experience
- There will need to be feedback from the user interface to the ontology engine; this will be via a fully documented API.

Future enhancements would add capability to 'see' the frames of the video, 'see' the contents of images, 'listen' to the audio, or 'read' text. The latter is the most feasible, for example by searching for key words in the text, building a semantic model of content or an ontology for the problem domain, and using this to dynamically classify and construct useful content based on the meaning of available materials.

Another challenging dimension is added when the dynamic assembly of learning objects, based on content descriptions is extended to distributed systems. An attempt is being made to apply knowledge engineering principles such as storage and retrieval of multimedia objects to the web. Practitioners are investigating these areas actively. Henze, Dolog, & Nejdl [14] have reported on the use of a logic description language, Resource Description Formats, RDF, to guide the formation of an ontology and metadata for three types of resource – domain knowledge, user knowledge and observer knowledge. These are used for personalisation of learning in a future semantic web, although the production of quality materials in an open system is problematic.

The theoretical foundations of logic languages and frameworks such as RDF hold the promise of producing practical tools and techniques for future adaptive multimedia presentation systems but they are not fully explored yet. Providing personalised on-line learning using an ontology engine to create adaptations in a closed system, let alone an open one such as the Web, is an active and complex research area [12]. Many writers are investigating competing methods and techniques to apply knowledge engineering based approaches to various application domains. This includes the use of multi-agent systems [15], neural networks or fuzzy logic filtering [16].

VIII. ONTOLOGIES, ADAPTION ENGINES AND THE API

Developing a Networking Ontology

There are a wide range of available ontology tools and models which attempt to describe knowledge domains using ontology capture and manipulation packages, e.g. Protégé Ontology Editor developed by Stanford California [17],[18]. Investigation into currently available ontology tools and models led to the decision to build our own prototype ontology of the digital computer networking knowledge domain so that it can be tightly customised to our students' particular learning domain.

However, we have tentatively concluded that these models are unlikely to contain the level of detail needed for digital networking [13].We are sceptical about the utility of constructing and executing, high-level, general-purpose ontology models in an adaptive multimedia system, especially if it is to operate in real-time [19]. This has also been supported by finding in other specialist areas such as the biomedical domain where formal ontologies can have clear limitations. Research by Shultz et al. [20] has taken the view that constructing large ontology models with many classes that range over wide topic-areas can be meaningful. More investigation is needed into this question.

Proposals to base real-time adaptation on feedback from students' responses to dynamically change the selection of menu links implies much closer integration between the ontology engine, the student's profile, or students' historical learned group profile, and the AMPS. Traditionally, two main components or sub-system types are identified in adaptive learning systems:

Case 1: Off-line recommender link mining engines, including web link miners that the tutor assists in generating adaptive presentations [12]. Output is in the form of candidate web links or menu items audited by the tutor that attempt to narrow the selections on offer to the student in the subject domain.

Case 2: Online engines that use pre-processed ontologies and combine them with individual or multiple student profiles that has been data mined, for example to find patterns that represent groups of students with given attainment levels. Outputs are recommendations for offering learning materials to these groups of students [12]. Materials presented are deemed appropriate to the student group as evaluated from outcome data such as Multiple Choice Question (MPQ) tests.

In addition to the problems already described, another drawback of Case 1 is that too many options can be presented to the tutor and the students. This makes the choices of learning materials presented to students even more problematic for a closed system such as ours. This is another reason why the writers decided to develop a restricted portion of an ontology of 'Digital Computer Networking' for use as a proof of concept model in the AMPS [21].

Figure 10 shows the contents of the Protégé ontology modelling tool [17]. This ontology was obtained using the writers' knowledge of the chosen 'Digital Computer Networking' problem domain. Knowledge of the curriculum in both academic and industrial certification courses that the writers have developed over many years of programme design and teaching of the topic to undergraduate and postgraduates at Bournemouth University was informally used to develop the ontology.

The ontology can be extracted from Protégé as an .owl file using the Manchester OWL Syntax [22], developed by the CO-

ODE project for writing OWL class expressions, or as an XML file as shown below in Figure 11 and Figure 12. This new information format is expected to be useful for analytic computational purposes as an input to the ontology engine.

A drawback of Case 2, making real-time adaptations hard to realise, is that the two sub-systems in the ontology and student model processes engine need to be combined and integrated for adaptations to be achieved in real-time, or in other words, without tutor assistance. The question therefore arises of how to model the functionality of these sub-systems and how to model the API between them to achieve close integration.

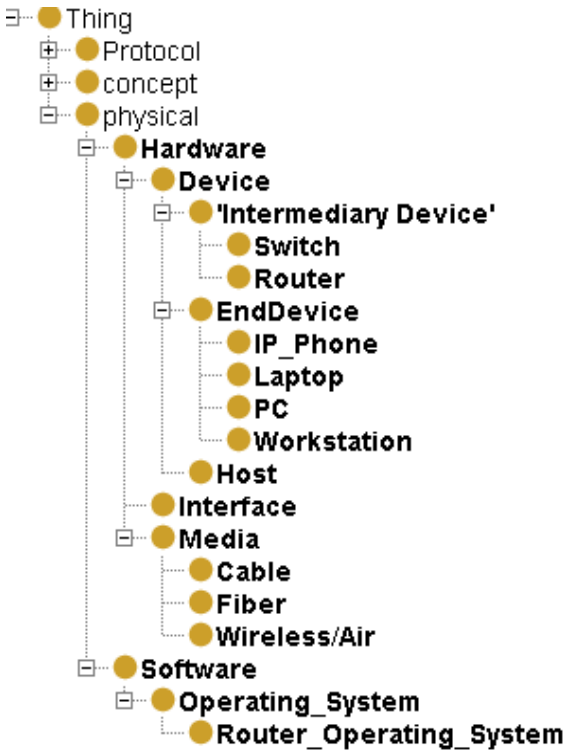


Figure 10: Sample Class Hierarchy of Digital Network Ontology Model

```
<!--
http://www.semanticweb.org/ontologies/2010/0
/OntologyOfDIgitalNetworking.owl#Device -->
<owl:Class rdf:about="#Device">
<rdfs:subClassOf rdf:resource="#Hardware"/>
</owl:Class>
```

Figure 11: Example fragment of a class from the owl file produced by Protégé

```
<SubClassOf>
<Class
URI="&OntologyOfDIgitalNetworking;Device"/>
<Class
URI="&OntologyOfDIgitalNetworking;Hardware"/>
</SubClassOf>
```

Figure 12: Example fragment of a class from the XML file produced by Protégé

The Adaptation Engine and AMPS API

Most adaptive systems contain a form of split architecture described above, but when considering the drawbacks mentioned, the writers have divided the future system into two sub-blocks and begun to develop an API between them. This allows separation and integration to be achieved simultaneously, so that the AMPS is able to perform adaptations closer to real-time.

Following Figure 13, firstly, there is an ontology engine-controller sub-block. Secondly, there is a user interface sub-block that uses standard object technology modelling methods such as model-view-controller notions, and a responsibility based class/object analysis method has been used to model the system. Messages can be bi-directional, providing feed-forward control and the feedback needed to be able to approach real-time adaptation. Thirdly, it is necessary to couple the ontology engine tightly to the user interface and to define the responsibilities of each sub-block. This requires detailed analysis and database design [23] including:

- Data about the inputs from the XML description of the ontology description tool that are processed by the ontology engine
- A diagram of user interface classes to be used to determine the optimal user interface behaviour
- Commands: these illustrate the input scenarios and can be described as a storyboard or state transition diagrams
- Messages: similarly, these explain possible output scenarios (e.g. menus, text, voice, and timeline)
- List of classes/object with functional requirements and an API will be modelled
- Choice of possible recommender algorithms [24]
- Implementation of methods
- Determination of evaluation approach will validate the effectiveness of adaptations.

Figure 13 is a first cut analysis output showing how sub-systems will collaborate and begins to locate functionality into sub-systems and conceptualise the API. The following classes have been included in the OntologyEngine sub–system:

:AdaptiveApp - Maintains abstract internal state of the UIApp object that normally would have one instance but could be many, this is so the engine takes control of the AMPS User Interface.
:ContextDependentMenuGenerator - Tells AdaptiveUIApp what to display
:OntologyEngine contains an Engine class that itself has a class structure. This will fundamentally consist of -
:OntologyEngine::Engine - The Engine class is responsible for the main control that drives the new AMPS system. The methods needed depend on the XML format (from/to the Protégé model) and the nature of the selected adaptation technique. These could be a data mining approach or a neural network approach. The effectiveness of adaptations will need to be evaluated to find the optimal choice.

OntologyEngine::AdaptiveApp - Maintains the state of the UIApp to make available to Engine. As explained above, this class is key and needed inside OntologyEngine to maintain state common to the engine and the User Interface.

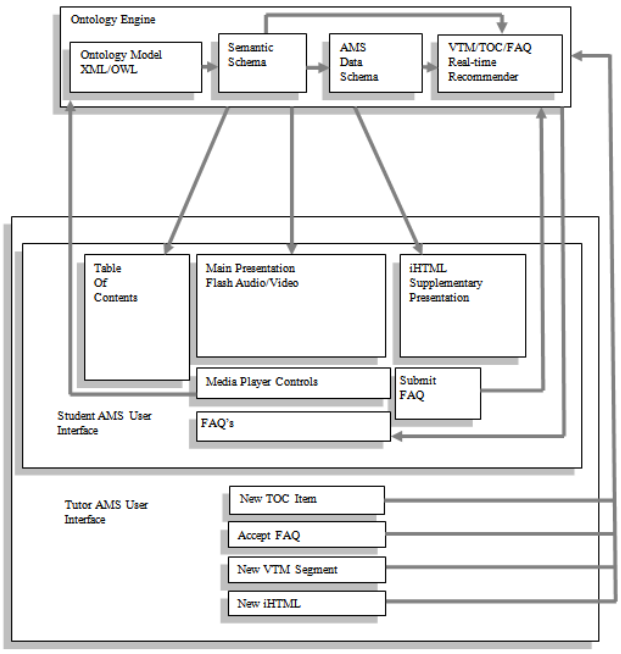


Figure 13: A Collaboration Graph of the AMPS

OntologyEngine::AdaptiveSegment – Describes sections of multiple components or segmented learning material, e.g. LEARNING OBJECT segments that can be enabled or disabled by the OntologyEngine::Engine to achieve adaptation.

Internally to the AMPS system, the OntologyEngine class itself has a structure that will need more detailed analysis than can be presented in this paper. Experiments with alternative class structures will be a critical determinant of feasibility, performance and usability. Methods and state will need to be further analysed as a guide to performance.

The design decision was taken to maintain the state of an AdaptiveUIApp class, which will mirror the AMPS state, internally to the Ontology Engine, rather than allow the User Interface to stand and operate alone as is the case with the current prototype implementation. This innovation will achieve the integration needed to approach runtime performance.

IX. CONCLUSIONS AND FUTURE WORK

An investigation has been undertaken into the requirements, underlying techniques and technologies needed for an adaptive multimedia presentation system. Research issues associated with this knowledge based approach to personalisation of learning have been outlined and begun to be explored. A process for adapting multimedia presentations through adding new content segments requested by student interaction, e.g. email, using a tree-branching sequencing system for multimedia segments has been implemented and evaluated. Content selection can make use of a form of knowledge based analysis of semantic contents of multimedia segments, dynamic generation of ontology information about video segments is stored, and retrieval proceeds dynamically according to the use of the semantic data in future forms of such a system.

Adaptation can take many forms of response to different types of stimuli. The AMPS is at present only adaptive in responding with manually produced additional video segments to the stimulus of student emails. This is considered a low level of adaption and the programme plans to increase the number of stimuli which it will automatically respond to. These stimuli need to include student prior knowledge and student ability which we call the "student signature" and will be developed further in another paper.

Feedback from students indicates the learning experience has been enhanced as evidenced by the results of the online survey presented above. Evaluation has shown that these adaptations were liked by students but do not achieve real-time adaptation in the traditional sense because of time delays. A more interactive approach to adaptation has been described and the foundation of an analysis model has been described.

Further Questions and Continuing Research

Summing up, work discussed in this paper has answered some of the research questions posed at the start of this paper, but has also indicated further questions and directions for research. The unanswered questions are:

- What is the usability level of the user interface and how can this be further improved?
- What further adaptation features are required and how are they to be evaluated?
- What model is best employed to define the interaction between the user interface and the adaptation engine?
- What is the full specification of the ontologies that are required and how is it best captured?
- How should database schemas be constructed for the AMPS for real-time extension at data and meta levels?
- How should the ontology engine structure be modelled and evaluated? Which possible data mining, or other 'smart' techniques are considered candidates for the algorithm or protocol?
- How do we determine the appropriate definition of an API, possibly by means of an IDL, between the ontology engine and the AMPS user interface presentation system?

A carefully derived student and tutor model remains to be developed more fully to automate real-time adaptations. We will address these questions in a future paper.

REFERENCES

[1] Cutts, S., Davies, P., Newell, D. and Rowe, N., (2009) Evaluation of an Adaptive Multimedia Presentation System (AMPS) with Contextual Supplemental Support Media Proceedings of the MMEDIA 2010 Conference, Athens, Greece.

[2] Adams G, Dolan M, Freed G, Hayes S, Hodge E, Kirby D, Michel T, Singer D (2009) Timed Text (TT) Authoring Format 1.0 – Distribution Format Exchange Profile (DFXP). Timed Text Working Group, W3C. Available at: http://www.w3.org/TR/2009/CR-ttaf1-dfxp-20090924/#intro [Accessed on 3 January 2010]

[3] Salton, G. & McGill, M.J. (1993) *'Introduction to Modern Information Retrieval'*. McGraw-Hill.

[4] Cristea, A.I., Smits, D., De Bra, P., (2005) 'Writing MOT, Reading AHA! - converting between an authoring and a delivery system for adaptive educational hypermedia'. A3EH Workshop, AIED'05. http://eprints.dcs.warwick.ac.uk/182/1/5-10-cristea4--.pdf pg no: 1

[5] Yang, C., & Yang Y. (2003) SMILAuthor: An Authoring System for SMIL-based Multimedia Presentations. *Multimedia Tools and Applications*, 21. 243-260 Kluwer Academic Publishers, Netherlands http://www.springerlink.com/content/t4853282j835k1g5/fulltext.pdf pg no: 245

[6] van Kesteren, A (2009) HTML 5 differences from HTML 4. W3C. Available at: http://www.w3.org/TR/html5-diff/ [Accessed on 3 January 2010]

[7] Evans, A., Fernandez, M., Vallet, D. and Castells, P., (2006) Adaptive Multimedia Access: From User Needs to Semantic Personalisation.

[8] Jun Yang, Q. L. (2007). 'Retrieval of Flash™ Movies by Semantic Content: Research Issues, Generic Framework, and Future Directions. *Multimedia Tools and Applications , 31*, 1-23.

[9] Ketter, W. Batchu, A., Berosik, G., McCready, D. (2008) *'*A Semantic Web Architecture for Advocate Agents to Determine Preferences and Facilitate Decision Making', *ACM*. http://delivery.acm.org/10.1145/1410000/1409554/a10-ketter.pdf?ip=194.66.74.22&CFID=26634273&CFTOKEN=55467158&__acm__=1308058054_947195616d66cedefaa36c67a47df46d

[10] Jeary S, Atfield-Cutts S, Phalp K, Mayes H, Bates, N,. (2010). 'Using IT Support to improve the quality of Peer Assisted Learning'. 29-31 March Inspire 2010, London, UK. http://eprints.bournemouth.ac.uk/13247/1/Inspire2010.pdf pg no: 8, 10

[11] Cisco (2009) Cisco Packet Tracer, available at: http://www.cisco.com/web/learning/netacad/course_catalog/PacketTracer.html [Accessed 29 January 2010]

[12] Shankar Vembu, M. K. (2006). 'Towards Bridging the Semantic Gap in Multimedia Annotation and Retrieval'. *1st International Workshop on Semantic Web Annotations for Multimedia (SWAMM).* http://74.125.155.132/scholar?q=cache:DN_NGBiPjcgJ:scholar.google.com/+Towards+Bridging+the+Semantic+Gap+in+Multimedia+Annotation+and+Retrieval&hl=en&as_sdt=0,5 pg no:8

[13] Frensel, D., van Harmelen, F., Horrocks, I., McGuiness, D., Patel-Schneider, P. (2001) *'OIL: An Ontology Infrastructure for the Semantic Web*', IEEE Intelligent Systems. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=920598 pg no:38

[14] Henze, N., Dolog, P. & Nejdl, W. (2004) *'Reasoning and Ontologies for Personalised E-Learning in the Semantic Web'*, Educational Technology & Society, 7(4), 82-97. http://www.ifets.info/journals/7_4/10.pdf pg no:82

[15] Ketter, W. Batchu, A., Berosik, G., McCready, D. (2008) *'*A Semantic Web Architecture for Advocate Agents to Determine Preferences and Facilitate Decision Making', *ACM*. http://delivery.acm.org/10.1145/1410000/1409554/a10-ketter.pdf?ip=194.66.74.22&CFID=26634273&CFTOKEN=55467158&__acm__=1308060281_afa4ea429e04c966c30ecc329cdb583d

[16] Teuteberg, F.(2003) 'Intelligent Agents for Documentation Categorisation and Adaptive Filtering Using a Neural Network Approach and Fuzzy Logic' in Knowledge-based Information Retrieval and Filtering from the Web (Ed. Abramowicz, W.), *Kluwer Academic.* 231-250

[17] Protégé (2009) Protégé Ontology Editor, Stanford University California, USA. http://protege.stanford.edu/ [Accessed online 28 January 2010]

[18] Stanford Centre for Biomedical Informatics Research, (2010). *Protégé - Ontology Editor and Knowledge Acquisition System*. Stanford, USA, Stanford Center for Biomedical Informatics Research supported by grant LM007885 from the United States National Library of Medicine Available from: http://protege.stanford.edu [Accessed 29 January 2010].

[19] Frensel, D., van Harmelen, F., Horrocks, I., McGuiness, D., Patel-Schneider, P. (2001) *'OIL: An Ontology Infrastructure for the Semantic Web*', IEEE Intelligent Systems. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=920598

[20] Shultz, S., Stenzhorn, H., Boeker, M., & Smith, B., (2009) Strengths and limitations of formal ontologies in the biomedical domain, RECIIS Electronic Journal of Communication, and Information and Innovation in Health, Rio de Janeiro, v.3, n.1, 31-45, Mar., 2009,

available from: http://www.reciis.cict.fiocruz.br/index.php/reciis/article/viewFile/241/253 [Accessed online 28 January 2010]

[21] Natalya F. Noy & Deborah L. McGuinness.(2009) *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford University, Stanford, CA, 94305 Available at: http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html [Accessed online 8 January 2010]

[22] Bio Health Informatics Group at The University of Manchester Department of Computer Science (2009) *The Manchester OWL Syntax developed by the CO-ODE project*. University of Manchester, UK. Available at: http://www.co-ode.org/about/ [Accessed 29 January 2010]

[23] Gruber,T. (2009) Encyclopedia of Database Systems, Ling Liu and M. Tamer Özsu (Eds.), Springer-Verlag, 2009. Available at: http://tomgruber.org/writing/ontology-definition-2007.htm [Accessed 8th January, 2010]

[24] Romero, C., Ventura, S., Delgado, J. A., De Bra, P., Salton, G. & McGill, M.J. (1993) Personalized Links Recommendation Based on Data Mining in Adaptive Educational Hypermedia Systems in *'Introduction to Modern Information Retrieval'*. McGraw-Hill.

A Scalable Solution to Deterministic Per-Flow Resource Booking

Pier Luca Montessoro Daniele De Caneva
Department of Electrical, Management and Mechanical Engineering
University of Udine, 33100 ITALY
{montessoro, decaneva}@uniud.it

Abstract— **This paper presents REBOOK, a resource reservation management algorithm for packet switching network. It provides deterministic, fast (real-time) dynamic resource allocation and release; it can be used as an engine supporting different network-oriented techniques for Quality of Service. Based on a stateful approach, it handles faults and network errors, and recovers from route changes and unexpected flows shutdown. The distributed scheme used to store flows information avoids the need of searching for entries within the routers' control memory when packets are received and guarantees constant complexity. REBOOK can be implemented in hardware and is compatible with any packet switching network. In the Internet, it can be integrated in TCP or used with UDP to make it network friendly. Moreover, a slightly extended RSVP implementation can be used as signaling and hosting protocol. A software implementation as standalone protocol has been developed to prove its effectiveness, robustness, and performances.**

Keywords- Quality of Service; network reliability; fast resource reservation; transmission rate control.

I. INTRODUCTION

The number of multimedia services on the Internet is rapidly growing, and the importance of Quality of Service (QoS) is increasing. A major problem in providing QoS guarantees in a packet switching network comes from the difficulty of handling, in routers, the state information belonging to active flows. In the core of the Internet, the conventional known techniques simply fail in keeping up-to-date the huge amount of flows information at a reasonable cost. This paper proposes a new technique that, involving end nodes' applications or edge routers or firewalls, provides constant-cost access to resource reservation information.

Quite often multimedia applications' designers prefer UDP as transport protocol because its efficiency, although those applications generally require high bandwidth and would benefit from congestion control mechanisms. Several methods have been introduced for adding QoS control mechanisms to packet switching networks, mainly based on adapting the sender's transmission rate in accordance with the network congestion state (see Section II), but typically with such approaches no QoS guarantees can be made effectively.

The proposed algorithm, that we call REBOOK [1], allows a control protocol to prevent congestion by reserving resources in advance. REBOOK is not dependant on TCP/IP protocols, as it can be used in any other packet switching network. Obviously, in the following the Internet and the TCP/IP protocol suite will be used as reference environment for its description.

REBOOK requires a hosting protocol to carry the algorithm's messages. They can be handled by a dedicated signaling protocol, like RSVP [3] or a new *ad hoc* protocol, or it can be embedded in a data transport protocol, like TCP using the options field.

In REBOOK, the node requested to send QoS-sensitive data (e.g., the sender of a multimedia stream) is responsible for resource reservation request, on the bases of the amount and type of data to be transmitted, application constraints and, possibly, SLA (Service Level Agreement) parameters. While the connection is active the amount of granted resources may be reduced to allow the activation of new flows in an almost-congested network or may be increased if switching nodes become less loaded. These events are acknowledged by the sender that will consequently adapt its transmission rate. RSVP is receiver-oriented mainly because it is designed to support singlecast and multicast flows as well. A possible implementation of REBOOK in RSVP for multicast support has been designed.

The proposed algorithm does not rely on any special network feature: it works even if only part of the network is REBOOK-aware; the resource reservation is effective even if part of a flow traverses unaware routers. There are no special requirements to routing, that can be asymmetrical (transmitting and receiving flows can follow different paths), except, obviously, its stability: in normal conditions data packets and control messages must follow the same route for the duration of the connection.

REBOOK does not rely on special hardware in routers either. Its status storage scheme allows direct access to table entries, without any hardware lookup feature, using conventional memory architecture. This makes its implementation faster and cheaper than today's typical solutions provided by hardware hashing.

Finally, REBOOK does not require any improvement in the switching fabric. No additional memory for queues and buffers nor different packet handling. On the contrary, the router architecture will drive the resource granting phase, depending on available resources left by previous reservations.

In the following, after a summary of related work in Section II, Section III discusses the scalability of the proposed stateful approach, whereas Section IV presents the algorithm itself. Section V and VI analyze implementation issues and experimental results, respectively. In Section VII, some conclusions are drawn and the future work is presented.

II. RELATED WORK

Congestion control in IP networks is a challenging issue, since it represents a critical factor for the robustness of the Internet [2]. Reservation of resources, admission control and traffic policing are among the most commonly used open-loop mechanisms to avoid congestion. On the other hand, closed-loop mechanisms rely on feedback to detect and prevent congestion [3].

A. Resource reservation and management

The IntServ architecture is an interesting implementation, because it uses RSVP [4] to reserve the resources required by the QoS-sensitive user's applications. Nevertheless, experience with real networks has revealed severe scalability problems of this architecture, due to the amount of routing and reservation information stored inside the routers.

Furthermore, RSVP implements a soft state model and uses periodic refresh control messages to manage its states, that introduces signaling overhead [5].

On the other side, the approach proposed by DiffServ is based on flow aggregates: it allows an efficient implementation inside the network. On the other hand, it conserves a statistical approach to resource provisioning and thus it does not provide any real service guarantee to any possible flow [6]. Enhancements to DiffServ may come from the Bandwidth Broker (BB) [7][8].

Cross-layer congestion control in IP networks has been addressed by XCP [9] that proposes a protocol-oriented model that puts the control state in the packets and not in the routers, with the objective of improving the scalability. Unfortunately, such schemes are hard to deploy in today's Internet [10].

Recent studies have also demonstrated that soft-state approaches coupled with explicit removal substantially improves the degree of state consistency while introducing little additional signaling message overhead [11]. This is the direction followed to design REBOOK.

B. Efficiency and path recovery

Resource reservation in packet networks is widely recognized as an essential requirement for applications, which require guaranteed minimum bandwidth, low delay, or both [12]; several fast resource reservation protocols are thus being studied and developed.

In order to increase RSVP efficiency, REDO RSVP [4] proposed a refresh mechanism made per aggregation instead of per flow, improving RSVP scalability by reducing the granularity of signaling information.

The YESSIR [13] and the LFS [14] protocols attempt to avoid complexity of RSVP by limiting its objectives and fulfilling bandwidth reservation for one-way, unicast flows.

MPLS with Traffic Engineering sets up label-switched paths (LSPs) along links with available resources: thus, ensuring that bandwidth is always available for a particular flow and avoiding congestion both in the steady state and in failure scenarios. The paradigm is that MPLS can easily address prioritized and/or guaranteed traffic along an arbitrary path, which can be independent from the underlying routing protocol. This would allow enhancing network utilization and fairness [15][16].

Interesting works on path selection algorithms are shown in literature [17][18][19], giving also some interesting solutions to failure recovery and QoS-aware fast rerouting procedures [19][20]. Nevertheless, the optimal resource allocation on all links and nodes along a reserved path when a topology change occurs is a common challenge, which relies on the trade-off between scalability and overall efficiency in resource management.

REBOOK is based on a Distributed Linked Data Structure (DLDS). DLDS are linked data structures that keep pointers to memory locations or indexes to table entries containing information stored in the routers that are very likely to be accessed in the future. When a packet whose handling requires access to that information is received, the DLDS pointer/index can be found in the packet itself and lookup procedures can be avoided. DLDS are *distributed* data structures since each pointer/index is not stored within the router it addresses, but in the end nodes, in the packets, and possibly in adjacent routers. DLDS are *dynamic*, since the collection of pointers/indexes is dynamically built and travels along the routes between the end nodes. To keep the pointers consistent in a dynamic environment, where route changes may send packets containing a pointer/index belonging to a router different by the one being traversed, a specific integrity check is adopted. Thanks to DLDS, REBOOK can improve many known techniques because it provides an efficient way to handle reservation information within the network nodes regardless the actual strategy to assign resources to the flows.

III. SCALABILITY

One widely accepted paradigm in networking is that packet switching is a scalable technique because it does not keep information for each connection (flow) traversing a node. Stateful approaches, in which some information are kept up to date for each connection in every router along the path, are not generally believed to be scalable enough to handle the increasing traffic on the Internet.

However, memory is no longer a limitation with today's technology. Provided that a tuple describing the status of a flow should contain network and transport layer addresses, some fields about the allocated resources and some control fields, we can roughly estimate a memory occupation of about 50 bytes per active flow, even for 128-bits IPv6 network addresses. In a conventional 4 GB memory the router could store information belonging to almost 86 million flows.

The open problem is the excessive computation time needed to handle the state information. As shown in the following, REBOOK solves this problem with a distributed status storage

scheme that keeps track of memory addresses in routers and allows direct access to the stored information, well avoiding the need of searching data continuously.

IV. THE ALGORITHM

The REBOOK algorithm allows resource booking/release on a packet network. It is based on several key concepts:

1) a distributed scheme for storing the resource reservation status
2) each router keeps a very limited amount of information for each flow requiring resources (as shown above, REBOOK can handle millions of flows in very high speed nodes)
3) a distributed scheme for keeping track of memory addresses (pointers) overcomes the need for searching the resource allocation tables
4) "keepalive" messages periodically signal the persistence of each flow along the path; the routers use this signaling to recover from route changes, uncommitted flow shutdowns, hosts or nodes faults, loss of REBOOK messages
5) the order of nodes traversed by a flow is kept in a distributed form and used to discover route changes.

REBOOK provides a unidirectional resource reservation, in the sense that to reserve resources in both direction of a flow two instances of the algorithm should be activated, even though some setup and shutdown messages can be glued together. REBOOK, in fact, can be easily integrated in existing transport or application layer protocols to merge end-to-end session setup and hop-by-hop resource reservation.

REBOOK works as follows: when a flow requires resource reservation, the host (or a border router that controls the QoS parameters for flows accessing the network) sends to the receiver (host or border router) a reservation request message that is normally routed along the path. Each node keeps track of the request and reserves the requested resource or the amount still available (if less than requested). Reduction in the allocated resources is written in the resource reservation request message while it traverses the router. The receiving end system sends back to the sender a resource reservation acknowledge notifying the current amount reserved. From that point on, periodical keepalive messages are sent to confirm the activity of the flow and to notify the routers along the path the current amount reserved. If the amount reserved by a router was subsequently reduced by the next ones along the path, from keepalive messages it will know how much can be released. Keepalive messages are also used to notify a booking reduction request from a overloaded router.

So far, REBOOK appears quite straightforward. The problem is that in the real world flows can suddenly disappear (due to host or router faults) or change way (dynamic routing). It is mandatory the quick identification and release of all the allocated resources that are no longer used. Even the smallest fraction of "lost" resources would produce catastrophic effects when cumulated over the days, months or even years long routers uptime. This requires continuous update of the resource allocation tables in routers, a task that so far has been believed to be too expensive, requiring special hardware architectures (e.g., Content Addressable Memories, hardware hash tables) or high computing power. The REBOOK's distributed status storage scheme overcomes these limitations with a pure software solution.

In order to make the resource reservation, the end node must communicate to the network the minimum amount of resources needed for the application to work and the maximum amount that can actually be used. How an application could know this information? And, more important, why an end node should not attempt to reserve much more than what it needs? There are several reasons, some general and others related to specific environments. The self-regulation of the end node is not new: the very basic TCP congestion control drastically reduces the transmission rate after a packet loss. Nodes share the resources trying not to overload the network. In the same way a multimedia application, on the basis of encoding and compression information, knows the peak transmission rate and the minimum bandwidth required to play the stream without interruptions. The reasons because a node should not overestimate its requirements depend on the environment. The most obvious case is a controlled environment where nodes are set-top box or computers with specific accounting and descrambling hardware for pay-per-view web TV distribution or similar multimedia services. A more general scenario is provided by ISPs that implement traffic shaping to limit high-speed peer-to-peer download; self-reducing bandwidth requests would avoid generalized slow-down of the user access and, on heavily loaded networks, connection refusal due to excessive resource reservation. In the future, speed-based accounting could become quite common; reducing the required bandwidth for non-real-time applications could help reducing access costs.

A. Rebook Messages and Data Structures

Figure 1 represents the REBOOK message fields. In order to compute the message size, IPv4 address format has been considered. Please note that for some message types ("RESET", for instance), not all the fields are needed, but in the following we will ignore this possible optimization.

A RESV message is used to start resource booking. The flow identification field is made of the tuple { *source IP, source PORT, destination IP, destination PORT* } and uniquely identifies a TCP or UDP flow. The resource reservation request is expressed by the *Resource* field, containing the minimum amount of resource needed by the flow to support the application and the maximum amount of resource that it can really use. R_{curr} is the actual resource available and reserved along the path; this value is reduced by the traversed router if the available resource is less than the maximum required. The remaining fields belong to the direct table access and fault recovery algorithms, and will be discussed later.

The reservation request carried by the RESV message is

Type	Flow ID	Resource	Path Length	Hop Counter / Reset	Resource Reservation Vector (RRV)
source to destination: RESV, KPALV UP_RESV, RL_RESV destination to source: RESV_ACK, PRL_ACK, RESET	IP_s, P_s, IP_d, P_d	R_{req}, R_{min}, R_{curr}	P_{LEN}	HOPS (N, -1 when reset)	[RAT index] $_{1..N}$ (RAT: resource allocation table)
1 byte	12 bytes	3 x 2 bytes	1 byte	1 byte	N x 3 or 4 bytes

Figure 1. REBOOK message format.

Flow ID	Resource	Path Position	Age	Local physical ports	Resource Release Request	List pointer
IP_s, P_s, IP_d, P_d	R_{req}, R_{min}, R_{curr}	P_{POS}	Age	IN, OUT	R_{rel}	list_ptr
12 bytes	3 x 2 bytes	1 byte	2 bytes	2 bytes	2 bytes	4 bytes

Figure 2. Resource Allocation Table format.

acknowledged by a RESV_ACK message, from receiver to sender. The KPALV type is used to identify the keepalive messages, from sender to receiver, while UP_RESV, RL_RESV, PRL_ACK are used to dynamically change the amount of allocated resources. The RESET message type signals that a flow's resource booking is no longer valid due to an unexpected event along the path.

Each time a new flow requires resource booking, the router creates a new *Resource Allocation Table* (RAT) entry. Its format is fixed, 29-bytes wide (IPv4, 53 bytes in IPv6), as shown in Figure 2.

The new entry is created from a free list using the *List pointer* field. The RAT entry stores the Flow ID field, the amount of requested and currently allocated resource and the system time (the *Age* field), used to identify reservations hanging due to faults or network errors. The two fields IN and OUT are router's implementation-dependent, and can be substituted by the actual internal information the router needs to handle the resources release when the flow terminates.

The field *Path Position* is the key that makes REBOOK robust at a very minimum cost and will be discussed in next Section.

The field R_{rel} allows a partial resource release when the router needs to free resources to make room for new flows.

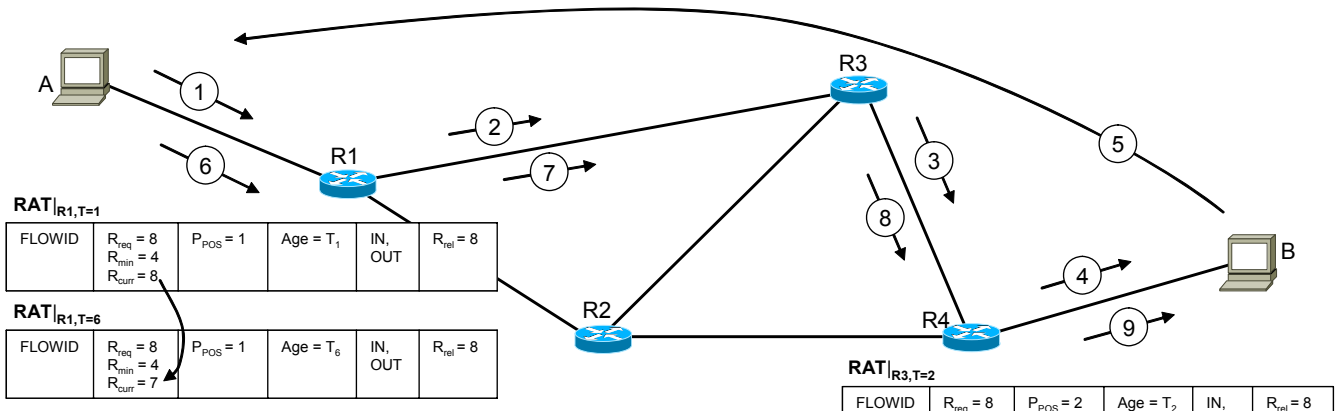
B. Resource Reservation Algorithm

Figure 3 represents a simple network in which host A must reserve a resource along the route to host B.

At the very beginning, host A sends a RESV message to the receiving host B requesting R_{req} resource (8 for instance) and stating that R_{min} (4 in the example) is the minimum acceptable. The R_{curr} field is initialized to R_{req}. The packet is travelling along the first hop, so the *Hop Counter* field is set to 1. The total length of the path (P_{len}) is still unknown and left to 0.

The router makes the resource reservation (if enough is available) and creates a RAT entry. The path position field (P_{pos}) is set to the current hop counter received in the RESV message and the *Age* field is set to the current system time. If resource availability is less than requested, the R_{curr} field in the outgoing message is set to the actual value (router R3 at step 3 in the example: $R_{curr} = 7$). Of course, previous routers in the path do not know yet that part of the reserved resources will not be used due to subsequent bottleneck: they will release them as soon as a keepalive message will be received.

Before increasing the hop counter and forwarding the RESV message to the next hop, a *Resource Reservation Vector* (RRV) entry is appended to the message, to save the index (or the memory address) of the newly created RAT entry. This index will return to the router in subsequent keepalive messages and will be used to update the RAT entry without the need of searching. Of course this approach makes the RESV messages increase their length from sender to receiver. This is limited to the RESV messages, only once for each flow, and the maximum message length for a 128 hops route (approximately the maximum length in IP, being half the maximum *Time To Live* value) would be less than 600 bytes. Anyhow, an alternative implementation for small fixed size messages is possible at the cost of a single backward message for each router traversed by the RESV message. This implementation is based on an address swapping mechanism.



1) *{"RESV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 8, P_{LEN} = 0, HOPS = 1, RRV = NULL }*

2) *{"RESV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 8, P_{LEN} = 0, HOPS = 2, RRV = [$ADDR_{R1}$] }*

3) *{"RESV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 0, HOPS = 3, RRV = [$ADDR_{R1}$, $ADDR_{R2}$]}*

4) *{"RESV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 0, HOPS = 4, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$]}*

5) *{"RESV_ACK", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 0, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

6) *{"KPALV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 1, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

7) *{"KPALV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 2, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

8) *{"KPALV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 3, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

9) *{"KPALV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 4, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

Figure 3. Resource Reservation.

When the RESV message is received by host B it is sent back to host A in the form of a RESV_ACK message. At this point both hosts know the maximum amount of resource reserved along the path, the length of the route and the list of indexes/memory addresses where the status information are stored in the routers. Host A sends the first keepalive message and will keep sending them periodically until the flow is terminated.

C. Errors, faults and route changes

Handling a keepalive message in the routers consists in three tasks. Thanks to the distributed recording of RAT indexes, the complexity of keepalive messages handling is *constant* regardless the number of active flows.

At first, an integrity check is performed, for security reasons and to identify errors, faults or route changes. The current hop counter in the KPALV message is checked, then it is used to get the RRV entry containing the index or the memory address of the flow entry in the router's RAT. The stored *Flow ID* and *Path Position* fields are compared with the *Flow ID* and the hop counter in the message. If some changes or faults occurred along the route, these values no longer match and a new resource booking reset request is signaled by setting the *Hop Counter* field to the *reset* value (-1) in the message to be forwarded.

The second task while handling a keepalive message is to partially release the allocated resources if greater than the amount reached at the end of the path by the RESV message, value now stored in the R_{curr} field. This is the case of router R1 at time 6 in Figure 3. There is no need for an explicit booking confirmation to the routers since keepalive messages will always contain this information. Unnecessary booking will be released sooner or later even if some keepalive message is lost. Frequency of keepalive messages must be set according to the required reaction time to unexpected events.

When a route change happens or a flow unexpectedly dies, keepalive messages no longer update the *Age* field in the RAT entry. Booked resources will be released thanks to a low-priority process that scans the list of active RAT entries and removes the expired ones.

D. Dynamic Resource Allocation

Sometimes a router may be required to allocate some resources, but it might happen that not enough are left. Nevertheless, it is possible that some flows have been allocated more resources than the minimum requested. Here comes the third task when handling keepalive messages: the dynamic resource reallocation. When needed, some RAT entries may be marked for resource release by setting the R_{rel} field to a value less than R_{curr}. When a keepalive message is processed for those entries, the R_{rel} value is set in the R_{req} field of the message to be forwarded, so that the receiving host B will notify the request by sending a partial release message (*PRL_ACK*) to the transmitting host A. Host A will reduce the corresponding activity and will put in subsequent keepalive messages the new R_{curr} value.

Two other message types complete the algorithm: UP_RESV and RL_RESV. UP_RESV is sent from the transmitting host to attempt a resource allocation upgrade for a flow currently active. Their handling is similar to the one for RESV messages. RL_RESV is used to release the allocated

resources under normal circumstances, that is, when the transmitting host terminates the connection and explicitly requires the resource release along the path.

E. Multicast

In order to support multicast flows, reservation and keepalive messages must be replicated at the multicast tree forks. Slightly different procedures in respect of the ones described above shall be called when a received REBOOK message is encapsulated in a network layer packet containing a multicast destination address. Obviously, the REBOOK engine must access (or receive) the multicast routing information, but this happens only during the resource reservation setup phase, as the output ports and other useful information will be stored in the *Multicast Resource Allocation Table* (MRAT) table. Unlike the RAT, in the MRAT a multicast flow is represented by a linked list of entries, each one related to a branch toward the destinations. Some fields are common to all the linked entries and an optimized implementation may collapse them into a single record: *Flow Id,* R_{req}, R_{min}, *Path Position, Age, IN*. The remaining fields are branch-dependent; in particular, keeping apart the R_{min} information allows each branch to reserve a different amount of resource without affecting the other branches. This is useful for multimedia flows featuring progressive or scalable encoding, provided that the router is able to forward the multicast packets according to the contents and to the required speed on each branch. To keep up to date the MRAT entries, in each router the keepalive messages are replicated, partially rewritten and sent along each branch.

The easiest way to setup resource reservation in a multicast environment is making each router at a multicast tree fork act as both a source (toward the subsequent nodes) and a destination (toward the preceding nodes) for REBOOK setup messages. The reservation should be driven by a signaling protocol, like RSVP [3] (next section will discuss the relations between REBOOK and RSVP). This way, the reservation request is receiver-initiated, as usual for multicast services, and REBOOK may anticipate the pointers collection using a backward pointers collection. When the first REBOOK packet sent by the receiver of the data flow reaches the sender, it contains the pointers referring to the traversed routers in reverse order. Like in RSVP, the actual reservation starts in the next step, when resource reservation messages are sent by the sender of the data flow to the receiver(s).

Dynamic joining and removal of hosts to and from the multicast services will be handled by the REBOOK agent active in each router.

V. IMPLEMENTATION ISSUES

Designing a REBOOK implementation integrated in industry-level routers requires choices depending on economical and technical constraints coming from hardware manufacturers and is therefore beyond the purpose of this paper. However, some general considerations can be drawn.

A. Impact on switching architectures

REBOOK does not require any dedicated hardware solution, even though it can be partially or fully implemented in hardware. The REBOOK management engine is required to handle reservation request/release and keepalive messages only, whose rate is orders of magnitude slower than the data traffic. The only mandatory constraint is the presence of an ingress filter to identify REBOOK messages: they must be delivered to the management process and are sent back to the switching fabric for forwarding. Depending on the architecture, this may require an additional internal buffer.

The switching architecture should not be affected by REBOOK at all. The REBOOK engine will contain parameters and rules stating the switch capabilities (port-to-port bandwidth, buffers length, etc.) and will continuously keep track of the amount of resources still available, in order to acknowledge or not the reservation requests. If all the data traffic was REBOOK compliant, resource would never be overloaded (e.g., buffer overflow) and no data packet would be lost, simply because the sender would not be allowed to transmit (or would be allowed to talk slower). In the more realistic scenario where only part of the traffic could be REBOOK compliant, priority tags or Type of Service fields could be used to identify REBOOK flows; packets belonging to such flows can be handled by separated queues and buffers to isolate them from the non-REBOOK traffic and to fulfil the resource reservation for REBOOK traffic. It is worthwhile to notice that REBOOK aims at reporting to the sender the maximum transfer rate allowed along the path to the receiver; as long as the sender respects this boundary, best-effort routers provide a QoS-like service.

B. RSVP and other hosting protocols

REBOOK can be implemented as a standalone control protocol and/or can be integrated in existing protocols, e.g., RSVP or TCP.

REBOOK can be used to improve RSVP by efficiently handling its resource reservation requests and by providing a deterministic tracking of the amount of reserved and available resources. As discussed above, the main adjustment required in respect of the algorithm described in Section IV is that the pointers collection may start during the RSVP receiver-initiated reservation request, instead of being activated by the sender. However, reservation setup and final confirmation of pre-allocated resource amounts are sender-driven: the sender knows the data flow bandwidth constraints, and the reservation messages are always processed in the same direction, thus working with symmetrical and asymmetrical routing as well.

To support multicast flows, REBOOK can work within RSVP provided that in intermediate nodes of the distribution tree the same RSVP process that merges reservation requests for multicast flows manage the entries in the MRAT described in Section IV.

Many multimedia streaming applications use TCP connections to control UDP data flows. REBOOK can be

integrated in TCP packets thus drastically reducing the need of additional packets for keepalive messages.

Anyhow, the easiest implementation is designing an *ad hoc* protocol around the REBOOK messages. This is the way followed for the experiments presented in the next section.

C. Deployment

We foresee several, non-exclusive ways to make REBOOK available at the application layer. The first and the most obvious one consists in including the REBOOK algorithm within clients and servers software for QoS-critical applications. It is quite easy to implement REBOOK as an add-on for already existing applications and let it negotiate the bandwidth required for the specific application. REBOOK could be implemented as a browser plug-in embedded within web pages of multimedia services. In this trail we are following the development of a plug-in module for the widespread multimedia application VLC.

An alternative that reduces the implementation effort is the deployment of REBOOK-aware agents at the boundary between each network and the Internet (Figure 4).

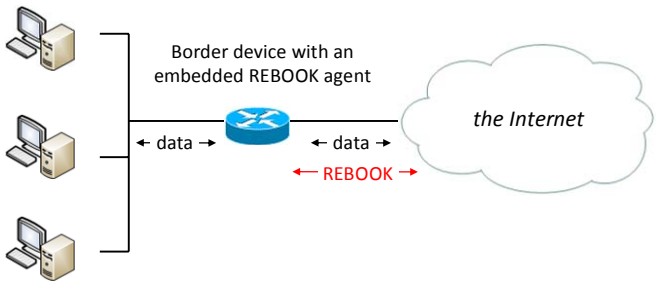


Figure 4. REBOOK agent at network edge.

Such agents will autonomously negotiate the required bandwidth with outer REBOOK aware routers on the basis of the traversed traffic, managing the QoS needs of the whole network. An agent of this kind could be easily integrated within firewalls that perform stateful packet inspection (some examples can be found in the web site of the major producers [22] [23] [24]). For a stateful firewall already keeping track of each traversed flow it is straightforward to handle REBOOK messages. Moreover, agents could be installed within boundary routers too, especially if they are software routers. A natural extension of this approach is the integration of REBOOK agents within border routers and traffic shaping appliances. In fact, flows that require resource reservation are typically characterized by an almost constant bit rate and thus they are easily identifiable and manageable by automatic resource reservation. This can be applicable even for networks invested by billions of flows per second thanks to the many algorithms for elephant flow identification present in literature [25] [26] [27]. This way the traffic shaping router will became an integrated bandwidth manager for the network; it is responsible of classifying flows, automatically issuing reservation requests only for flows that really need it, assuring fairness in bandwidth sharing and finally assuring that flows will not exceed the bandwidth reserved to each of them. Moreover, it is worthwhile to notice that REBOOK agents would prevent possible Denial-of-Service attacks based on REBOOK messages, as no reservation request could be accepted if coming directly from end nodes.

A key feature of the REBOOK algorithm is that REBOOK-aware devices and hosts may be deployed progressively. Obviously, it is impossible to deploy REBOOK or any other new protocol at the same moment throughout the entire Internet. Indeed, REBOOK might never be deployed everywhere. However, since REBOOK does not interfere with routing, unaware routers are transparently traversed by REBOOK messages. Only REBOOK-aware nodes handle the messages as described and guarantee the resource reservation. Nevertheless, as will be discussed below, REBOOK may improve network performances even in partially deployed networks.

Intermediate clouds that do not support REBOOK are not capable of performing resource reservations, so strict service guarantee cannot be made. However, if such clouds have sufficient excess capacity, they can provide acceptable and useful real-time services. The problem is now shifted to estimation of the service provided by that cloud. Depending on the real framework, this problem has different solutions (Figure 5).

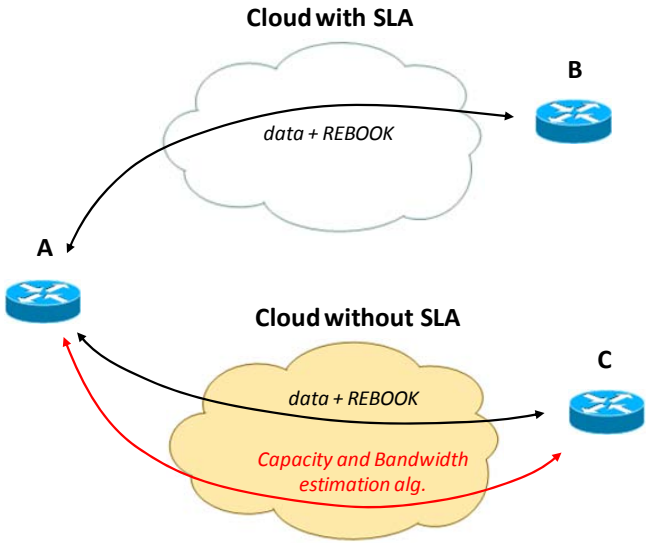


Figure 5. REBOOK and SLA support.

In the first case (A-B) the owner of the REBOOK-aware network uses the cloud to perform tunneling and has some kind of SLA with the owner of the cloud. The straightforward solution is to assign to each flow routed through the cloud a reservation compatible with that SLA. If there is no agreement between the owners (case A-C), the resource reservation control may be driven by the end systems (that monitor the data flow) instead of intermediate systems (that should communicate the available resource amounts). Non-REBOOK-aware nodes traversed by a data flow may drop packets if congested; the packet loss or keepalive messages missing rate may be monitored by the receiver; when these

events overcome a predefined threshold, PRL_ACK REBOOK messages may be generated by the receiver itself in order to reduce the reserved bandwidth and, therefore, the maximum sender's transmission rate. In extremely congested situations, when entire sequences of keepalive messages are missing, the reservation (and the flow) may be dropped. However, more complex strategies can be adopted: REBOOK-aware routers that communicate through a non-REBOOK cloud could monitor the state of the route traversing the cloud by means of one of the many bandwidth or capacity estimation algorithms (e.g., [28][29][30]) and consequently make an estimation of the resource amount available for reservation.

Additionally, it is worth noting that even if a complete Internet coverage is not possible, whole REBOOK-aware networks are absolutely plausible. In fact, multi-content providers that supply a wide range of services like IP television, Internet access and telephony are becoming popular. These providers usually manage entire networks with the need for QoS guarantees for their services. Such closed, single-owner networks could easily deploy REBOOK-aware devices.

Another situation that is gaining importance nowadays is the QoS management within overlay-networks. There is an increasing interest in their use to deliver multimedia content like video on demand, video telephony and so on. Integrating REBOOK in them is easier than interfacing a real router since these architectures are software-based and run on general-purpose servers.

D. Security

Cooperation between intermediate systems and end systems requires trust. Just like the TCP congestion-control mechanism, REBOOK works as soon as all the participating nodes behave as expected. As described at the beginning of Section IV, there is no advantage for an unfaithful node that stores or communicates invalid pointers. However, a security issue may come from DoS (Denial of Service) attacks, in two directions: invalid pointers and over-reservation requests.

REBOOK provide an intrinsic robust solution to the invalid pointers problem. The consistency check prevents the use of invalid information, but a key feature of REBOOK is that each pointer is never used by agents other than the router that stored it in a previous phase of the algorithm: pointers are not *communicated* to others, but only *stored* in a distributed data structure. Therefore, they can be encrypted and signed with symmetrical cryptography, without the need of key exchange.

The over-reservation problem may come from tampered software in end nodes. But the sender-driven reservation model makes the server and the service provider, and not the end user, responsible for correct reservation. Moreover, several possible applications are related to specific highly controlled environment such as video-on-demand distribution, where the end nodes are proprietary set-top boxes or web browsers plug-in. Lastly, nodes and REBOOK applications may be required to authenticate before starting the reservation, using some of the existing protocols and data encryption mechanisms already available.

More detailed studies will be required after the applications have been designed, but this is beyond the scope of this paper.

E. Software architecture

The REBOOK engine has been fully implemented. For the experiments, a standalone connectionless UDP-based signaling protocol has been designed and used to exchange REBOOK messages.

REBOOK implementation consists in three portable modules written in C language: router, sender and receiver. Each module, or a combination of two or all of them, can be attached to software router kernels, server programs and client applications. Thanks to several preprocessor directives a Dynamic Link Library or a linkable object code can be produced; moreover, a pure C single agent or multiple instances of C++ classes can be generated, allowing the same to be included in real routers and in simulators as well.

Figures 6, 7 and 8 show the interactions between the REBOOK modules and the host and router software.

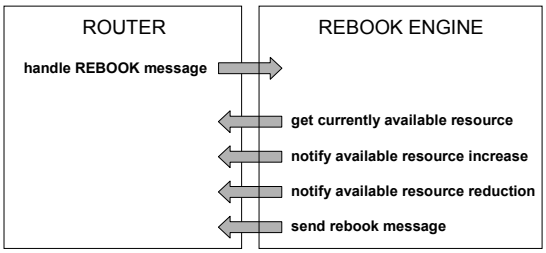


Figure 6. Router module software interface.

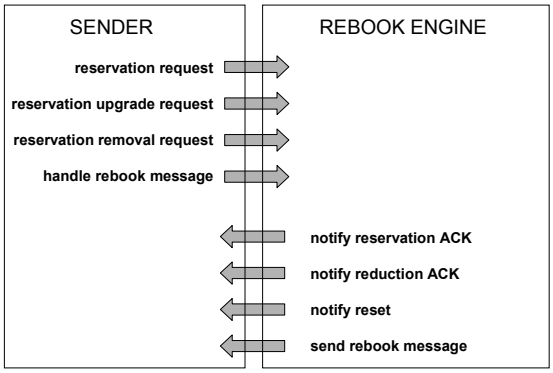


Figure 7. Sender module software interface.

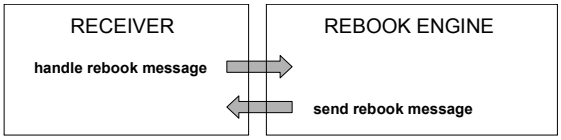


Figure 8. Receiver module software interface.

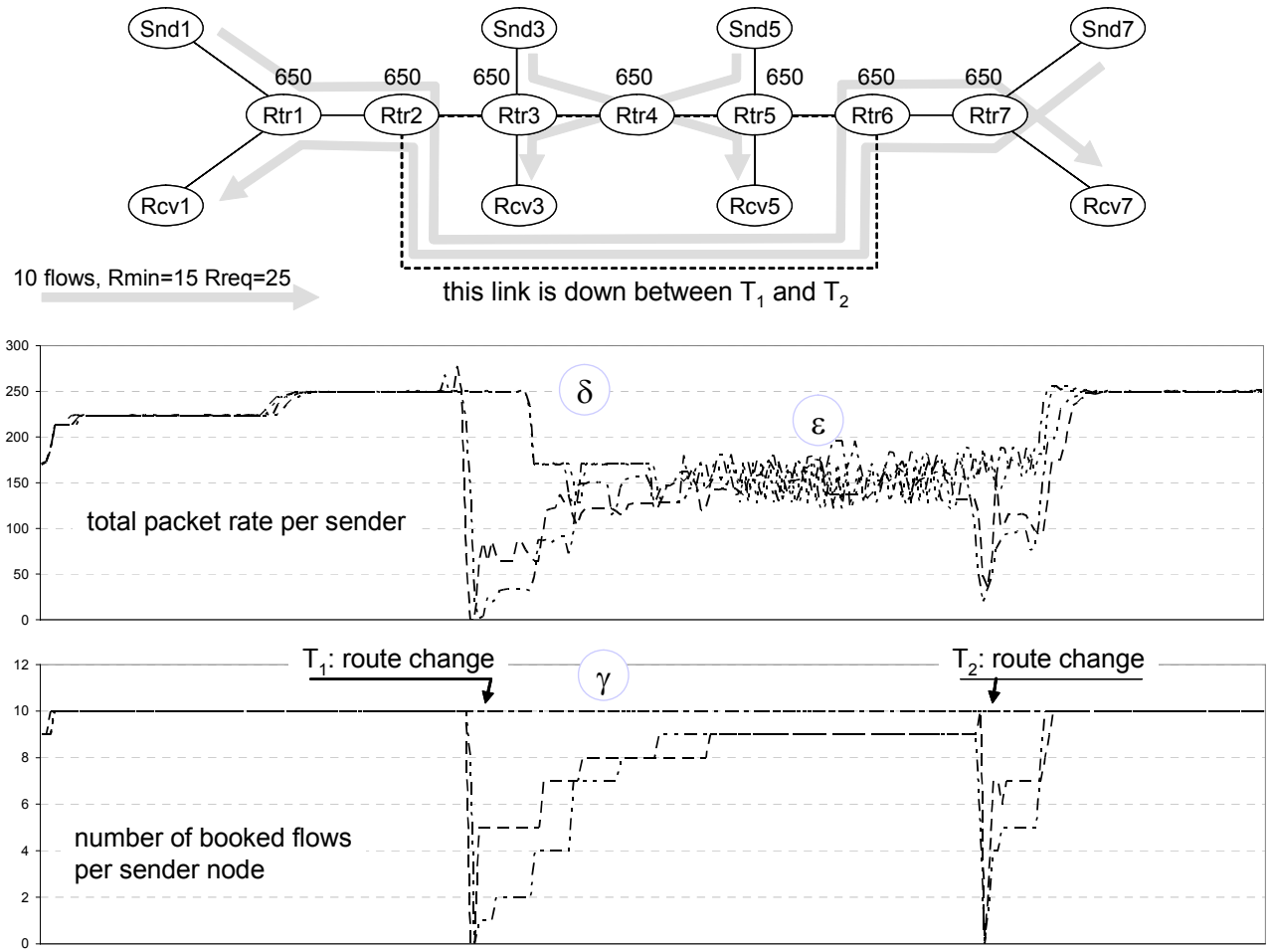


Figure 9. Congestion prevention during route change

VI. EXPERIMENTAL RESULTS

Several experiments have been designed and run to demonstrate the REBOOK robustness, performance and scalability in a real distributed environment. In order to make the experimental framework as close as possible to real, complex and possibly overloaded networks, a router emulation environment has been developed, allowing us to use computer lab Personal Computers as routers. This way, we could emulate a network with several routers actually running asynchronously, where packets can be really dropped and route changes happen asynchronously as well.

Five kinds of nodes have been developed.

Sender host: it can transmit several UDP flows towards one or more destinations; for each flow REBOOK control packets are handled to book resources along the path, and the flow transmission packet rate is modulated according with the granted bandwidth.

Receiver host: it receives and handle REBOOK messages as described in previous sections. Data packets contain a counter used by the receiver for signaling when packets are lost.

REBOOK-aware router: it is the key module, running the resource booking procedures. A periodically activated thread performs the RAT "cleanup" procedure, i.e., removes the entries expired due to repeated missing keepalive messages. Each router is assigned a maximum capacity (total number of packets per second that can be forwarded). If the traffic exceeds this value in a given time window, packets are dropped (both data packets and REBOOK messages, of course).

REBOOK-unaware router: REBOOK has been tested even in mixed environments where only some routers are REBOOK-aware. This kind of router treats REBOOK messages as normal data packets and drops packets exceeding its capacity.

Routing Control Center: this is the module that sends the routing tables to the routers. Each routing table update is acknowledge by the router and becomes immediately operative. Therefore, during route changes rules can become temporarily incoherent and packet routing errors are possible.

Figure 9 shows a 7-routers network where 40 data flows are exchanged between 4 sender-receiver pairs. Before and after the route change the network capacity is large enough to accept all the flows at full speed. When the link between routers 2 and 6 is dropped, flows from senders 1 and 7 start

competing for bandwidth in routers 3, 4 and 5. It is interesting to notice that the flows *Snd3-Rcv5* and *Snd5-Rcv3* are not affected by the route change and their booking is maintained (γ). Instead, flows *Snd1-Rcv7* and *Snd7-Rcv1* are dropped and when the senders start sending new reservation requests the routers signal to the already active senders the need of partial resource release. As soon as this happens (δ), new flows can be accepted and the system finds a new stability (ε). When the link between routers 2 and 6 is restored new reservations are made for flows *Snd1-Rcv7* and *Snd7-Rcv1* obtaining permission to send at full speed again.

Figure 10 shows the result of an experiment that demonstrates how REBOOK can be useful even in partially REBOOK-aware networks to limit the packet loss by controlling the sender's transmission rate on the basis of the packet loss rate measured at the receiver side.

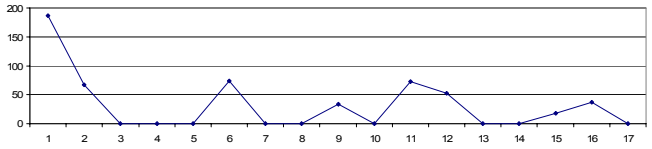


Figure 10. Packets drop control in partially REBOOK-unaware network

In the experiment, one router along the path does not support REBOOK and, in addition, it is a bottleneck due to insufficient capacity in respect of the number of active flows. The REBOOK-unaware router is transparent to REBOOK messages (the hop counter field in keepalive messages is updated by REBOOK-aware routers only). Keepalive messages and data packets are monitored by the receiver; when the number of lost packets exceeds a given threshold, the receiver autonomously generates a PR_ACK message to the sender, just like when a resource release request comes from a router. The graph of Figure 9 reports the number of dropped packet in the REBOOK-unaware router. Periodically, the sender attempts to increase the resource reservation, REBOOK-aware routers acknowledge the requests, the REBOOK-unaware router restarts dropping packets and the receiver asks the sender to reduce the transmission rate again.

Since REBOOK requires control messages in addition to data packets, a possible issue regards the traffic overhead. The experiments showed an increase in traffic load less than 1.8% with neglectable keepalive messages processing time. More precisely, on the congested network of Figure 9 we measured an increase of 5% for the seven routers total CPU time in the REBOOK-enabled run. However, since about 15% of the packets have been lost during the run without REBOOK, the average CPU time *per delivered packet* has been indeed reduced by 9%.

Routers must periodically remove obsolete entries in the RAT to free resources belonging to rerouted or dead flows. In our implementation the RAT is an array whose used and unused elements are linked in two list. To measure the actual management cost the RAT has been populated with 10.000.000 entries representing data flows with expiration times randomly distributed over 100 cleanup process scheduled activations. On a Pentium 4, 2.80 GHz, 512 MB RAM computer running Windows XP each cleanup run required, in average, approximately 100 ms CPU time.

VII. CONCLUSION AND FUTURE WORK

This paper presented an innovative algorithm for robust and deterministic resource reservation, based on a Distributed Linked Data Structure that provides direct access to flow information within the routers. This makes the algoritm computational cost *constant*, regardless the number of active flows.

Several options are available to implement the algorithm. REBOOK has been fully implemented as a standalone protocol in a software-based router emulator and has been extensively tested on heavily loaded networks with dynamically changing topologies. It demonstrated to be scalable and robust.

Many research directions can bee foreseen starting from REBOOK: investigating the integration in existing protocols, with special focus on multicast-oriented protocols and applications; REBOOK engine hardware implementation for high performance routers; REBOOK-aware firewalls, proxy servers and traffic shaping routers design; fair resource release request strategy within REBOOK-aware routers; extension to other fields like High Performance Computing (HPC) and Wireless Sensor Networks (WSN).

REFERENCES

[1] P. L. Montessoro, D. De Caneva, "A Distributed Algorithm for Efficient and Scalable Resource Booking Management," Proceedings of CTRQ 2010 Third International Conference on Communication Theory, Reliability, and Quality of Service, June 13-19, 2010 - Athens/Glyfada (Greece), pp. 128-134

[2] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," IEEE/ACM Trans. on Networking, vol. 7, no. 4, pp. 458-472, August 1999.

[3] Cui-Qing Yang and Alapati V. S. Reddy, "A Taxonomy for Congestion Control Algorithms in Packet Switching Networks," IEEE Network Magazine, Vol. 9, Number 5, July/August 1995.

[4] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, "RSVP: A new resource ReSerVation Protocol," IEEE Network, vol. 7, no. 5, pp. 8-18, September 1993.

[5] L. Mathy, D. Hutchison, S. Schmid and G. Coulson, "Improving RSVP for Better Support of Internet Multimedia Communications," Proceedings of ICMS'99, Florence, Italy, June 9-11, 1999. IEEE press, pp 102-106.

[6] W. Almesberger, S. Giordano, R. Mameli, S. Salsano and F. Salvatore, "Combining IntServ and DiffServ under Linux," Public file.

[7] S. Sohail and S. Jha, "The Survey of Bandwidth Broker," Technical Report UNSW CSE TR 0206, School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia, May 2002.

[8] Z. Zhang, Z. Duan and Y. Hou, "On Scalable Design of Bandwidth Brokers," IEICE Trans. Communications, Vol. E84-B, No.8 August 2001.

[9] D. Katabi, M. Handley and C. Rohrs, "Congestion Control for High Bandwidth Delay Product Networks," SIGCOMM'02. Pittsburgh, Pennsylvania, USA. August 19-23, 2002.

[10] Yong Xia, L. Subramanian and S. Kalynaraman, "One more bit is enough," SIGCOMM'05. Philadelphia, Pennsylvania, USA. August 22-26, 2005.

[11] Ping Ji; Zihui Ge, J. Kurose and D. Towsley, "A Comparison of Hard-State and Soft-State Signaling Protocols," Networking, IEEE/ACM Transactions on , vol.15, no.2, pp.281-294, April 2007
[12] F. Kuhns, J. Turner and S. Norden, "Lightweight Flow Setup for Wirespeed Resource Reservatio," Proceedings of the Allerton Conference on Communication, Control and Computing, 2003.
[13] Ping Pan and H. Schulzrinne, "YESSIR: A Simple Reservation Mechanism for the Internet," Communication review, vol. 29, no. 2, April 1999.
[14] F. Kuhns, J. Turner and S. Norden, "Lightweight Flow Setup for Wirespeed Resource Reservation," Proceeding of the Allerton Conference Communication, Control and Computing, 2003.
[15] I. Minei, "MPLS DiffServ-aware Traffic Engineering," Juniper Networks, 2004, White Paper
[16] V. Sharma et al., "Framework for Multi-Protocol Label Switching (MPLS)-based recovery," RFC 3469, 2003
[17] F. Rafique Dogar, Z. Uzmi and S. Baqai, "CAIP: A Restoration Routing Architecture for DiffServ Aware MPLS Traffic Engineering," 5th Workshop on Design of Reliable Communication Networks (DRCN), pp 55-60, 2005.
[18] T. Anjali, C. Scoglio, J. de Oliveira, I. Akyildiz and G. Uhl, "Optimal Policy for LSP Setup in MPLS Networks," Computer Networks, vol. 39, no. 2, pp. 165–183, June 2002.
[19] B.A. Movsichoff, C.M. Lagoa and Hao Che, "End-to-End Optimal Algorithms for Integrated QoS, Traffic Engineering, and Failure Recovery," IEEE/ACM Transactions on Networking, vol.15, no.4, pp.813-823, August 2007
[20] A. Kvalbein, A.F. Hansen, T. Cicic, S. Gjessing and O. Lysne, "Fast IP Network Recovery Using Multiple Routing Configurations," INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings , vol., no., pp.1-11, April 2006
[21] R.S. Bhatia, M. Kodialam, T.V. Lakshman and S. Sengupta, "Bandwidth Guaranteed Routing With Fast Restoration Against Link and Node Failures," Networking, IEEE/ACM Transactions on , vol.16, no.6, pp.1321-1330, Dec. 2008
[22] Juniper Networks web site, www.juniper.net, 2010.
[23] Cisco Systems web site, www.cisco.com, 2010.
[24] HP networking products and solutions web site, http://h17007.www1.hp.com/us/en/, 2010.
[25] K. Psounis, A. Ghosh, B. Prabhakar, and G. Wang. „SIFT: a Simple Algorithm for Trucking Elephant Flows and Taking Advantage of Power Laws," Proceedings of the 43rd Allerton Conference on Communication, Control, and Computing, Urbana-Champain, Illinois, USA, September 2005
[26] C. Guang, G. Jian, "Online identifying elephant flows through a scalable non-uniform sampling algorithm", Proceedings of ICCT 2008. 11th IEEE International Conference on Communication Technology, 10-12 Nov. 2008.
[27] M. Zadnik, M. Canini, A. W. Moore, D. J. Miller, W. Li, "Tracking Elephant Flows in Internet Backbone Traffic with an FPGA-based Cache," Proceedings of the 19th International Conference on Field Programmable Logic and Applications, Prague 2009.
[1] [28] V J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, L. Cottrell "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," Proc. Passive and Active Measurement Conference, La Jolla, CA, Apr. 2003
[29] S. Suthaharan, S. Kumar, "Measuring Available Bandwidth: pathChirp's Chirp Train Structure Remodeled," Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian, 7-10 Dec. 2008.
[30] R. Prasad, C. Dovrolis, M. Murray, K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," Network, IEEE , vol.17, no.6, pp. 27-35, Nov.-Dec. 2003.

An Information Flow Approach for Preventing Race Conditions: Dynamic Protection of the Linux OS

Jonathan Rouzaud-Cornabas, Patrice Clemente, Christian Toinard
ENSI de Bourges – Laboratoire d'Informatique Fondamentale d'Orléans
88 bd Lahitolle, 18020 Bourges cedex, France
{jonathan.rouzaud-cornabas,patrice.clemente,christian.toinard}
@ensi-bourges.fr

***Abstract*—In the literature, the notion of Race Condition deals with the interference between two processes A and B carrying out three interactions involving a shared object. The second interaction of the concurrent process B interleaves with the first and the third interactions of process A. Preventing Race Conditions attacks between concurrent processes is still an open problem. Many limitations remain such as preventing only Race Conditions on a file system or controlling only direct interactions with the shared context. That paper covers those various problems. First, it gives a formal definition of direct and indirect information flows at the scale of a complete operating system. Second, it proposes a general formalization of Race Conditions using those information flows. In contrast with existing formalizations, our definition is very effective and can be implemented on any operating system. Third, it provides a Mandatory Access Control that enables to prevent general Race Conditions at the scale of a whole Linux operating system. The Race Conditions can be easily expressed as a Security Properties policy. A honeypot experimentation provides a large scale evaluation of our dynamic MAC enforcement. It shows the efficiency to prevent both direct and indirect Race Conditions. Performances are encouraging us to follow our approach of a dynamic MAC for enforcing a larger range of security properties.**

***Keywords*-Computer security; data security; access control; operating systems.**

I. Introduction

A Race Condition (*RC*) happens when there is an unpredictable schedule between accesses of two users or processes to a conflicting resource, having at least one of the two users/processes modifying the shared resource. Depending on the scheduling of the accesses, the content of the resource may be unexpected. Historically, attacks based on *RC* used some unpredictable behaviors of OS (e.g., with signals) in order to modify the behavior of legitimate processes for example. The time-of-check-to-time-of-use (TOCTTOU) flaw happens when a process checks the attributes of a file, and performs operations on it assuming that the attributes have not changed when actually they have. In the literature, some authors have proposed formal description of *RC*s but provide only partial or no implementation at all. Other work deals with *RC*s within parallel programs. Remaining work focuses on protecting against *RC*s attacks but only specific ones, e.g., TOCTTOU. Many limitations remain, and in practice only a partial cover of this problem is proposed.

This paper is an extended version of [1]. In this paper we propose a global approach to deal with *RC*s at the scale of a complete computer system, at the operating system level. We provide a formal modeling of OS, and a definition of information flows related to any system call available on OS. That definition enables to formalize general *RC*s at the scale of a whole operating system. It considers a general information flow including both direct information flows and indirect information flows. General information flows are more difficult to prevent since they can involve several processes and resources using covert channels. A definition of *RC*s is given, based on three general information flows between concurrent processes. This definition is provided using a general framework for the definition of any security property related to information flows. A dynamic Mandatory Access Control is proposed to enforce the required properties of *RC*s within the Linux kernel. An experimentation is presented with several honeypot hosts exposed to the Internet including well known vulnerabilities. It shows that our MAC approach correctly prevented the *RC*s attacks. Finally, a performance study shows the real efficiency of our implementation.

The paper is organized as follows. Next section details existing work and motivates the paper. Section III gives a formal definition of the operating system and information flows. In Section Section III-D is presented our general definition of *RC*s, followed in Section III-E with the description of our linux kernel module PIGA-DYNAMIC-PROTECT. Section IV presents experiment results. Lastly, Section V gives performance evaluations, before concluding the paper, in Section V-B.

II. Related Work

In this section, we first present the state of the art of race condition. Then, we do a scope of the different protection mechanisms that have been proposed for operating system. Finally, we explain our motivation.

A. Race Condition

The authors of [2] have proposed an informal definition of a race condition:

Definition 2.1 (Informal definition of a race condition): A race condition happens when there is an unpredictable

schedule between accesses of two subjects X and Y to a shared resources I, having at least one of the two former subjects performing a write operation to the shared object between two accesses of the other subject to the shared object.
Thus, the state of the system depends on the execution order and thus is unpredictable.

It exists different types of race condition: on data [3], on signals [4] and on any shared resources. The more common case of race condition is the one associated with preliminary checks done by applications on resources. This race condition is known as time-of-check-to-time-of-use (TOCTTOU) and has been defined in [5]. Most of the filesystems are vulnerable to such attack [6], [7]. Moreover, with the increase of processors' cores within a single operating system, the detection of (and protection against) race condition becomes more and more complex. Indeed, within this scope, the detection becomes a NP-Complete problem [8].

Four main approaches are used to protect against race conditions:

- Code analysis: By analyzing the code before compiling it, it is possible to detect and find parts of code that could lead to race condition flaws [7], [9], [10], [11];
- Dynamic detection: By auditing system calls that could be used to build a race condition attack, it is possible to detect them. The most known approach is the one by Ko and Redmond [12] that detects race conditions but are not able to protect against them. Other approaches [13], [14] have used the same idea;
- Dynamic protection: When the concurrency is detected, it is possible to kill or suspend the corresponding process or system call. The authors of [15], [16], [17] have proposed this kind of approach to protect a system.
- Filesystems: It is possible to build filesystems that take into account race conditions and concurrency. Two kinds of such filesystems have been proposed: one using transactional filesystems [18], [19] and one with system calls designed to deal with concurrency [20], [21].

But, except fully transactional systems, no approach proposes a complete protection against race conditions. Transactional approaches misfit for operating systems. Moreover, the previous approaches can not express explicitly which race condition to control and do not deal with all the kinds of race condition.

B. Operating System Protection

Two main models of protection are currently used in operating systems. They are used to control the access of the users on files and others objects based on privileges. The Role Based Access Control [22] (RBAC) does not change anything in term of security, it eases the writing of security policies.

The Discretionary Access Control (DAC) is the oldest protection model. But it is always the main access control model used in modern operating system (Unix, MS Windows, Mac OSX, ...). With DAC, the privileges are set by the user who owns the object. For example, the owner of a file defines the read, write, execute privileges on its files for the other users on the system (himself, the ones within his group and the others). Multiple studies [23], [22], [24] have shown the weakness of DAC models. Indeed, it is based on the fact that users can set efficiently the permissions on their files. But any errors can lead to a security flaw. For example, if the users password file can be written by any users, anyone with an account on the operating system can change super-user password and thus obtain its privileges.

The Mandatory Access Control (MAC) allows to setup a policy that can not be change by end-users. To monitor the access between subjects and objects, Anderson [25] has proposed to use a Reference Monitor. This concept is the base of Mandatory Access Control. It was defined for Bell&LaPadula approach [26] but is now used to describe any mechanism that put the management of the security policy outside the scope of end-users. To ease the writing of security policies for operating system, it is needed to associate each entity (subject and object) with a type. This approach of Type Enforcement has been introduced in [27]. It facilitates the definition and aggregation of security model. But the drawback of the MAC approach is the difficulty to define efficient security policies. Indeed, you need to have an in-depth knowledge of how work the operating system (and its applications) and the security objectives that you want to reach. It is the cause of the low usage of the MAC approach (GRSecurity, SELinux) in modern operating systems.

Approaches like [28], [29], [30] try to bring the ease of DAC with the protection of MAC. They state that: "a good enough and usable policy is better than a very good security but hard to manage". But it is dedicated to the protection of a desktop operating system from network attacks. Moreover, they can not express security models and the quality of their protection is questionable as they does not take into account indirect flows and based their models on DAC permissions.

Others approaches [31], [32] are oriented toward the control of information flow. The purpose is to isolate users from each others. In this kind of approach, the security is enforced by a reference monitor within the system but the policy are written by the application developers. Indeed, [31] states that developers are best to know which security is needed by their applications. Moreover, the protection is done by the operating system with a reference monitor in the kernel. Thus, even if the application contains flaws, it respects the security policy. This model has been extended to GNU/Linux with the Flume [33] framework. They also proposes a limited language to describe the information flow control policy. But all those approaches can not combine flows. Thus, they are limited to describe simple security models. Moreover, they request to rewrite part of the applications' code. The authors of [34] have modelized Flume to prove that it does not contain covert channels. It does not prove the expressiveness of their language but the dependability of their system.

C. Motivation

First, we have explain the need of a mandatory approach to avoid the weakness of a discretionary control. But, current mandatory models do not explicitly take into account transitive flows and are hard to implement on a real operating system. Second, the existing protections of operating systems do not propose a language to express security objectives. Thus, they can not be used to express a complete formal security policy. The approaches based on the control of the information flows are difficult to use and do not propose a language to express security properties.

Our goal is to ease the definition of any security properties in order for the operating system to guarantee the requested properties. In this paper, we focus on race conditions but others properties have been already defined [35]. The proposed security property for protecting against race conditions needs to take into account all kinds of race conditions (filesystems, signals, any kind of data, etc). This property is formally expressed. It explicitly defines the subjects that are protected. Moreover, our language takes into account the dynamicity of a real operating system i.e. its state can change at anytime. The dynamicity is very important as we want to be able to implement our language and our protection on existing operating systems. The ability to implement our proposal is important as it is a mean to provide a large scale of real world experimentations.

III. System and security properties modeling

In order to formalize the security property that protects against *RC* attacks, in terms of activities on the operating system, let us first define the model of the target system. The first requirement is to be able to associate a unique security label (also called security context) to each system resource. A security context can be a file name or the binary name executed by a process. Our system fits well for DAC OS (GNU Linux, Microsoft Windows) or MAC ones such as SELinux whereas security contexts are special entities controlled by the kernel.

A. System dates, entities and operations

In essence, an operating system is defined by a set of entities performing operations on other entities. Those entities are referred here as 'security contexts'. Acting contexts are called subject contexts while passive ones are called object contexts.

Formally, an operating system consists of the following elements:

- A set of system dates $\mathcal{D}$.
  Any $d \in \mathcal{D}$ is a number representing a system date.
- A set of subject security contexts $\mathcal{SSC}$.
  Each $ssc \in \mathcal{SSC}$ characterizes an active entity, i.e. processes, that can perform actions, i.e. system calls.
- A set of object security contexts $\mathcal{OSC}$.
  Each $osc \in \mathcal{OSC}$ characterizes a passive entity (file, socket, ...) on which system calls can be performed.
- A set of all security contexts $\mathcal{SC} = \mathcal{SSC} \cup \mathcal{OSC}$, with $\mathcal{SSC} \cap \mathcal{OSC} = \emptyset$.
  For example, let us consider the apache webserver reading an HTML file. The apache process is identified as a subject ($/usr/bin/apache \in \mathcal{SSC}$ in a classical Linux system or $apache\_t \in \mathcal{SSC}$ in a SELinux environment) and the file is considered as an object ($/var/www \in \mathcal{OSC}$ in a classical Linux system or $var\_www\_t \in \mathcal{OSC}$ in a SELinux environment).
- A set of elementary operations $\mathcal{EO}$.
  $\mathcal{EO}$ denotes all the elementary operations, i.e. system calls, that can occur on the system (i.e. $read\_like$ and $write\_like$ operations).
- A set of interactions : $\mathcal{IT} : \mathcal{SSC} \times \mathcal{EO} \times \mathcal{SC} \times \mathcal{D} \times \mathcal{D}$.
  Each element of $\mathcal{IT}$ is thus a 5-uple that formally represents an interaction on the system. In essence, an interaction $it \in \mathcal{IT}$ represents a subject $ssc \in \mathcal{SSC}$ invoking an operation $eo \in \mathcal{EO}$ on a given context $tsc \in \mathcal{SC}$, starting at a system date s and ending at a system date e.
- A system trace T.
  The execution of an operating system can be seen as a set of invoked interactions. The executed interactions modify the OS state [17]. When we consider prevention, we work with an invocation trace. The invocation trace contains thus all tried interactions, even those which are finally not allowed to be executed. Thus, each time an interaction it_i occurs on a given system (before being allowed, in case of prevention system), the corresponding system trace becomes $T_i \leftarrow T_{i-1} \cup it_i$.

B. Information Flows

1) Direct Information Flows: In terms of information flows, when an interaction occurs (i.e. an elementary operation is performed), there is one potential consequence: that interaction can produce an information flow from one security context to another.

An *information flow* transfers some information from a security context sc_1 to a security context sc_2 using a $write\_like$ operation or to sc_1 from sc_2 using a $read\_like$ operation[1].

The formal modeling of the system is then extended with the following sets:

- A subset of $\mathcal{EO}$ of $read\_like$ operations $\mathcal{REO}$.
- A subset of $\mathcal{EO}$ of $write\_like$ operations $\mathcal{WEO}$.

Definition 3.1 (Direct Information Flow): Given a system trace T, a direct information flow from a subject context ssc performing a $write\_like$ operation to a target context tsc, starting at a system date s and ending at a system date e (formally an interaction: (ssc, weo, tsc, s, e), where $ssc \in \mathcal{SSC}, tsc \in \mathcal{SC}, weo \in \mathcal{WEO}, s \in \mathcal{D}, e \in \mathcal{WEO}$, and $s \leq e$), is denoted by: $ssc \overset{T}{\triangleright}_{[s,e]} tsc$.

Symmetrically, a direct information flow from a subject context ssc performing a $read\_like$ operation to a target

[1]To be able to decide if an interaction produces an information flow between two security contexts, we use a mapping table (not detailed here) that says for each $eo \in \mathcal{EO}$ if it can flow information – and in what direction between the two security contexts – or not.

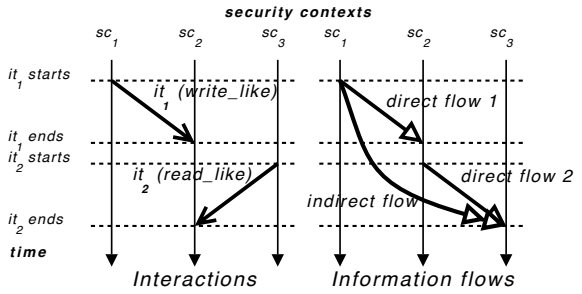


Fig. 1. r/w_like interactions and corresponding information flows.

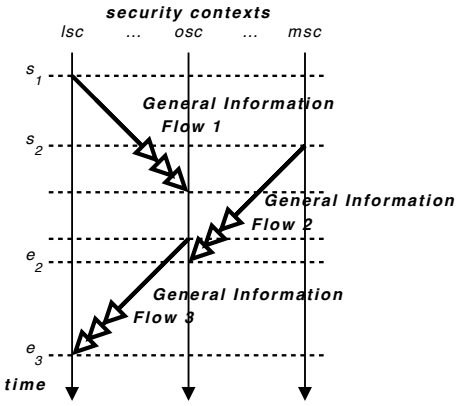


Fig. 2. Example of *RC* between lsc and msc.

context tsc starting at s and ending at e is denoted by: $tsc \overset{T}{\triangleright}_{[s,e]} ssc$.

Notice that we use the following abuses of notation when respectively s, e or both are not explicitly required: $\overset{T}{\triangleright}_{[\_,e]}$, $\overset{T}{\triangleright}_{[s,\_]}$, $\overset{T}{\triangleright}$.

2) Indirect Information Flows: As said previously, an information flow can occur directly between two security contexts. But it can also happen in many indirect ways. For example, there can exist a first flow $ssc \overset{T}{\triangleright} osc$ and then a second flow $osc \overset{T}{\triangleright} tsc$. We consider this as an *indirect information flow* from ssc to tsc. Transitively, there can theoretically be an infinite number of intermediary contexts between ssc and tsc.

We propose the following definition of indirect information flows:

Definition 3.2 (Indirect Information Flow): Given a system trace T, an indirect information flow from one context sc_1 to another context sc_k, starting at a system date s_1 and ending at a system date e_k, denoted by $sc_1 \overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[s_1,e_k]} sc_k$ (i.e., k is the number of contexts involved in the indirect flow), is formally defined by:

$$\left(\begin{array}{c} \exists k \in [3..+\infty], \forall i \in [1..k-2], sc_i \in \mathcal{SC}, \{s_i, e_i\} \in \mathcal{D}, \\ (sc_i \overset{T}{\triangleright}_{[s_i,\_]} sc_{i+1}) \wedge (sc_{i+1} \overset{T}{\triangleright}_{[\_,e_{i+1}]} sc_{i+2}) \\ \wedge (s_i \leq e_{i+1}) \end{array} \right)$$

The Figure 1 shows an example of such an indirect information flow where $k = 3$. The first interaction it_1 is a $write\_like$ operation from sc_1 to sc_2. The second interaction it_2, is a $read\_like$ operation of sc_3 to sc_2. Thus, sc_3 gets information from sc_2. So, there is an indirect information flow between sc_1 to sc_3 via the shared context sc_2.

3) General Information Flows: The two previous definitions lead to a more general definition of information flows. In essence, there exists an information flow between two security contexts *iff* there exists a direct flow or an indirect flow between those contexts.

Definition 3.3 (General Information Flow): Given a system trace T, an information flow from one context sc_1 to another context sc_k, starting at the system date s_1 and ending at the system date e_k, denoted by $sc_1 \overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[s_1,e_k]} sc_k$, is formally defined by:

$$(sc_1 \overset{T}{\triangleright}_{[s_1,e_k]} sc_k) \vee (sc_1 \overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[s_1,e_k]} sc_k)$$

Again, by abuse of notation, $\overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[\_,e]}$, $\overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[s,\_]}$, and $\overset{T}{\blacktriangleright\!\!\blacktriangleright}$ is used when s, e or both are not explicitly required.

C. Race Condition

As presented earlier, [2] gave a general definition of *RC* that we express here under our formalism in order to be able to define an enforceable security property to prevent *RC* attacks.

Definition 3.4 (General Race Condition): A *RC* happens when there is an unpredictable schedule between accesses of two security contexts sc_1 and sc_2 to a conflicting data context osc (i.e. a shared security context), having at least one of the two former contexts (e.g., sc_1) performing a $write\_like$ operation to the shared context osc between two accesses of the other context (e.g., sc_2) to the conflicting data context osc.

D. Race Condition Security property

Using the General *RC* definition above, and in order to detect or prevent *RC* based attacks, we propose to define a general Security Property for the prevention of *RC* attacks.

Security Property 3.1: A security context lsc is protected against a *RC* from another security context msc *iff* msc can not transfer information to a shared context osc between two accesses of lsc to this shared context osc.

Formally: $\text{No\_Race\_Condition}(lsc, msc, T) \Leftrightarrow$

$$\neg \left(\begin{array}{cc} \exists osc \in \mathcal{SC} \wedge & \\ \big((lsc \overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[s_1,\_]} osc) \vee (osc \overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[s_1,\_]} lsc)\big) \wedge & (1) \\ (msc \overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[s_2,e_2]} osc) \wedge & (2) \\ \big((lsc \overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[\_,e_3]} osc) \vee (osc \overset{T}{\blacktriangleright\!\!\blacktriangleright}_{[\_,e_3]} lsc)\big) \wedge & (3) \\ \big((s_1 \leq e_2) \wedge (s_2 \leq e_3)\big) & (4) \end{array} \right)$$

In the security property definition above, lsc typically represents a legitimate security context while msc represents a potentially malicious (attacker's) context.

There are many temporal situations covered by this definition, including partially or totally concurrent situations.

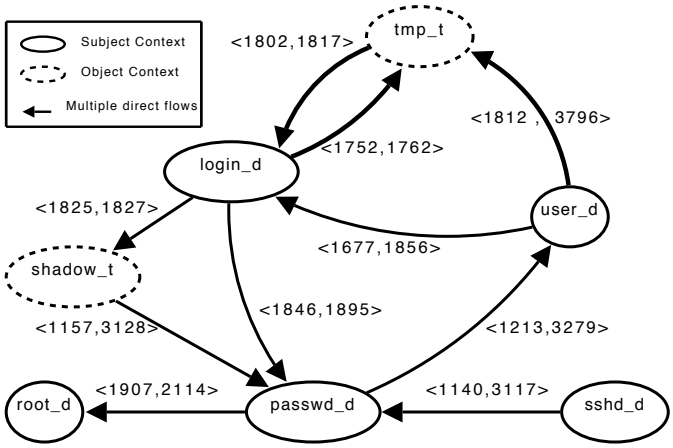


Fig. 3. IFG for the login *RC* scenario.

The Figure 2 shows one of those possible schedules. The first information flow represents an access from a legitimate context lsc to a shared context osc. It corresponds to the line #1 of the $\mathrm{No\_Race\_Condition}$ security property. The second information flow corresponds to line #2 of the property. It represents a flow from a malicious context msc to osc after or concurrently to the first flow. The third flow on the Figure corresponds to the line #3 of the property. It represents the second access from lsc to osc (partially) after or during the modification of osc by msc. The line #4 of the property expresses temporal constraints between the flow, for sequential or concurrent situations.

E. Enforcing the RC security property

In contrast to [2], this definition, based on information flows, allows us to provide an effective and efficient algorithm and related implementation for the protection against *RC* attacks.

Our solution computes an Information Flow Graph (IFG). An example of an IFG is given in Figure 3. The IFG manages the temporal relationships between the security contexts using one parameter on each edge connecting two security contexts. This parameter is a couple of system dates of the first and last occurrences of the represented information flow. Actually, those are the dates of $read\_like$ or $write\_like$ interactions.

Thus, an edge on the IFG (e.g., $tmp\_t \blacktriangleright_{<1802,1817>} login\_d$) can represent multiples flows (e.g., three flows $tmp\_t \overset{T}{\triangleright}_{[1802,1804]} login\_d$, $tmp\_t \overset{T}{\triangleright}_{[1805,1809]} login\_d$ and $tmp\_t \overset{T}{\triangleright}_{[1815,1817]} login\_d$) using only two dates (e.g., 1802 and 1817).

Obviously, using multiple direct flows instead of single direct flows clearly provides an over-approximation of the latter ones on the system. But this has the great advantage of highly reducing the IFG's size. The number of nodes is theoretically bounded by $n = \mathcal{O}(|\mathcal{SC}|)$, whereas the number of vertexes is theoretically bounded by $v = \mathcal{O}(|\mathcal{SC}| \times |\mathcal{SSC}| - 1)$. It thus can fit in memory[2].

[2]E.g., for a Gentoo Linux OS, $n < 580$ and $v < 80000$.

Within IFG, the search of an indirect flow between two security contexts is done by searching for a path between the two nodes corresponding to the security contexts. We are using a Breadth First Search (BFS) algorithm that also verifies causal dependency between each direct flows that compose the indirect one. The use of a BFS algorithm allows to have a theoretically bounded complexity of searching indirect flow due to the nature of IFG.

When we want to compare indirect flows, we need to find all the occurrence of those flows. Our indirect flow algorithm is able to enumerate all the flows between two security context. For example, on the Figure 4, the algorithm enumerates two flows from sc_1 to sc_5. Those two flows have different ending date: 1010 for the first one f_{3a} ($\#1 \rightarrow \#2 \rightarrow \#4$) and 1150 for the second one f_{3b} ($\#1 \rightarrow \#3 \rightarrow \#5$). This ability is important for the race condition algorithm as we search for temporal relation between flows. If we have only one of the two occurrences of the flow, we can miss a race condition. For example, if we have only f_{3a} that ends at 1010 and we want to compare it with a flow f_2 that start at 1025, the condition $s_2 \leq e_3$ will be false. But if we have the two flows f_{3a} and f_{3b}, the condition will be true as $start(f_2) \leq end(f_{3b})$.

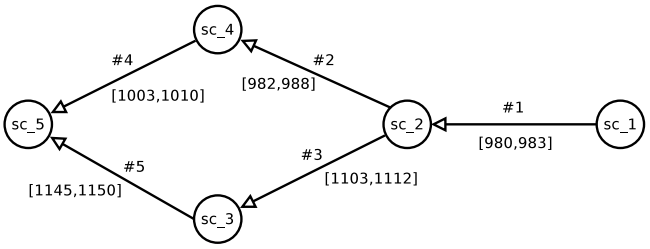


Fig. 4. Search all the occurrences of a flow between sc_1 and sc_5 in the IFG

Using the IFG and the general information flow algorithm, we are able to compile the property 3.1 into an algorithm. The algorithm 1 allows to detect any race condition between a legal entity lsc and a malicious one msc. This algorithm uses a function that returns every flows corresponding to three general flows between lsc and msc. Indeed, we need to verify the temporal relationships between those flows as described in the property 3.1 ($(s_1 \leq e_2) \wedge (s_2 \leq e_3)$). In the algorithm, it_1 is the current interaction i.e. the interaction that is in the authorization process. The algorithm 1 is a variation of the property 3.1 with the third flow between the shared object osc and the legal security context lsc must be a direct flow. In practice, it is usually the case. Moreover, this algorithm goes further than previous approaches as it takes into account indirect flows in two case out of three. This variation of the property 3.1 allows to reduce the overall complexity of the algorithm. Indeed, we only use the indirect flow algorithm two times instead of three.

F. Protecting an Operating System

This section presents PIGA-DYN-PROTECT i.e. our dynamic approach to guarantee the security properties expressed

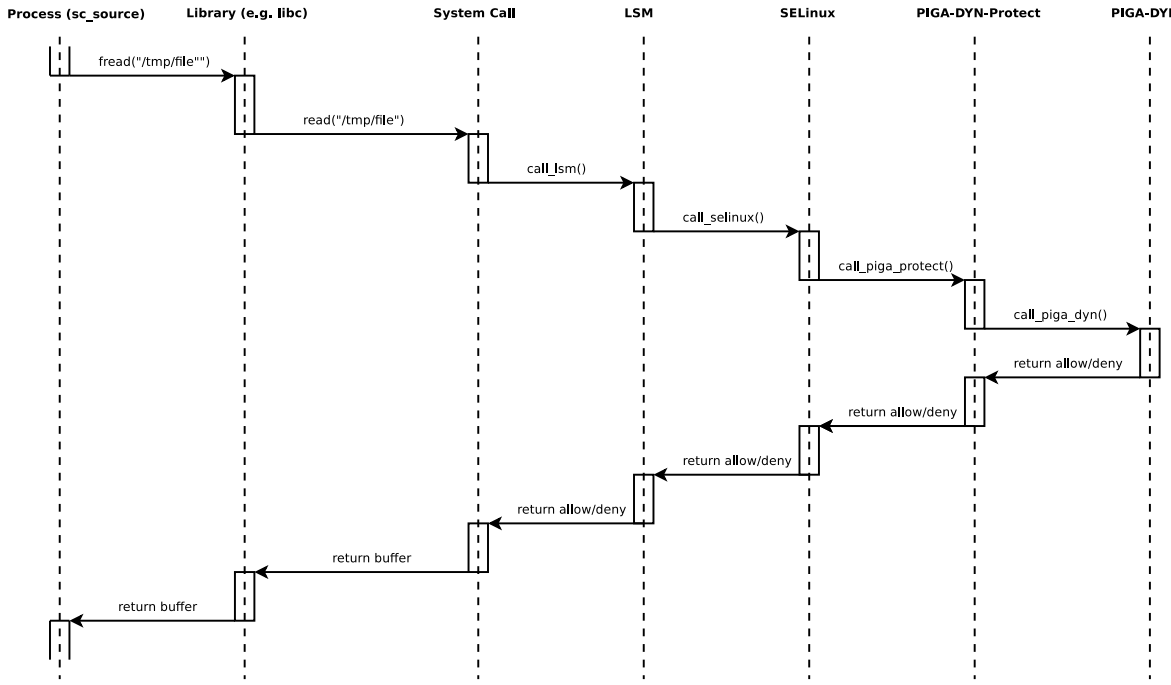


Fig. 5. Global architecture of our protection system

Algorithm 1 Algorithm to detect race condition from lcs to mcs : $No\_Race\_Condition(lsc, msc, it_1, IFG)$

Require: lsc, msc, it_1, IFG
  if $op(it_1) \in \mathcal{WEO}$ **or** $op(it_1) \in \mathcal{REO}$ **then**
    $iflow_1 = lcs \overset{T}{\rightarrowtail} dest(it_1)$
    $iflow_2 = mcs \overset{T}{\rightarrowtail} dest(it_1)$
    $list\_flow_1 = searchAllPath(iflow_1, IFG)$
    $list\_flow_2 = searchAllPath(iflow_2, IFG)$
    if $list\_flow_1 \neq$ NULL **and** $list\_flow_2 \neq$ NULL **then**
      for all $flow_1 \in list\_flow_1$ **do**
        for all $flow_2 \in list\_flow_2$ **do**
          if $start(flow_1) \leq end(flow_2)$ **then**
            return FALSE
          end if
        end for
      end for
    end if
  end if
  return TRUE

with our language. PIGA-DYN-PROTECT enforces a policy that is a set of security properties. Our architecture reuses SELinux security contexts. SELinux provides a context for all processes, users and system resources. In contrast with approaches like [30], [36], [37], [17], [16], we do not require a unique context for each entity e.g., $/tmp/file1$ and $/tmp/file2$ are grouped in a common label tmp_t. However, the SELinux policy is not required. Our solution is a satisfying approach since defining a consistent SELinux policy is a complex task. On the other hand, [38], [39] show that a SELinux protection policy can still present around a million of attack vectors. So, adding a protection against *RC* over SELinux security contexts is very efficient.

Figure 5 describes the global architecture of our protection. This architecture is composed of several components. We choose to illustrate each component based on the execution of a system call. In this example, an application calls a function `fread`. The library C (`libc`) provides this function. Then, the library calls the `read` system calls to communicate with the kernel of the operating system. This system call allows to execute a read in kernel space. Indeed, it is the only way to perform an input/output access to transfer information from or to a physical resources like a hard drive. Before this input/output operation, the kernel checks the discretionary permissions such as read, write and execution rights. If the operation is allowed by DAC then it calls the Linux Security Module (LSM). LSM is used to plug new security modules that hook system calls. LSM is used to call the SELinux module. Then, this module applies Type Enforcement on the system i.e. it sets a context for each entity of the operating system. PIGA-DYN-Protect is called by SELinux and collects all the information needed to build the trace of the system calls. This trace is sent to PIGA-DYN that uses it to build the IFG. It computes that graph to verify if the current system call goes against a security property. The decision is returned to PIGA-DYN-Protect then to SELinux. Finally, LSM receives the decision and allows or denies the system call.

In practice, for our *RC* property, the decision is taken at the last step of an attack attempt i.e. on the third system call. Thus, our solution prevents efficiently the third flow of a *RC* attack.

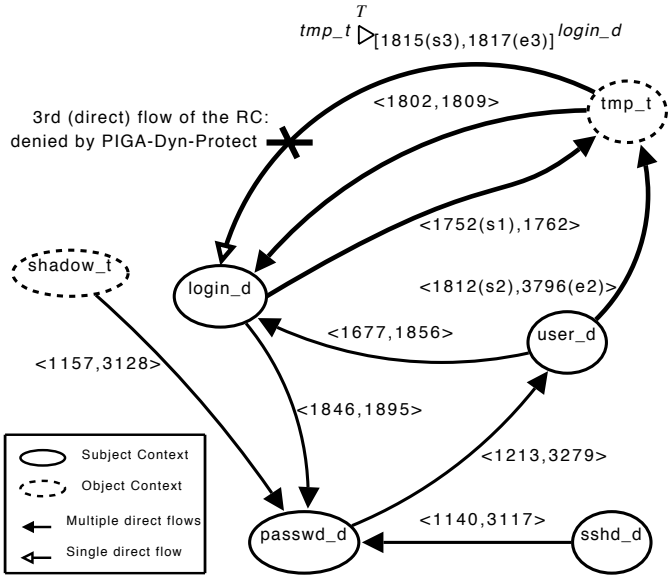


Fig. 6. IFG for the real protection against the login *RC* attack.

Moreover, our solution protects against unknown attacks (e.g., 0-Day attacks) and various covert channels associated with indirect flows.

G. Example of prevention of a Race Condition

Let us give an example of the No_Race_Condition property. The property: No_Race_Condition($login\_u$:$login\_r$:$login\_d$, $user\_u$:$user\_r$:$user\_d$, T) prevents a user process ($user\_u$:$user\_r$:$user\_d$) to interfere with the login process ($login\_u : login\_r : login\_d$) via a *RC* attack.

The IFG given in Figure 3 describes the so-called 'login *RC*' attack scenario. Thus, the first flow of the considered property can be seen as the direct flow $login\_d \overset{T}{\triangleright}_{[1752,1762]} tmp\_t$. The second flow is a direct flow $user\_d \overset{T}{\triangleright}_{[1812,3796]} tmp\_t$. The third flow is the direct flow $tmp\_t \overset{T}{\triangleright}_{[1802,1817]} user\_d$. Using this IFG, we are able to block the system call corresponding to the third flow at the kernel level. It thus avoids to exploit the login's vulnerability via a *RC* attack.

IV. Real World Experiment

To make real world experimentation of our solution, we have setup a high-interaction honeypot. It contains services with exploitable flaws to ease the attacks. A virtual machine provides the different services. During six month, we experimented many instances of the No_Race_Condition security property. Let us give the results for two of them. The first one was the protection against the Login *RC* attack (already mentioned) and the second was the protection against the PhpBB *RC* attack. Others instances, not presented here, cover attacks on filesystems, network services and shell scripts.

As any mandatory protection, PIGA-Dyn-Protect is an over-approximation of the attacks thus it can generate false positives. On the other side, the false negatives are due to a bad definition of the property or an incorrect configuration of it. Thus, it can be fixed through a new definition for the property or another configuration of it.

A. The Login attack RC to gain root privilege

To welcome attackers for the first evaluated attack: the 'Login Race Condition', we setup SSH servers on three machines of our honeypot. Those servers accepted attackers with the automatic creation of accounts when couples of login/password were tried.

The attack scenario is the following. An attacker ($user\_d$) connects to the SSH server (ssh_d). Then, the attacker uses the authentication service ($passwd\_d$) and gains a user session ($user\_d$) on the machine. He uses his session to execute the login command ($login\_d$). Then, he exploits the login's vulnerability by changing a value stored in the temporary file (tmp_t). When the login process comes back later to read the (modified) value, that allows a privilege escalation of $user\_d$ to $login\_d$ that has root privileges. The attacker uses this privilege escalation to modify the shadow passwords file $shadow\_t$ and especially the root password. The attacker then uses the login command again to open a root session ($root\_d$) with the new root password. This attack only involves direct information flows.

Let us consider a system without our protection solution in order to detail the 'Login *RC*' attack scenario. The IFG given in Figure 3 corresponds to the violation of a No_Race_Condition security property. In this figure, the temporal constraints (line #4) of the No_Race_Condition security property between $user\_d$ and $login\_d$ are true: $(1752(s_1) < 3796(e_2)) \wedge (1812(s_2) < 1817(e_3))$.

Our PIGA-Dyn-Protect module cancels the last interaction of the third flow in order to avoid the *RC* success. In the example, PIGA-Dyn-Protect denies the execution of the interaction at system date 1817. Thus PIGA-Dyn-Protect cancels the interaction corresponding to the (single) direct flow $tmp\_t \overset{T}{\triangleright}_{[1815,1817]} login\_d$: the third flow violating the No_RC property does not appear in the real IFG, as shown in Figure 6.

As shown in the figure 8, we were able to monitor multiple attacks during the six months experimentation. Moreover, through SELinux logs analysis [40], we were able to validate that all the attacks were detected by our solution. With the protection mode, all the attacks were blocked and no attacker was able to gain root privilege.

B. PhpBB RC for Remote Shell Execution

We also experimented another kind of *RC* attacks: the PhpBB *RC* attack that can lead to a 'Remote Shell Execution'. In order to collect attacks, we build a fake PhpBB forum. We also advertised about *cgi scripts* (bash and binary) execution.

For this experiment, the security property against *RC* was configured as the following: No_Race_Condition($apache\_u$:$apache\_r$:$apache\_d$, $apache\_u$:$apache\_r$:$phpbb\_d$, T). Compared to the login *RC*,

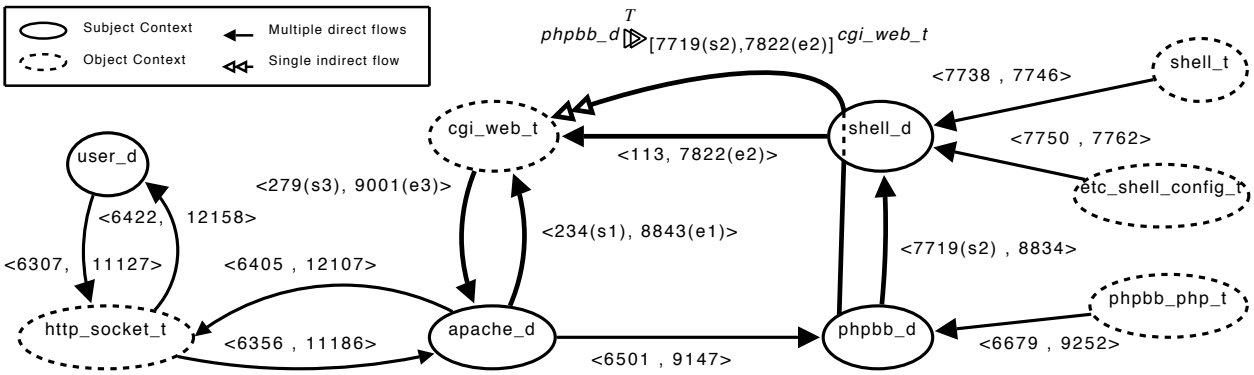


Fig. 7. IFG for the phpBB *RC* attack scenario.

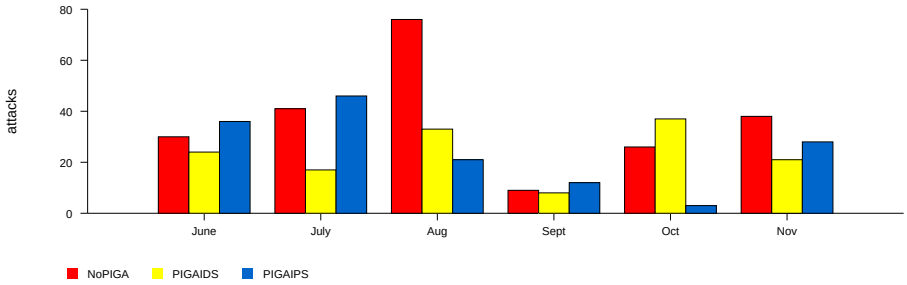


Fig. 8. Number of attacks per month per machine for the login race condition

this attack is a more complex one. The Figure 7 describes the PhpBB attack scenario. First, the attacker ($user\_d$) has to connect to the apache server ($apache\_d$) through a socket ($http\_socket\_t$) and accesses to the PhpBB forum ($phpbb\_d$). Then, the attacker uses the remote execution exploit in the PhpBB forum to execute a shell ($shell\_d$). He uses the shell to modify the cgi scripts (cgi_web_t). In summary, this *RC* based PhpBB attack involves direct and indirect information flows. The first flow of the attack is $apache\_d \overset{T}{\triangleright}_{[234,8843]} cgi\_web\_t$. The second one is indeed an indirect flow: $phpbb\_d \overset{T}{\blacktriangleright}_{[7719,7822]} cgi\_web\_t$ (via $shell\_d$). The third flow is a direct one: $cgi\_web\_t \overset{T}{\triangleright}_{[279,9001]} apache\_d$.

In the *RC* PhpBB attack scenario given in Figure 7, the temporal relations between flows violate the temporal constraints (line #4) of the No_Race_Condition security property between $apache\_d$ and $phpbb\_d$: $(234(s_1) < 7822(e_2)) \wedge (7719(s_2) < 9001(e_3))$.

Again, our PIGA-DYN-PROTECT module denies the last interaction of the third flow. It thus forbids the execution of the corresponding interaction at the system date 9001.

As shown in the figure 9, we were able to monitor multiple attacks during the six months period of our experimentation. Same as login RC attacks, all the phpBB attacks were detected and blocked.

V. EFFICIENCY

A. Completeness

To evaluate the correctness of our approach, we configured our honeypot hosts with PIGA-DYN-PROTECT in both detection and prevention mode. That way, we could verify that every detected attack was prevented or not. During the six months of experiment, we detected 146 instances of the login *RC* attack and 574 instances of the PhpBB *RC* attack. All attacks were blocked by our protection mechanism.

B. Performances

In order to evaluate the performances of our solution, several benchmarks are proposed for three different configurations:

- a classical Linux system with DAC and SELinux TE.
- a Linux system with DAC and SELinux TE with PIGA-DYN in analysis mode for detecting the violation of the required security properties (PIGA-IDS).
- a Linux system with DAC and SELinux TE with PIGA-DYN in protection mode for enforcing the required security properties (PIGA-IPS).

The hardware configuration was the same, a Pentium-4 3Ghz with 1Gb of memory.

We use lmbench [41] suite running on these three machines to measure bandwidth and latency. Lmbench attempts to measure performance bottlenecks in a wide range of system applications. These bottlenecks have been identified, isolated

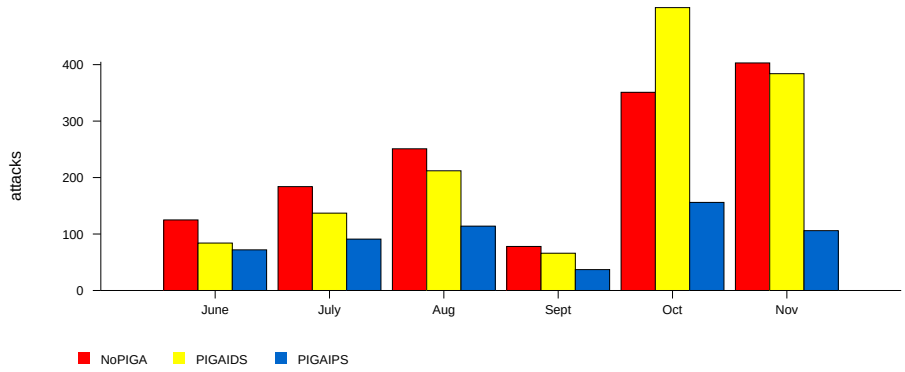


Fig. 9. Number of attacks per month per machine

Operation	Description
memory read	Measuring time to read x byte word from memory
memory write	Measuring time to write x byte word to memory
memory read/write	Measuring time to read an write x byte word to memory

TABLE I
MEMORY OPERATIONS FROM LMBENCH

and reproduced in a set of microbenchmarks that measures the system latency and bandwidth of data movement. First, we focus on the memory subsystem and measure bandwidth with various memory operation listed in the table I.

As shown in the Figure 10, the difference between the three different configurations is small. With data blocks larger than 512Kb, the three configurations have almost the same performances. With data blocks smaller than 512Kb, the overhead due to our security component is unnoticeable. In the worst case, like memory read/write, the overhead is 5%. Consequently, we can state that our security component has little to no influence on data copy to and from the memory.

In a second time, we used lmbench to measure latency in five different parts of the operating system:

- Process: it creates four different types of process and evaluates the time it takes 1) to invoke a procedure, fork a process and close it, 2) fork a process and invoke the execve system call and 3) fork a process and invoke an interactive shell.
- Context switching: it measures the context switching time for a number of processes of a given size (in Kb). The processes are all connected through a ring of Unix Pipe where each process reads its input pipe, does some work and writes in its output pipe i.e. the input pipe of the next process.
- System call: it measures the time to write one byte to $/dev/null$. It permits to evaluate the interaction time with the system.
- Filesystem: it measures the time to create and delete files with a size between 0 and 10Kb.
- Network: it measures the time taken to make a HTTP request ($GET/$) on a LAN and a WAN HTTP server.

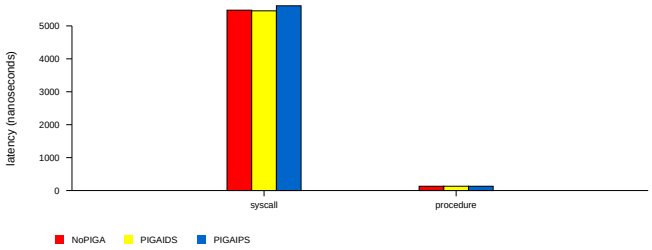


Fig. 11. Syscall and Procedure Latency for the three operating systems

Figure 11 displays, on the first set of columns, the latency (in nanoseconds) to invoke a system call. The operating system without our solution and with our solution in analysis mode have the same latency. The overhead due to our solution in protection mode is minimal (2% on average). Thus, our solution has little to no influence on the system call latency. The second set of columns shows the latency when invoking a function in a program. The three operating systems have the same latency. Our solution does not change the procedure call latency. Indeed, the call of a procedure inside a program does not require to compute any authorization.

Figure 12 displays the average time (in microseconds) to create and destroy three types of process: first one is just a creation and a deletion, the second one is a creation then an execution (execve), the third one is a creation then the invocation of a shell (/bin/sh). The overhead of our two solutions is high. Indeed, thousands of fork system calls, within few seconds, create thousands of path searches. The overhead is about 300% when using our solution in analysis mode and

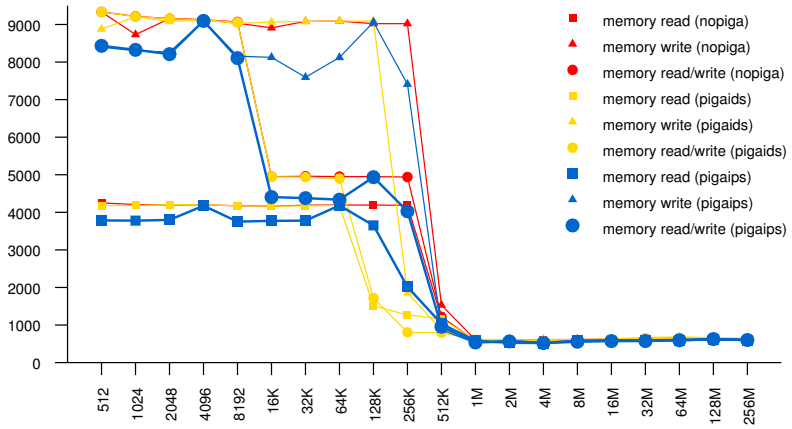


Fig. 10. Memory bandwidth performance

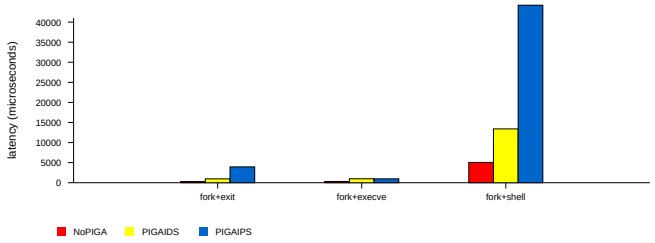


Fig. 12. Fork Latency for the three operating systems

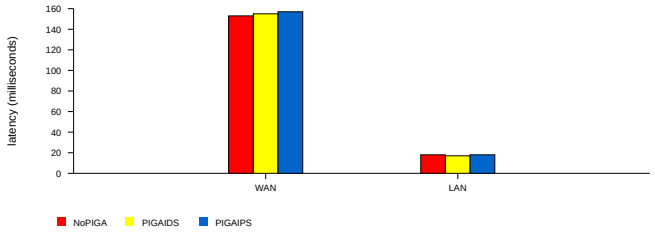


Fig. 13. Network Latency for the three operating systems

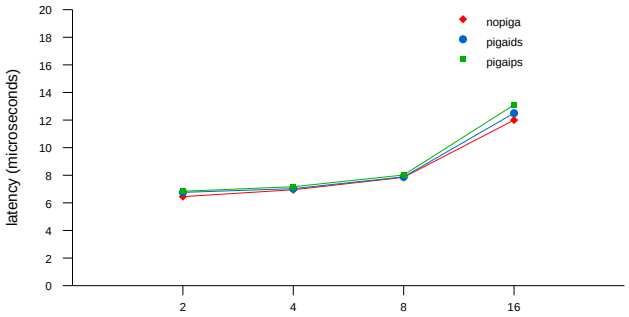


Fig. 14. Context Switch Latency for the three operating systems

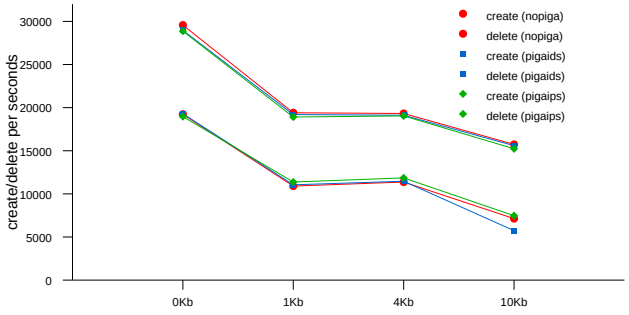


Fig. 15. Filesystem Latency for the three operating systems

800% in protection mode. This overhead can be reduced by pre-computing the path on the IFG, such approaches are under development. However, thousands of forks within a limited time really is unusual. So, this overhead is not relevant of a common usage.

As one can see on the Figure 13, the network latency is the same for the three operating systems. The amount of access control verifications for a network system call is small since our solution does not have to do expensive path searches.

Figure 14 shows the average latency when switching from a process to another one when 2, 4, 8 and 16 processes communicate together through a ring of Unix pipe. The overhead in analysis mode is about 4% on average. A context switching generates dozens of access control requests. Our solution is efficient since it computes the dozens of requests in a very short delay. With our solution in protection mode, the overhead is less than 6%.

Figure 15 shows the timing of creating and deleting files on a filesystem (an ext3 filesystem). It shows how many files can be created in one second with a size of 0, 1, 4 or 10Kb and

how many files can be deleted in one second with the same size. The overhead of our two solutions is less than 1%.

This performance analysis shows that our solution (in analysis or protection mode) is efficient for common usages. The fork overhead corresponds to an unusual stress of the system. To cope with such situations, we are looking at different precomputing and optimization methods while keeping a stable memory usage.

Our paper presents a novel approach to protect a complete operating system against attacks using Race Conditions. It provides a large state of art showing that this problem still is opened. Our solution prevents efficiently against both direct race conditions and indirect ones. A new protection property is defined to cope with these various race conditions. That property fits with the enforcement by an operating system against RCs.

An implementation is proposed for guaranteeing the proposed security property within a Linux system. Our implementation provides a dedicated MAC approach. For compatibility reasons, SELinux contexts are reused. However, the SELinux enforcement is not required and the solution supports ordinary DAC systems. It requires only security contexts associated to the different system resources.

Our approach computes the illegal activities dynamically. It allows to easily define security properties against RCs attacks.

Real examples show the efficiency of our approach for preventing real attacks using race conditions. A complete benchmark has been carried out to show the low overhead of our solution. Improvements are on the way to cope with unusual stress of the system.

Since our approach allows to formalize a wide range of security properties, future works will tackle other security properties dealing with integrity, confidentiality and availability.

REFERENCES

[1] J. Rouzaud Cornabas, P. Clemente, and C. Toinard, "An Information Flow Approach for Preventing Race Conditions: Dynamic Protection of the Linux OS," in *Fourth International Conference on Emerging Security Information, Systems and Technologies SECURWARE'10*, (Venise Italie), pp. 11–16, 07 2010.

[2] R. H. B. Netzer and B. P. Miller, "What are race conditions? some issues and formalizations," *LOPLAS*, vol. 1, no. 1, pp. 74–88, 1992.

[3] Y. Yu, T. Rodeheffer, and W. Chen, "Racetrack: efficient detection of data race conditions via adaptive tracking," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 221–234, 2005.

[4] T. Tahara, K. Gondow, and S. Ohsuga, "Dracula: Detector of data races in signals handlers," (Beijing, China), pp. 17–24, IEEE Computer Society, 2008.

[5] W. S. McPhee, "Operating system integrity in os/vs2," *IBM Syst. J.*, vol. 13, no. 3, pp. 230–252, 1974.

[6] M. Bishop, "Race conditions, files, and security flaws: or, the tortoise and the hare redux," *Technical Report*, vol. 95, sept. 1995.

[7] M. Bishop and M. Dilger, "Checking for race conditions in file accesses," *Computing Systems*, vol. 9, pp. 131–152, 1996.

[8] R. H. Netzer and B. P. Miller, "On the complexity of event ordering for shared-memory parallel program executions," in *In Proceedings of the 1990 International Conference on Parallel Processing*, pp. 93–97, 1990.

[9] H. Chen and D. Wagner, "Mops: an infrastructure for examining security properties of software," in *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 235–244, ACM, 2002.

[10] B. Chess, "Improving computer security using extended static checking," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pp. 160 – 173, 2002.

[11] B. Schwarz, H. Chen, D. Wagner, J. Lin, W. Tu, G. Morrison, and J. West, "Model checking an entire linux distribution for security violations," in *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, (Washington, DC, USA), pp. 13–22, IEEE Computer Society, 2005.

[12] C. Ko and T. Redmond, "Noninterference and intrusion detection," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, (Oakland, CA, USA), pp. 177–187, 2002.

[13] K.-s. Lhee and S. J. Chapin, "Detection of file-based race conditions," *International Journal of Information Security*, vol. 4, pp. 105–119, 2005. 10.1007/s10207-004-0068-2.

[14] A. Joshi, S. T. King, G. W. Dunlap, and P. M. Chen, "Detecting past and present intrusions through vulnerability-specific predicates," in *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 91–104, ACM, 2005.

[15] C. Cowan, S. Beattie, C. Wright, and G. Kroah-Hartman, "Raceguard: kernel protection from temporary file race vulnerabilities," in *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, (Berkeley, CA, USA), pp. 13–13, USENIX Association, 2001.

[16] E. Tsyrklevich and B. Yee, "Dynamic detection and prevention of race conditions in file accesses," in *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, (Berkeley, CA, USA), pp. 17–17, USENIX Association, 2003.

[17] P. Uppuluri, U. Joshi, and A. Ray, "Preventing race condition attacks on file-systems," in *Proceedings of the 2005 ACM symposium on Applied computing - SAC '05*, (New York, New York, USA), p. 346, ACM Press, 2005.

[18] F. Schmuck and J. Wylie, "Experience with transactions in quicksilver," in *SOSP '91: Proceedings of the thirteenth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 239–253, ACM, 1991.

[19] C. P. Wright, R. Spillane, G. Sivathanu, and E. Zadok, "Extending acid semantics to the file system," *Trans. Storage*, vol. 3, no. 2, p. 4, 2007.

[20] D. Mazieres and M. Kaashoek, "Secure applications need flexible operating systems," in *Operating Systems, 1997., The Sixth Workshop on Hot Topics in*, pp. 56 –61, 5-6 1997.

[21] D. Tsafrir, T. Hertz, D. Wagner, and D. Da Silva, "Portably solving file tocttou races with hardness amplification," in *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, (Berkeley, CA, USA), pp. 1–18, USENIX Association, 2008.

[22] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," in *15th National Computer Security Conference*, (Baltimore, MD, USA), pp. 554–563, Oct. 1992.

[23] ITSEC, "Information Technology Security Evaluation Criteria (ITSEC) v1.2," technical report, June 1991.

[24] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell, "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments," in *Proceedings of the 21st National Information Systems Security Conference*, (Arlington, Virginia, USA), pp. 303–314, Oct. 1998.

[25] J. Anderson, "Computer security threat monitoring and surveillance," tech. rep., James P. Anderson Company, Fort Washington, Pennsylvania, April 1980.

[26] D. E. Bell and L. J. La Padula, "Secure computer systems: Mathematical foundations and model," Technical Report M74-244, The MITRE Corporation, Bedford, MA, May 1973.

[27] T. Fraser and L. Badger, "Ensuring continuity during dynamic security policy reconfiguration in dte," pp. 15 –26, may. 1998.

[28] N. Li, Z. Mao, and H. Chen, "Usable mandatory integrity protection for operating systems," in *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, (Washington, DC, USA), pp. 164–178, IEEE Computer Society, 2007.

[29] T. Fraser, "LOMAC: Low Water-Mark integrity protection for COTS environments," *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pp. 230–245, 2000.

[30] Z. Mao, N. Li, H. Chen, and X. Jiang, "Trojan horse resistant discretionary access control," in *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies*, (New York, NY, USA), pp. 237–246, ACM, 2009.

[31] S. Vandebogart, P. Efstathopoulos, E. Kohler, M. Krohn, C. Frey, D. Ziegler, F. Kaashoek, R. Morris, and D. Mazières, "Labels and event

processes in the asbestos operating system," *ACM Trans. Comput. Syst.*, vol. 25, no. 4, p. 11, 2007.

[32] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières, "Making information flow explicit in histar," in *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2006.

[33] P. Efstathopoulos and E. Kohler, "Manageable fine-grained information flow," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 4, pp. 301–313, 2008.

[34] M. Krohn and E. Tromer, "Noninterference for a practical difc-based operating system," in *SP '09: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, (Washington, DC, USA), pp. 61–76, IEEE Computer Society, 2009.

[35] P. Clemente, J. Rouzaud-Cornabas, and C. Toinard, "From a generic framework for expressing integrity properties to a dynamic enforcement for operating systems," in *Transactions on Computational Science XI* (M. Gavrilova, C. Tan, and E. Moreno, eds.), vol. 6480 of *Lecture Notes in Computer Science*, pp. 131–161, Springer Berlin / Heidelberg, 2010.

[36] H. Liang and Y. Sun, "Enforcing mandatory integrity protection in operating system," in *ICCNMC '01: Proceedings of the 2001 International Conference on Computer Networks and Mobile Computing (ICCNMC'01)*, (Washington, DC, USA), p. 435, IEEE Computer Society, 2001.

[37] N. Li, Z. Mao, and H. Chen, "Usable mandatory integrity protection for operating systems," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, (Oakland, CA, USA), pp. 164–178, May 2007.

[38] J. Briffaut, J.-F. Lalande, and C. Toinard, "A proposal for securing a large-scale high-interaction honeypot," in *Workshop on Security and High Performance Computing Systems* (R. K. Guha and L. Spalazzi, eds.), (Cyprus), ECMS, 2008.

[39] M. Blanc, J. Briffaut, J.-F. Lalande, and C. Toinard, "Enforcement of security properties for dynamic mac policies," in *Third International Conference on Emerging Security Information, Systems and Technologies* (IARIA, ed.), (Athens/Vouliagmeni, Greece), pp. 114–120, IEEE Computer Society Press, June 2009.

[40] M. Blanc, P. Clemente, J. Rouzaud-Cornabas, and C. Toinard, "Classification of malicious distributed selinux activities," *Journal Of Computers*, vol. 4, pp. 423–432, may 2009.

[41] L. McVoy and C. Staelin, "lmbench: portable tools for performance analysis," in *ATEC '96: Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, (Berkeley, CA, USA), pp. 23–23, USENIX Association, 1996.

Experiences with the Automatic Discovery of Violations to the Normalized Systems Design Theorems

Kris Ven, Dieter Van Nuffel, Philip Huysmans, David Bellens, Herwig Mannaert
Department of Management Information Systems
University of Antwerp
Prinsstraat 13
2000 Antwerp, Belgium
{*kris.ven,dieter.vannuffel,philip.huysmans,david.bellens,herwig.mannaert*}*@ua.ac.be*

***Abstract*—Evolvability is an important concern for the design and development of information systems. The Normalized Systems theory has recently been proposed and aims to ensure the high evolvability of information systems. The Normalized Systems theory is based on the systems theoretic concept of stability and proposes four design theorems that act as constraints on the modular structure of software. In this paper, we explore the feasibility of building a tool that is able to automatically identify violations to these Normalized Systems design theorems in the source code of applications. Such a tool could help developers in identifying limitations to the evolvability of their applications. We describe how a prototype of such a tool was developed and report on the evaluation of this tool consisting of the analysis of the source code of four open source software applications. Our results demonstrate that it is feasible to automatically identify violations to the Normalized Systems design theorems. In addition, the results show that there is considerable variety in how well the different theorems are adhered to by various software applications. We also identified some issues and limitations with the current version of the tool and discuss how these issues can be addressed in a future version.**

***Keywords*-normalized systems; modularity; software architecture; quality**

I. Introduction

Contemporary organizations are operating in increasingly volatile environments and must be able to respond quickly to changes in their environment in order to gain a competitive advantage [2], [3]. Since organizations are becoming increasingly dependent on information technology (IT) to support their operations, the evolvability of the IT infrastructure will determine to a large extent how quickly organizations are able to adapt. It has indeed been shown that IT offers opportunities to organizations to increase their agility and flexibility [4]–[6]. Organizations therefore require increasing levels of evolvability of their information systems. Unfortunately, information systems struggle to provide the requested levels of evolvability, often due to poorly designed software architectures [7].

The Normalized Systems theory has recently been proposed by Mannaert and Verelst [8] and aims to address these evolvability issues. The Normalized Systems theory is concerned with how information systems can be developed based on the systems theoretic concept of stability [8]–[10]. It argues that the main obstacle to evolvability is the existence of so-called *combinatorial effects*. Combinatorial effects occur when the effort to apply a specific change increases as the system grows [8], [10]. The Normalized Systems theory eliminates these combinatorial effects by defining clear design theorems. These Normalized Systems design theorems act as constraints on the modular structure of software. Adhering to these theorems results in information systems that exhibit stability.

Organizations currently have a large number of in-house developed information systems in use. These information systems are likely to contain combinatorial effects that limit their evolvability. These combinatorial effects exist due to violations to the Normalized Systems design theorems. Organizations will therefore be looking towards ways to identify these combinatorial effects in their code base and to devise solutions to improve the evolvability of their information systems. Manually inspecting the source code may be a possibility, but is likely to be a very time-consuming task. The automatic identification of combinatorial effects therefore seems to be a very interesting alternative. In this paper, we explore the feasibility of building a tool to automatically identify violations to the Normalized Systems design theorems in the source code of applications. Although our main focus—similar to our previous research [9], [10]—is on information systems, this tool could be used to perform an evaluation of any type of software application. In this paper, we describe the development and evaluation of a prototype of such a tool. In our previous work, we already described the evaluation of this prototype using a single case [1]. Our current work further builds on this research by analyzing the source code of four open source software applications.

The rest of this paper is structured as follows. In Section II, we describe the previous work related to this study and focus on providing an introduction to the Normalized Systems theory. The methodology of our study is described in Section III. Section IV describes the development of our

tool. The evaluation of the tool is described in Section V. The results from the evaluation are discussed in Section VI. Finally, our conclusions are offered in Section VII.

II. Previous Work

In this section, we provide an overview of the literature related to our current study. We start with an introduction to the topic of software evolvability. Next, we focus on providing a background on the Normalized Systems theory.

A. Software Evolvability

It is a well-known problem in software engineering that the structure of software degrades and becomes more complex over time as changes are applied to it. One of the main challenges with respect to the evolvability of information systems is Lehman's Law of Increasing Complexity which states that: "*As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it.*" [11] This law implies that over time, the structure of software will become more complex—thereby requiring increasing effort to add new functionality to an existing system—unless preventive measures are taken [11]–[13]. This is clearly an important concern for information systems development. There is widespread belief that the software architecture determines the evolvability of software to a large extent [14]. As a result, a number of frameworks have appeared in literature that attempt to evaluate software architectures based on a number of quality attributes, including evolvability [15], [16]. Some of the most well-known evaluation methods include the *Architecture Trade-off Analysis Method (ATAM)* [17] and the *Software Architecture Analysis Method (SAAM)* [18]. Unfortunately, it has been noted that a theoretical foundation for studying software evolvability and evolution is largely missing [19]. As a first step towards such a theoretical foundation, Lehman derived a list of definitions, theorems and axioms with respect to software evolution based on a large empirical research project spanning multiple years [19].

The approaches mentioned above typically define a set of principles that should ensure the evolvability of software systems. Unfortunately, some of these principles are defined rather informally and leave considerable room for interpretation. As a result, these approaches struggle to consistently achieve evolvability in realistic software development environments. This is frequently a consequence of the fact that it is difficult to reach consensus among practitioners about how a principle should exactly be applied in practice. For example, when asked to evaluate alternative designs for a software system based on a principle such as loose coupling, practitioners frequently disagree on the best solution. Several tools exist that calculate a set of metrics of a software system in order to provide an idea of the evolvability of the software system. However, such assessments require a white-box approach. A statement that the software is more or less evolvable based on such assessments therefore have a limited meaning. The Normalized Systems theory is similar to these previous approaches in taking evolvability as the primary concern for developing software systems. The main difference with these previous approaches is that the Normalized Systems theory is based on the systems theoretic concept of stability and aims to provide clear principles on software evolvability. Such clear principles avoid the situation in which developers or software architects disagree on the exact interpretation of a principle. By stating that a software system is compliant with the Normalized Systems theory, a more black box assessment of evolvability is therefore possible, since this defines to which anticipated changes the software is stable.

B. Normalized Systems

In this section, we will provide a brief background on the Normalized Systems theory. However, the aim of this section is not to fully explain Normalized Systems, or to elaborate on the theorems and their rationale. Instead, we further build upon the previous work that is available in this area. For more details, we refer the reader to our previous work describing the Normalized Systems theory [8]–[10], [20]–[22].

The basic assumption of the Normalized Systems approach is that information systems should be able to evolve over time and should therefore be designed to accommodate change. This implies that the software architecture should not only satisfy the current requirements, but should also support future requirements. The Normalized Systems approach uses the systems theoretic concept of *stability* as the basis for developing information systems [8]–[10], [20]. In systems theory, stability refers to a system in which a bounded input function results in bounded output values, even as $t \rightarrow \infty$ (with t representing time). When applied to information systems, this means that applying a specific change to the information system should always require the same effort, irrespective of the size of the information system or the point in time at which the change is applied. The Normalized Systems approach further relies on the *assumption of unlimited systems evolution* [8]–[10]. This means that the system becomes ever larger in the sense that the number of modules become infinite or unbounded as $t \rightarrow \infty$. This may seem an overstated assumption, but actually, it is quite logical as even the introduction of a single module or dependency every twenty years corresponds to an infinite amount for an infinite time period.

Information systems exhibiting stability with respect to a defined set of changes are called *Normalized Systems* [8], [10]. In contrast, when changes do require increasing effort as the system grows, *combinatorial effects* are said to occur [8], [10]. In order to obtain stable information systems, these combinatorial effects should be eliminated. In order to

identify and avoid most of these combinatorial effects, a set of four *design theorems* was developed [8]–[10], [20]. It is important to note that it has been formally proven that these theorems contribute to achieving systems theoretic stability in software [9]. We will now briefly describe each of these theorems. More details are beyond the scope of this paper and can be found in the literature [8]–[10], [20].

The first theorem, *separation of concerns*, requires that every change driver or concern is separated from other concerns. This theorem allows for the isolation of the impact of each change driver. This principle was informally described by Parnas already in 1972 as what was later called *design for change* [23]. This theorem implies that each module can contain only one submodular task (which is defined as a change driver), but also that workflows should be separated from functional submodular tasks. For instance, consider a function F consisting of task A with a single version and a second task B with N versions; thus leading to N versions of function F. The introduction of a mandatory version upgrade of task A will not only require the creation of the additional task version of A, but also the insertion of this new version in the N existing versions of function F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect.

The second theorem, *data version transparency*, requires that data is communicated in version transparent ways between components. This requires that this data can be changed (e.g., additional data can be sent between components), without having an impact on the components and their interfaces. For instance, consider a data structure D that is passed to N versions of a function F. If an update of the data structure is not version transparent, it will also demand the adaptation of the code that accesses this data structure. Therefore, it will require new versions of the N existing processing functions F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. Data version transparency can, for example, be accomplished by appropriate and systematic use of web services instead of using binary transfer of parameters. This also implies that most external APIs cannot be used directly, since they use an enumeration of primitive data types in their interface.

The third theorem, *action version transparency*, requires that a component can be upgraded without impacting the calling components. Consider, for instance, a processing function P that is called by N other processing functions F. If a version upgrade of the processing function P is not version transparent, this will cause besides upgrading P, the adaptation of the code that calls P in the various functions F. Therefore, it will require new versions of the N existing processing functions F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. Action version transparency can be accomplished by appropriate and systematic use of, for example, polymorphism or a facade pattern.

The fourth theorem, *separation of states*, requires that actions or steps in a workflow are separated from each other in time by keeping state after every action or step. For instance, consider a processing function P that is called by N other processing functions F. Suppose the calling of the function P does not exhibit state keeping. The introduction of a new version of P, possibly with a new error state, would force the N functions F to handle this error, and would therefore lead to N distinct code changes. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. This theorem suggests an asynchronous and stateful way of calling other components. Synchronous calls—resulting in pipelines of objects calling other objects that are typical for object-oriented development—result in combinatorial effects.

It must be noted that each of these theorems is not completely new, and even relates to the heuristic knowledge of developers. However, formulating this knowledge as theorems that identify these combinatorial effects aids to build information systems that contain a minimal number of combinatorial effects. A remarkable aspect of these theorems is that a violation of each one of these theorems, by any developer at any moment during development or maintenance, results in a combinatorial effect. This suggests how difficult it is to realize software without combinatorial effects [8], [10].

The design theorems show that software constructs, such as functions and classes, by themselves offer no mechanisms to accommodate anticipated changes in a stable manner [8], [10]. The Normalized Systems approach therefore proposes to encapsulate software constructs in a set of five higher-level software elements [8], [10]. These elements are modular structures that adhere to these design theorems, in order to provide the required stability with respect to the anticipated changes [8], [10]. From the second and third theorem it can straightforwardly be deduced that the basic software constructs, i.e., data and actions, have to be encapsulated in their designated construct. As such, a *data element* represents an encapsulated data construct with its get- and set-methods to provide access to its information in a data version transparent way. So-called cross-cutting concerns, for instance access control and persistency, should be added to the element in separate constructs. The second element, *action element*, contains a core action representing a single functional task or change driver. Four different implementations of an action element can be distinguished: *standard* actions, *manual* actions, *bridge* actions and *external* actions [24]. In a standard action, the actual task is programmed in the action element and performed by the same information system. In a manual action, a human act is required to fulfill the task. The user then has to set the state of the life cycle data element through a user interface after the completion of the task. A process step can also require more complex

behavior. A single task in a workflow can be required to take care of other aspects which are not the concern of that particular flow [8], [10]. Therefore, a separate workflow will be created to handle these concerns. Bridge actions create these other data elements going through their designated flow. When an existing, external application is already in use to perform the required task, the action element would be implemented as an external action. These actions call other information systems and set their end state depending on the external systems' reported answer. Arguments and parameters of an action element need to be encapsulated as separate data elements, and cross-cutting concerns such as logging and remote access should be added as separate constructs. Based upon the first and fourth theorem, workflow has to be separated from other action elements [8], [10]. These action elements must be isolated by intermediate states, and information systems have to react to states. To enable these requirements, three additional elements are identified. A third element is a *workflow element* containing the sequence in which a number of action elements should be executed in order to fulfill a flow. A consequence of the stateful workflow elements is that state is required for every instance of use of an action element, and that the state therefore needs to be linked to or be part of the instance of the data element serving as argument. A *trigger element* is a fourth element that controls the states (both regular and error states) and checks whether an action element has to be triggered. Finally, the *connector element* ensures that external systems can interact with data elements without allowing an action element to be called in a stateless way [8], [10].

It is important to note that the basic underlying motivation of the Normalized Systems theory is to strive towards establishing an objective and scientific foundation to analyze the evolvability characteristics of information systems. The previous studies on this topic can be considered a first initial step towards turning software engineering into a classical engineering science that is based on laws and exhibits predictability [9].

III. Methodology

Our research has been conducted using the design science methodology. Our research goal is to develop a tool for the automatic identification of violations to the Normalized Systems design theorems in the source code of information systems. The design science methodology is appropriate in this case, since design science is primarily aimed at *solving* problems by developing and testing *artifacts*, rather than *explaining* them by developing and testing *theoretical hypotheses*. The design science research tradition focuses on tackling ill-structured problems in a systematic way [25]. Peffers et al. consider information systems to be an applied research discipline, meaning that theory from disciplines such as economics, computer science and social sciences are frequently used to solve problems between information technology and organizations [26]. In this research, we will use the Normalized Systems theory as the basis to develop a tool to identify potential issues with respect to the evolvability of software. Hence, we start from a solid theoretical foundation to develop a tool that has a large potential to be used in practice.

March and Smith have developed a classification scheme to position design science research efforts. This scheme identifies 4 different research outputs (i.e., construct, model, method and instantiation) and 4 different research activities (i.e., build, evaluate, theorize and justify) [27]. Our research is concerned with the *build* and *evaluate* phases of an *instantiation* artifact. The instantiation refers in this case to a tool to identify violations to the Normalized Systems design theorems. If such a tool could be developed, it would illustrate the feasibility of the automatic identification of violations. The importance of building instantiations has been emphasized by Newell and Simon, by writing: "*Each new program that is built is an experiment. It poses a question to nature, and its behavior offers clues to the answer*" [28].

Consistent with the design science methodology, an iterative approach will be followed in this research [26], [29], [30]. We started by first defining and motivating the problem based on the literature on Normalized Systems. Therefore, the research entry point is objective-centered, and is concerned with developing a tool to identify violations to the Normalized Systems design theorems [26]. Based on the Normalized Systems design theorems, we derived a number of violations that may occur in Java applications. In this first iteration, it is not our aim to create an exhaustive list of potential violations. Hence, we provide a lower bound for the existence of such violations in information systems. This constitutes a contribution towards the Normalized Systems approach, since this provides insight into which concrete violations to the Normalized Systems design theorems can be found in practice. Next, we investigate the feasibility of building a tool that can automatically identify manifestations of these violations in the source code of information systems.

Finally, we conduct a first evaluation of the tool. Evaluation is considered to be a key element in the design process [31]. To this end, we evaluate the tool by applying it to a set of Java applications, interpreting the resulting output and verifying the violations in the source code. The correctly identified violations confirm the utility of this tool. This first version of the tool is an important milestone, as it will give valuable feedback on the feasibility of the automatic inspection of the source code with respect to violations to the Normalized Systems design theorems. Furthermore, we will use the lessons learned from this first evaluation to improve the efficiency of our tool. In the following iterations, we will further develop and refine our tool. Future improvements include, for example, detecting a larger number of violations to the Normalized Systems design theorems. These future

versions will be evaluated using other applications as a test case. We seek to further evaluate the tool in the future by applying it to larger and more complex applications.

IV. Tool Development

In the *build* phase of our research, we iteratively developed a tool prototype for the automatic identification of violations to the Normalized Systems design theorems. Before developing the tool, we first needed to determine for which programming language we wanted to build the tool. Since each programming language has its own constructs and syntax, different violations are possible in different programming languages. Therefore, separate parsers should be developed for each programming language. We decided to focus on the Java programming language. This choice was motivated by a number of reasons. First, Java is a popular programming language that is used by a large number of applications, including traditional GUI applications and web-based applications. Second, Java EE is a popular framework to build enterprise applications, making it very relevant in an organizational context. Third, the reference implementation for Normalized Systems was also built in the Java EE environment [8], [10].

A graphical overview of the architecture of this tool is shown in Figure 1. It shows that the tool consists of two main components—`NSTVdoclet` and `NSTVdetect`—that have to be run in succession. The former component is responsible for parsing the source code, while the latter component is responsible for actually analyzing the source code for manifestations of violations to the Normalized Systems design theorems. This approach allowed us to decouple the parsing of the source code and the actual inspection of the source code, which is consistent with the separation of concerns theorem.

As shown in Figure 1, the first step of the analysis consists of processing the Java source code by the `NSTVdoclet` component. The `NSTVdoclet` component is written as a custom doclet to `javadoc`. The `javadoc` tool is part of the Java 2 SDK. By default, it generates documentation in HTML format of the API of a Java application. The `javadoc` tool is, however, easy to extend by creating custom doclets that provide output in an alternate format. The `NSTVdoclet` component filters the information obtained by `javadoc` since not all this information is required by `NSTVdetect`. Next, the output is written away in a temporary database. The information contained in this database is an internal representation of the source code that is to a large extent independent on a specific programming language (e.g., in terms of classes that have methods that take parameters of a certain type and that possibly throw an exception). This method has three main advantages. First, it allows us to reuse the source code parsing algorithm of `javadoc`. This avoids having to write a custom Java source code parser. In addition, the output provided by `javadoc` is clearly documented at the API-level, making it easy to parse and process this information. Second, most Java applications ship with an `ant` build file that allows the automatic compilation of Java source code. In most cases, this `ant` build file includes a `javadocs` target that generates the API-documentation for the application using `javadoc`. If such a build target is available, it is quite easy to specify in the build file that a custom `javadoc` doclet must be used. This ensures that parsing the source code does not require much effort, on the condition that the standard `javadoc` documentation can be generated. In general, it is sufficient to modify the `javadoc` task to indicate that a custom doclet that must be used by specifying the `doclet` and `docletpath` attributes (see also Figure 2).

In the second step of the analysis, the `NSTVdetect` component processes the information in this database and analyzes it to identify manifestations of violations to the Normalized Systems design theorems. Consistent with the separation of concerns theorem, the `NSTVdetect` component delegates the responsibility of the actual detection of these manifestations of violations to an extensible set of modules. Each module analyzes the internal representation of the source code for manifestations of a specific violation. Each module writes its output to a separate report file.

An $M-N$ relationship exists between these violations and the Normalized Systems design theorems: a single design theorem can be violated in several ways, while a single violation can refer to more than one theorem. Our tool currently supports the detection of manifestations of three violations that may occur in Java applications. The identification of these violations is based upon—and consistent with—previous work [8]–[10]. Although the current list of violations is not exhaustive, it includes common violations against the Normalized Systems design theorems and covers all four design theorems. This list can be further expanded in the future. As such, the current list represents a lower bound of the violations to the Normalized Systems design theorems that exist in Java applications. We will now discuss these violations and how they are detected by each module in more detail.

A. Import Multiple Concerns Violation

A first violation occurs when a class combines more than one concern by using the `import` statements in Java. Such a class violates the separation of concerns theorem and therefore results in combinatorial effects. Java classes can import and use functionality from external technology environments and packages by using the `import` instruction. This may introduce dependencies on these external technologies in an implicit way since each of these technologies can change independently in the future. Consider a specific concern that is combined with one or more other concerns in N different classes. If this concern changes in the future (e.g., when it is decided to use an alternative external technology),

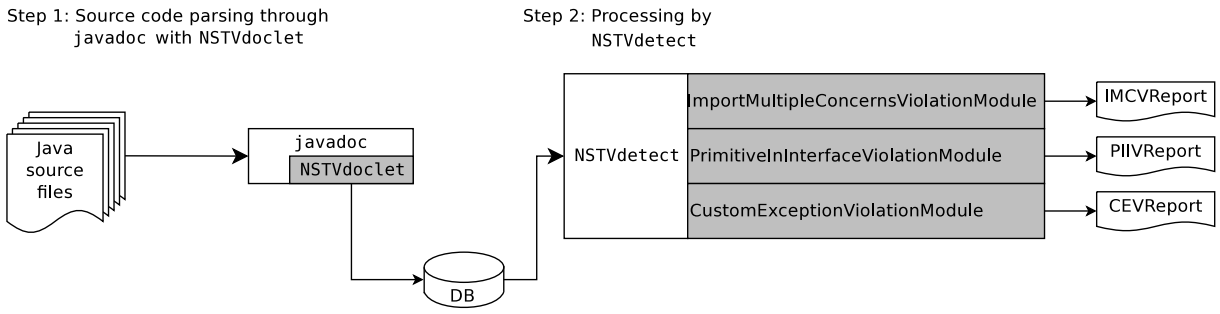


Figure 1. Tool Architecture

Original `javadoc` task for iText:

```
<javadoc
   destdir="${itext.docs}"
   author="true" maxmemory="128m"
   private="true">
```

Modified `javadoc` task for iText:

```
<javadoc
   destdir="${itext.docs}"
   author="true" maxmemory="128m"
   private="true"
   doclet="ua.mis.NSTVdoclet.main.NSTVdoclet"
   docletpath="nstvdoclet.jar">
```

Figure 2. Example of modification of ant build file

then this change has an impact on N different classes. Since N becomes unbounded over time, the impact of this change will increase over time as well, thereby resulting in a combinatorial effect. The separation of concerns design theorem requires that each change driver or concern is isolated from other concerns, so that each concern can evolve independently. This implies that each module should contain only one change driver [8]–[10]. As a result, a class should not combine two or more concerns.

The *Import Multiple Concerns Violation* module determines which concerns are used by each class based on the imported libraries (using the `import` statements in Java). We consider that the `import` statements in Java may provide a rough, but useful indication of which external technologies are used by a specific class, and therefore which concerns are addressed in the class. Before running the analysis, the researcher must define which concerns are present in the application, as well as which libraries fall under each concern. Depending on the application, one concern could, for example, be the use of the Java Swing packages for the graphical user interface, while a second concern could be the use of the Java JDBC packages to support database access. According to the separation of concerns theorem, both concerns should not be combined in a single class. This is consistent with the concept of multi-tier architectures. Another concern could be the use of another application, such as Cocoon to provide a web-based user interface. Based on this definition of concerns, it is determined how many different concerns are combined in each class. The researcher must define these concerns with care to ensure that the libraries correspond to the various concerns in the application as much as possible, in order to minimize the number of false positives identified by this module.

B. Primitive in Interface Violation

The second violation occurs when the interface of a method contains a primitive data type or a class of the type `java.lang.String`. Such a method violates both the data version transparency and action version transparency theorem and therefore results in combinatorial effects. Consider a method that is called by N other methods in the application and that contains one or more primitive data types or the `java.lang.String` class in its interface. If the functionality of this method is extended in the future, this extra functionality may require additional information to be sent to the method. However, the data that is sent to this method is not data version transparent. Since primitive data types or the `java.lang.String` class can only contain single values, it is not possible to send additional information without having an impact on this data structure. The method is not action version transparent either since the interface of the method will need to change to accept this additional information. It is therefore not possible to upgrade to a new version of the method without having an impact on the rest of the system. Hence, if additional data is needed by this method, this will have an impact on the N methods that call this method. In each of these N methods, the additional data needs to be initialized and the method call has to be adapted according to the modified interface of the method. Since N becomes unbounded over time, the impact of this change will increase over time as well, thereby resulting in a combinatorial effect. To resolve this issue, it is better to encapsulate the method parameters in a dedicated object with a default constructor. This constructor assigns neutral values to each of the parameters, which can be overwritten by calling the appropriate set methods. Future changes would

then have no effect on methods calling the method [8]–[10].

We further illustrate this principle with a practical example. Consider a method that allows the user to search for a specific string in a set of files and that takes a single parameter of the type `java.lang.String` that specifies the text to search for. Next, assume that the developers want to extend the search functionality in the future by allowing users to make use of regular expressions. In other words, the method should support searching based on normal text as well as regular expressions. In that case, the interface of the method should be extended with a `boolean` variable to indicate whether the string is a regular expression. This will, however, affect all other methods calling the search method. The data is not data version transparent since its structure does not allow to send additional data without any additional changes to the system. In addition, the method is not action version transparent since the interface of the method will change, and it is not possible to upgrade to a new version of the method without having an impact on the rest of the system. One could argue that overloading could be used in this case, by having one method with as interface `(java.lang.String)` and another method with as interface `(java.lang.String, boolean)`. However, this solution also has limitations with respect to evolvability since this approach is only feasible when the unique interface of the method can be guaranteed. If the same method would be extended with the functionality to make the search case-sensitive or not, another method with as interface `(java.lang.String, boolean)` would need to be added. However, this is impossible, since a method with this signature has already been defined. To resolve this issue, it is better to encapsulate the search parameters in a new class `SearchConfiguration` that can be extended with additional fields as new functionality is added to the search method. The default constructor of the `SearchConfiguration` object should assign a neutral value to newly added parameters (e.g., to indicate that a search string is not a regular expression). By calling the appropriate `set` method, the default settings can be overwritten. Future changes would then have no effect on methods calling the search method. This solution would be compliant with the data and action version transparency theorems.

The *Primitive in Interface Violation* module therefore inspects the interface of each non-private method and determines whether the interface includes one or more primitive data types or the `java.lang.String` class.

C. Custom Exception Violation

The third violation occurs when a method throws a custom exception (i.e., an exception that is not part of the default Java environment). The Java programming language provides the exception mechanism to handle errors that occur during the execution of a method. If an exception is thrown by a method, the calling method must process this error, either by catching and handling the error internally, or by throwing the exception further upward the stack. This constitutes a violation to the separation of states theorem and therefore results in combinatorial effects. Consider a method that is called by N different methods in the application. If the developer working on this method decides to introduce a new error state by having the method to throw a new exception, then this has an impact on the N methods that call this method, since they are forced by the Java environment to either catch or throw this exception further upward the stack. Hence, the error handling takes place in N different places. Since N becomes unbounded over time, the impact of this change will increase over time, thereby resulting in a combinatorial effect. Instead, the error state should be stored and error handling should be performed by a separate and dedicated module [8]–[10].

The *Custom Exception Violation* module therefore determines how many custom exceptions are thrown by all methods. We consider the use of standard Java exceptions (e.g., `java.lang.Exception` and `java.io.IOException`) to be acceptable, since they are related to the background technology being used. Even in this case, the use of these exceptions should be kept to a minimum. The use of custom exceptions should be avoided since such errors should be handled in a stateful way.

V. Case Study

We now discuss the *evaluation* phase of the design research process. In order to evaluate our tool, we analyzed the source code of a number of Java applications. These applications needed to satisfy three criteria. First, we focused on Java applications that are distributed under an open source license since this provides us with free access to the source code of these application. Second, the applications should represent a moderate development effort. Applications should not be too small to be disregarded as a toy example, but should also not be too large and too complex to complicate the evaluation of our tool. Third, we preferred to select applications that are quite popular and widely adopted to use applications that are used in real-life settings, rather than laboratory applications.

Based on these criteria, we selected four applications: (1) *Apache Lucene*, a fully-fledged text search engine; (2) *jEdit*, a programmers' text editor that supports a large number of programming languages; (3) *JabRef*, a bibliography reference manager that is used to edit BibTeX files; and (4) *iText*, a library that can be used by applications to facilitate the creation and manipulation of PDF documents. Details on the source code of these four open source software products can be found in Table I.

It must be noted that these four programs represent rather simple applications since they are written from scratch in Java, use a limited number of external libraries, and are

Table I
DETAILS OF SELECTED OPEN SOURCE SOFTWARE PRODUCTS

Name	Version	Classes	Methods	LOC
Apache Lucene	3.0.0	367	2,744	81,290
jEdit	4.3.1	477	1,980	163,015
JabRef	2.5	980	5,988	98,982
iText	5.0.5	583	5,663	140,883

not based on advanced frameworks. As a result, they do not exhibit the complexity of contemporary information systems, on which the Normalized Systems theory focuses. However, our aim in this paper is on exploring the feasibility of building a tool that is able to automatically identify violations to these Normalized Systems design theorems in the source code of applications. Our previous research has shown that even small applications are likely to contain several violations to the Normalized Systems theorems [1]. Therefore, applying our tool to analyze a complex information system is likely to result in a very large set of violations. Interpreting these results would complicate the evaluation of our tool. As a result, the four applications selected above are suitable for our purpose.

It is important to note that we do not want to make any claims with respect to the quality of the four applications in our case study. Instead, we want to perform an evaluation of our tool and its ability to automatically identify manifestations of violations to the Normalized Systems design theorems.

A. Import Multiple Concerns Violations

As mentioned in Section IV-A, we must first specify which concerns are present in a given application before running the analysis with `NSTVdetect`. To this end, we developed a shell script to extract a list of the unique package names that were imported by all classes of a given application from the temporary database created by `NSTVdoclet`. The task of the researcher is then to group these import statements in a set of concerns. The concerns that were identified for each of the four applications in our case study—as well as the packages that relate to each concern—are displayed in Table II. In this analysis, all packages belonging to the application itself were not considered to be a separate concern. In the case of jEdit, for example, all packages belonging to the `org.gjt.sp.jedit.*` package were considered part of the application itself, and not a separate concern.

A summary of the output from the *Import Multiple Concerns Violations* module is shown in Table III. According to the separation of concerns theorem, a class should not address more than one concern. The results from our analysis show a relatively low to moderate number of manifestations of this violation. The percentage of classes that are compliant with the separation of concerns theorem is (in decreasing order) 98.9% (363 out of 367 classes) for Lucene, 85.6% (499 out of 583 classes) for iText, 82.8% (395 out of 477 classes) for jEdit, and 63.9% (626 out of 980 classes) for JabRef. Lucene therefore performs very well, although it must be noted that only three concerns were identified for this application. JabRef has the highest number of violations with 36.1% of its classes, but also has the largest number of concerns. It therefore appears that there may be a relationship between the number of concerns that are identified for an application and the number of violations to the separation of concerns theorem. Overall, we can conclude that the separation of concerns theorem is rather to very well adhered to in all four applications.

A more detailed analysis showed that in those classes in which more than two concerns are combined, the Java IO concern is frequently combined with other concerns, such as Java Swing (e.g., JabRef and jEdit) or Java Net (JabRef and iText). Although most of these concerns are related to the default Java SDK API, it does create dependencies on different packages within the API. This data also suggests that file system functions (Java IO and Java Net) are combined with user interface functions (Java Swing). This may neglect the concept of multi-tiers and would therefore require attention in a further screening of the source code.

B. Primitive in Interface Violation

A summary of the output of the *Primitive in Interface* module is shown in Table IV. It can be seen that the percentage of methods that do not contain any manifestations of this violation and that do not contain any primitive data types or the `java.lang.String` class in their interface is (in decreasing order) 69.6% (4170 out of 5988) for JabRef, 61.7% (1693 out of 2744) for Lucene, 57.1% (1130 out of 1980) for jEdit, and 55.6% (3146 out of 5663) for iText. However, these percentages were calculated by including those methods that do not take any parameters and therefore require no input. If we exclude those methods from our analysis, the percentage of valid methods is 46.7% (1592 out of 3410) for JabRef, 36.0% (591 out of 1642) for Lucene, 31.8% (1172 out of 3689) for iText, and 30.6% (374 out of 1224) for jEdit. As could be expected, this lowers the proportion of valid methods considerably. Since these numbers are rather small, it can be concluded that the data and action version transparency theorems are not well adhered to in all four products.

C. Custom Exception Violation

A summary of the output of the *Custom Exception Violation* module is shown in Table V. As already mentioned in Section IV-C, we considered the use of standard Java exceptions to be acceptable, since they represent the background technology being used. This means that methods throwing `java.lang.Exception` and `java.io.IOException` exceptions were not considered a violation. The results show that the percentage of methods

Table II
LIST OF CONCERNS IDENTIFIED IN THE SELECTED APPLICATIONS

Application	Concern	Description
Lucene	Java IO	`java.io.*`
	Java Net	`java.net.*`
	Java Security	`java.security.*`
jEdit	Java Swing	`java.awt.*`, `javax.swing.*`
	Java Beans	`java.beans.*`
	Java IO	`java.io.*`, `java.nio.*`
	Java Net	`java.net.*`
	Java Security	`java.security.*`
	Java XML	`org.xml.sax.*`
	Microstar	`com.microstar.*`
JabRef	Java Swing	`java.awt.*`, `javax.swing.*`
	Java Beans	`java.beans.*`
	Java IO	`java.io.*`, `java.nio.*`
	Java Net	`java.net.*`
	Java SQL	`java.sql.*`
	Java XML	`javax.xml.*`, `org.w3c.dom.*`, `org.xml.sax.*`
	Java Plugin	`org.java.plugin.*`
	antlr	`antlr.*`, `org.antlr.*`
	glazedlists	`ca.odell.glazedlists.*`
	jgoodies	`com.jgoodies.*`
	ritopt	`gnu.dtools.ritopt.*`
	microba	`com.michaelbaranov.microba.*`
	jempbox	`org.jempbox.*`
	pdfbox	`org.pdfbox.*`
iText	Java Swing	`java.awt.*`, `javax.swing.*`
	Java IO	`java.io.*`, `java.nio.*`
	Java Net	`java.net.*`
	Java Security	`java.security.*`
	Bouncy Castle	`org.bouncycastle.*`
	dom4j	`org.dom4j.*`, `org.w3c.dom.*`, `org.xml.sax.*`

Table III
IMPORT MULTIPLE CONCERNS VIOLATIONS

	Number of		
Application	Concerns	Classes	Percentage
Lucene	0	110	30.0%
	1	253	68.9%
	2	4	1.1%
	Total:	*367*	*100.0%*
jEdit	0	174	36.5%
	1	221	46.3%
	2	59	12.4%
	3	12	2.5%
	4	11	2.3%
	Total:	*477*	*100.0%*
JabRef	0	306	31.2%
	1	320	32.7%
	2	228	23.3%
	3	86	8.8%
	4	34	3.5%
	5	6	0.6%
	Total:	*980*	*100.0%*
iText	0	273	46.8%
	1	226	38.8%
	2	74	12.7%
	3	7	1.2%
	4	3	0.5%
	Total:	*583*	*100.0%*

that do not throw any custom exceptions is (in decreasing order) 97.3% (5828 out of 5988) for JabRef, 95.4% (5401 out of 5663) for iText, 94.8% (2600 out of 2744) for Lucene, and 92.1% (1823 out of 1980) for jEdit. Interestingly, if we only consider those methods that throw at least one exception, it shows that the percentage of valid methods is 81.2% (621 out of 765) for Lucene, 69.5% (364 out of 524) for JabRef, 61.2% (414 out of 676) for iText, and 32.9% (77 out of 234) for jEdit. As could be expected, this lowers the percentage of valid methods. This decrease is most notable for jEdit which appears to make quite extensive use of custom exceptions. Overall, a rather mixed image therefore emerges with respect to the adherence to the separation of states theorem.

VI. DISCUSSION

Although the tool to automatically detect violations to the Normalized Systems design theorems is still a prototype, our evaluation has shown that there is much potential for such automated analysis. Compared to our original study [1], we evaluated our tool by analyzing the source code of four applications, instead of a single application. This provides more trust in the fact that the tool can be applied to a large set of software programs. Our evaluation has also shown that our `NSTVdoclet` tool can be easily integrated with `javadoc`, and offers sufficient information for identifying manifestations of violations to the Normalized Systems design theorems. Much information about the structure of software can already be derived from the API information obtained by `javadoc`. Focusing on the API-level has

Table IV
PRIMITIVE IN INTERFACE VIOLATIONS

Application	Violations[a] per Method	All methods		Methods with parameters	
		Method Count	Percentage	Method Count	Percentage
Lucene	0	1693	61.7%	591	36.0%
	1	719	26.2%	719	43.8%
	2	167	6.1%	167	10.2%
	3	103	3.8%	103	6.3%
	4	39	1.4%	39	2.4%
	5	12	0.4%	12	0.7%
	6	3	0.1%	3	0.2%
	7	7	0.3%	7	0.4%
	8	1	0.0%	1	0.1%
	Total:	*2744*	*100.0%*	*1642*	*100.0%*
jEdit	0	1130	57.1%	374	30.6%
	1	535	27.0%	535	43.7%
	2	187	9.4%	187	15.3%
	3	60	3.0%	60	4.9%
	4	50	2.5%	50	4.1%
	5	9	0.5%	9	0.7%
	6	7	0.4%	7	0.6%
	7	2	0.1%	2	0.2%
	Total:	*1980*	*100.0%*	*1224*	*100.0%*
JabRef	0	4170	69.6%	1592	46.7%
	1	1334	22.3%	1334	39.1%
	2	294	4.9%	294	8.6%
	3	132	2.2%	132	3.9%
	4	34	0.6%	34	1.0%
	5	18	0.3%	18	0.5%
	6	6	0.1%	6	0.2%
	Total:	*5988*	*100.0%*	*3410*	*100.0%*
iText	0	3146	55.6%	1172	31.8%
	1	1622	28.6%	1622	44.0%
	2	452	8.0%	452	12.3%
	3	157	2.8%	157	4.3%
	4	150	2.6%	150	4.1%
	5	52	0.9%	52	1.4%
	6	56	1.0%	56	1.5%
	7	14	0.2%	14	0.4%
	8	11	0.2%	11	0.3%
	9	3	0.1%	3	0.1%
	Total:	*5663*	*100.0%*	*3689*	*100.0%*

[a] Number of primitive and `java.lang.String` data types used in interface

several advantages. First, the Normalized Systems approach is concerned with the modular structure of software. Hence, inspecting the structure of classes and the interface of methods is consistent with this view. Second, the API-level represents a medium-level view on the modular structure of software. The package level can considered to be too high-level, as much information is abstracted away on this level. Conversely, considering the actual source code level may be too low-level.

By applying our tool to four different open source software applications, we were also able to determine how these applications differ in their adherence to the Normalized Systems design theorems. The results show that there is considerable variety in how well the different theorems are adhered to. Our data showed that the separation of concerns theorem—a well-accepted principle by practitioners—was rather well to very well adhered to by all four applications. The data and action version transparency theorems were, however, not well adhered to by the four applications since many methods made use of primitive data types in their interface. A rather mixed view was present with the separation of states theorem, where some applications made relatively little use of custom exceptions (e.g., Lucene), while other applications made rather intensive use of them (e.g., jEdit). Such violations are not fatal, but identify potential sources for combinatorial effects that limit the evolvability of the software. Given some limitations of the tool, we do not intend our results to be an assessment of the evolvability of the four applications. Instead, the aforementioned analysis was meant to be an evaluation of the tool.

The identification of violations by our tool was based on the Normalized Systems theory. The Normalized Systems theory states that in order to guarantee evolvability, all combinatorial effects must be eliminated from the source code of

Table V
CUSTOM EXCEPTION VIOLATIONS

Application	Violations[a] per Method	All methods: Method Count	All methods: Percentage	Methods with exceptions: Method Count	Methods with exceptions: Percentage
Lucene	0	2600	94.8%	621	81.2%
	1	134	4.9%	134	17.5%
	2	5	0.2%	5	0.7%
	3	5	0.2%	5	0.7%
	Total:	*2744*	*100.0%*	*765*	*100.0%*
jEdit	0	1823	92.1%	77	32.9%
	1	154	7.8%	154	65.8%
	2	3	0.2%	3	1.3%
	Total:	*1980*	*100.0%*	*234*	*100.0%*
JabRef	0	5828	97.3%	364	69.5%
	1	124	2.1%	124	23.7%
	2	16	0.3%	16	3.1%
	3	20	0.3%	20	3.8%
	Total:	*5988*	*100.0%*	*524*	*100.0%*
iText	0	5401	95.4%	414	61.2%
	1	248	4.4%	248	36.7%
	2	13	0.2%	13	1.9%
	3	1	0.0%	1	0.1%
	Total:	*5663*	*100.0%*	*676*	*100.0%*

[a] Number of custom exceptions thrown

applications. To realize this, the Normalized Systems theory posits four theorems that must be adhered to. The violations detected by our tool are based on these four theorems. As a result, the violations identified by our tool represent potential issues with respect to the evolvability of the application. The seriousness of these violations depends on the changes that will be applied to the software in the future. If a violation is present in a certain part of the application that will not change in the future, then the violation will have no impact on the evolvability of the software. For instance, when an interface of a method contains a primitive data type, but the interface will remain constant over time, then this violation does not impact the evolvability of the software. However, when the interface does change, it will require a modification in all parts of the software that call this method. In that case, a combinatorial effect is present that impacts the evolvability of the software. For a detailed discussion of the impact of a violation to the Normalized Systems theory, we refer the reader to earlier work [9], [10].

Given the considerable amount—and nature—of violations to the Normalized Systems theory identified in our case study, it seems likely that it would require much work to resolve these issues. Given the limited availability of resources, it is very unlikely that resolving all violations is feasible. Instead, developers could use the output of this tool to identify parts in the application that require specific attention. They could then attempt to normalize specific parts of the application, so that these parts in themselves become stable for the future. Within each part, however, combinatorial effects would still be allowed and not all violations would be addressed. These results therefore provide further empirical support for the statement that building information systems without combinatorial effects is extremely difficult, and that constructs of traditional programming languages offer no protection against violating the Normalized Systems theorems [10]. It seems unlikely that it is feasible to fully normalize an existing application given the limitations in time and budget available in practice. However, the Normalized Systems approach further provides a set of five software elements that are proven to be free of combinatorial effects and that can be used a building blocks for new applications [8], [10]. With those elements, it is possible to build applications that are largely free of combinatorial effects. As illustrated in previous work, a set of seven complex real-life applications have been developed in the process of refining the Normalized Systems theory [10]. In addition, independent applications that are compliant with the Normalized Systems theory are currently being built by several external organizations in Belgium and The Netherlands.

A. Lessons Learned

Based on the case study, several lessons can be learned about the feasibility of automatically detecting violations to the Normalized Systems design theorems. These lessons may be useful for future versions of the tool.

First, by separating the parsing of the source code by `NSTVdoclet` and the analysis by the `NSTVdetect` tool in two components, it may be relatively easy to add support for additional programming languages in the future. Evidently, other programming languages would require a different implementation of the `NSTVdoclet` component to parse the source code. The output of this component is currently largely language-independent: information about the source code is stored in terms of classes, methods,

and parameters. These concepts apply to all object-oriented programming languages. The only concept that is not supported by all object-oriented programming languages is the Java exception. Hence, the database format should provide the possibility to support language-specific extensions. The `NSTVdetect` tool can be extended with additional modules that provide support for other programming languages. Some existing modules, such as the *Primitive in Interface Violation* module also applies for programming languages such as C++ or C#. Other modules may only apply to a specific set of programming languages. In that case, each module must specify to which programming language(s) it applies, so that the `NSTVdetect` component can decide whether the module should be invoked when performing a specific analysis.

Second, the output obtained by the tool provides us with feedback on the current implementation of the modules that test for manifestations of violations to the Normalized Systems design theorems. Concerning the *Import Multiple Concerns Violation* module, we have observed that applications frequently combined the Java Swing concern with the Java IO and/or Java Net concerns. This may suggest that input/output instructions are combined with the user interface. However, a closer inspection of the source code showed that this was a false positive. For example, in order to provide icons in toolbars in the user interface, Java Swing provides the `ImageIcon` class. It is common to initialize a new object of this class by using the constructor taking a `java.net.URL` object as parameter. This, for example, allows the icon file to be part of the jar file that contains the application. This explicitly couples the Java Swing and Java Net concerns. To avoid this, one of the other constructors provided by the `ImageIcon` class should be used instead, for example, by sending the raw image data as a byte array. Similar violations may occur when objects of the class `java.io.File` are passed as a parameter to a method. In future research, we may try to find ways to automatically identify and report such instances to avoid manual inspection.

The output also allows us to consider whether the use of import statements is a good basis to identify concerns in an application. On the one hand, we believe this is indeed a quick and convenient way to identify the primary concerns that are present in an application. The Normalized Systems theory states that the use of an external technology always implies a different concern that should be separated [10]. Since external technologies must be made available in Java application through the `import` statement, this provides a good way to identify concerns. On the other hand, this method may neglect the internal structure of the application to some extent (when different concerns are present within the application itself), and it also leaves some room for deciding on which concerns are present within the application. This may result in false positives or false negatives in the detection process. It may therefore be worthwhile to consider other methods for identifying concerns within an application.

Concerning the *Primitive in Interface Violation* module, we identified a large number of methods that include one or more primitives in their interface. It appears that a manual inspection is required to investigate whether it is worthwhile to resolve these issues by making the data and methods version transparent. In case the interface can be expected to remain stable, it may not be worth the effort to encapsulate the parameters in their own dedicated object. Nevertheless, developers should remain aware that not addressing this issue can mean that additional methods may call this method in the future, thereby leading to an increase in combinatorial effects. It is theoretically possible that some of the methods that include primitive data types have a corresponding wrapper method that is version transparent and that should be called instead of the underlying method. In other words, it is possible that the source code includes both a version non-transparent method and an additional version transparent method. Our tool is currently not able to detect such instances. However, given the large number of manifestations of this violation found in all four applications, it can be expected that this is not done very often. Moreover, any non-version transparent public method can be called by additional methods in the future, thereby leading to an increase in combinatorial effects.

With respect to the *Custom Exception Violation* module, we can easily identify those methods that throw a custom exception. Further investigation of the source code of the four applications with respect to this issue showed that in several cases the calling method did not do anything when a method throws an exception, except for logging the error. However, since this external method throws an exception, the class that contains the calling method must import the package containing the exception. For example, if an external method throws the `java.io.IOException`, it must be imported by the class containing the calling method. Interestingly, this may further contribute to the *Import Multiple Concerns Violation*, if that class also imports other concerns. We have indeed noticed that several user interface classes import the `java.io.IOException` class since they must be able to react to exceptions thrown by methods of other classes. Although we considered the use of the default Java exceptions to be acceptable, the same reasoning applies to custom exceptions or exceptions from external technologies. This further emphasizes the $M-N$ relationship between the design theorems and violations (see Section IV).

B. Limitations

Since this tool is still a prototype, we acknowledge several limitations with respect to our findings.

A first important limitation is that the tool currently provides a lower bound of manifestations of violations to the Normalized Systems design theorems. The results therefore

provide a first-cut and rough assessment of violations to the Normalized Systems design theorems at the API-level. This assessment can increase sensibilisation about—and give a first impression of—the code quality of an application with respect to evolvability. Currently, we have distinguished between three violations against the Normalized Systems design theorems in Java applications. Each module of `NSTVdetect` checks for manifestations of a specific violation. All modules share a common interface and receive an internal representation of the source code as input. The tool can be extended in the future with new modules that check for additional violations to the Normalized Systems design theorems in order to detect a larger number of violations to the Normalized Systems design theorems.

A second limitation is that there is a risk for the existence of false positives reported by the tool. Although our experiences suggest that it is feasible to automatically detect several violations to the Normalized Systems design theorems, it still remains necessary to perform a manual inspection of the source code afterwards. This manual inspection provides more insight into the seriousness of the issues identified in the analysis. This is especially the case for the import multiple concerns violation since the choice of how libraries are grouped into concerns is to some extent arbitrary. For example, the Java API can considered to be rather stable. Hence, importing packages from several parts of the Java API may not constitute a very large risk with respect to combinatorial effects. A manual inspection is therefore required to investigate whether some issues reported by the tool are false positives. Notwithstanding the fact that some manual work is still required, our tool significantly reduces the effort compared to manually inspecting the code base for violations. In addition, the tool can quickly highlight potentially problematic parts in the source code that should be analyzed manually with priority.

A third limitation is that the definition of which concerns are present in the application is still to some extent arbitrary, since the researcher must first define which concerns are present in the application based on the import statements that are used in the application. This approach first rises the question as to whether the use of a library is an appropriate indication of a concern. A concern is considered to be a separate change driver or, in other words, a technology that can change independently from the background technology. The use of an external library represents the use of an external technology and therefore always represents a different change driver or concern. Although there may be other concerns that do not correspond to the use of a library, our tool is therefore able to provide at least a lower bound of the concerns present in a given application. A second question with respect to this approach is whether all concerns are correctly identified. In this case study, we probably did not select the best concerns to evaluate the evolvability of the four applications. For example, since Java is the background technology, it would make sense not to identify Java as a separate concern. In the case study, we only considered the `java.lang.*` and `java.util.*` packages to be part of the background technology in order to identify a larger number of concerns that would allow us to better evaluate our tool. In case the tool would be used to assess the evolvability of a software product, it would make sense to only identify external technologies as a separate concern. In addition, we have assumed that all packages related to the application itself are part of the background technology. For example, in the case of JabRef, all packages below `net.sf.jabref.*` were ignored when creating the list of concerns. Depending on the application being assessed, it may be interesting to further distinguish between multiple concerns in the application itself, to allow different parts of the application to evolve independently.

A final issue concerns the question of how far software developers should go in adhering to the Normalized Systems theorems. We are aware that some of these theorems, their implications, and the violations identified by this tool may seem rather radical at first sight. As mentioned in Section II-B, the Normalized Systems theory uses the assumption of unlimited systems evolution [8]–[10]. This means that the code base of the application will continue to increase over time. The aim of the Normalized Systems design theorems is to eliminate all combinatorial effects. Since combinatorial effects are very easily introduced into the source code, very strict and clear design theorems are required to eliminate them [8]–[10]. In this respect, the Normalized Systems theory encourages software developers to strive towards applying these theorems to the greatest extent possible [9], [10]. In practice, some trade-off is likely to take place to judge whether the additional effort of containing combinatorial effects is warranted by the likelihood that a future change would manifest itself. For example, it is possible that it is reasonable to use primitive data types in the interface of some methods that are not exposed to outside applications and that can be expected not to require additional data in the future (i.e., to have a stable interface). Similarly, the use of custom exceptions may be appropriate in cases when the method is unlikely to be called by other parts of the application. However, such decisions should be carefully considered. Developers should also be aware that not adhering to the Normalized Systems design theorems may have a negative impact on the future evolvability of the software. In order to fully comply with the Normalized Systems design theorems, at least all the violations identified in the source code should be addressed.

Notwithstanding these limitations, we feel that this tool can be very useful to investigate the quality of an application at the API-level with respect to evolvability using the Normalized Systems design theorems.

VII. Conclusion and Future Work

In this paper, we have explored the feasibility to automatically identify violations to the Normalized Systems design theorems. To this end, we developed a prototype of a tool that is able to detect violations in Java applications at the API-level. A case study was performed to evaluate the tool by analyzing the source code of four open source Java applications. A first contribution of our study is that we have shown that it is indeed possible to detect violations to the Normalized Systems design theorems in an automated manner. A manual inspection should, however, still provide more insight into the seriousness of the issues identified in the analysis and to identify any possible false positives. A second contribution is that we have performed an analysis of the source code of four software applications. The results showed that the tool can be applied to a range of software applications and that there is considerable variety in how well the different theorems are adhered to by various software applications. Our results show that all four applications adhere rather well to the separation of concerns theorem. However, we identified a larger number of violations to the data and action version transparency theorems. With respect to the separation of states theorem, a rather mixed picture emerged. It can therefore be expected that the output of this tool will be useful to assess the current design of applications and to identify potential limitations to their evolvability. However, given the current limitations of our tool, we do not want to make any claims with respect to the quality of the four applications in our case study. Instead, this case study provided valuable feedback that will be used to further improve our tool. A third contribution is that we have distinguished between three violations to the Normalized Systems design theorems that may occur in Java applications. Although this list is not exhaustive, our results already show that quite a large number of manifestations of these violations can be found in the source code of Java applications. One of the limitations of our tool is that it currently provides a lower bound for the existence of violations, and is not able to detect all possible violations to the Normalized Systems design theorems.

In future research, we intend to further develop this tool to improve its ability to automatically detect violations to the Normalized Systems design theorems. Our current research efforts focus on two different topics.

First, we are are extending our list of violations to identify a larger proportion of violations to the Normalized Systems design theorems. Instead of the top-down approach used in this paper (i.e., by starting from the Normalized Systems design theorems and deriving which violations could be found in the source code of applications), we are currently using a bottom-up approach. This approach consists of reviewing the source code of a number of open source software applications with the aim of identifying fragments of the source code that represent manifestations of violations to one or more of the Normalized Systems design theorems. Based on these observations, different violations will be identified. This approach therefore has an empirical foundation, instead of the theoretical approach followed in this paper. For each new violation, an additional module can be added to the `NSTVdetect` tool.

Second, we are further developing the architecture underlying our tool. One focus area is the intermediate format of the database that is used to save the structure of the source code. We are currently investigating if a suitable ontology exists that can be used to store this information in a language-independent format, while still allowing for language-specific features to be added. This would facilitate the support of different programming languages by the tool. Another area is to evaluate whether the information obtained by `javadoc` is sufficient to identify the new violations that are being discovered in our work on the previous topic.

As the tool further evolves, we will also apply our tool to evaluate more complex information systems that incorporate various frameworks (e.g., software component models, user-interface frameworks, and communication frameworks). This would allow us to assess how many violations to the Normalized Systems theory are detected in typical information systems, compared to the rather simple projects included in our case study.

This work should greatly facilitate the automatic identification of combinatorial effects in large-scale, real-life information systems. This tool could be used by organizations to analyze their information systems for the manifestation of violations to the Normalized Systems design theorems. Based on the output of this tool, organizations can take measures to improve the evolvability of their information systems.

References

[1] K. Ven, D. Van Nuffel, D. Bellens, and P. Huysmans, "The automatic discovery of violations to the normalized systems design theorems: A feasibility study," in *Proceedings of the 5th International Conference on Software Engineering Advances (ICSEA 2010),* August 22–27, 2010, Nice, France, J. G. Hall, H. K. Kaindl, L. Lavazza, G. Buchgeher, and O. T. Takaki, Eds. Los Alamitos, CA: IEEE Computer Society, 2010, pp. 38–43.

[2] D. J. Teece, G. Pisano, and A. Shuen, "Dynamic capabilities and strategic management," *Strategic Management Journal*, vol. 18, no. 7, pp. 509–533, 1997.

[3] K. M. Eisenhardt and J. A. Martin, "Dynamic capabilities: What are they?" *Strategic Management Journal*, vol. 21, no. 10/11, pp. 1105–1121, 2000.

[4] S. Neumann and L. Fink, "Gaining agility through IT personnel capabilities: The mediating role of IT infrastructure capabilities," *Journal of the Association for Information Systems*, vol. 8, no. 8, pp. 440–462, 2007.

[5] V. Sambamurthy, A. Bharadwaj, and V. Grover, "Shaping agility through digital options: Reconceptualizing the role of information technology in contemporary firms," *MIS Quarterly*, vol. 27, no. 2, pp. 237–263, jun 2003.

[6] L. Fink and S. Neumann, "Exploring the perceived business value of the flexibility enabled by information technology infrastructure," *Information & Management*, vol. 46, no. 2, pp. 90–99, mar 2009.

[7] J. L. Zhao, M. Tanniru, and L.-J. Zhang, "Services computing as the foundation of enterprise agility: Overview of recent advances and introduction to the special issue," *Information Systems Frontiers*, vol. 9, no. 1, pp. 1–8, 2007.

[8] H. Mannaert and J. Verelst, *Normalized Systems—Re-creating Information Technology Based on Laws for Software Evolvability*. Kermt, Belgium: Koppa, 2009.

[9] H. Mannaert, J. Verelst, and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability," *Science of Computer Programming*, in press. DOI: 10.1016/j.scico.2010.11.009.

[10] ——, "Towards evolvable software architectures based on systems theoretic stability," *Software: Practice and Experience*, in press. DOI: 10.1002/spe.1051.

[11] M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, Sept. 1980.

[12] L. Belady and M. M. Lehman, "A model of large program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 225–252, 1976.

[13] M. Lehman and J. Ramil, "Rules and tools for software evolution planning and management," *Annals of Software Engineering*, vol. 11, pp. 15–44, 2001.

[14] D. Garlan and D. E. Perry, "Introduction to the special issue on software architecture," *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 269–274, 1995.

[15] R. Bahsoon and W. Emmerich, "Evaluating software architectures: Development, stability, and evolution," in *Proceedings of ACS/IEEE International Conference on Computer Systems and Applications*, 2003.

[16] M. Ali Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation," in *Proceedings Australian Software Engineering Conference (ASWEC)*, 2004, pp. 309–318.

[17] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Proceedings of the Fourth IEEE International Conference on Engineering Complex Computer Systems (ICECCS'98)*, 1998.

[18] R. Kazman, G. Abowd, L. Bass, and M. Webb, "SAAM: A method for analyzing the properties of software architectures," in *Proceedings of the 16th International Conference on Software Engineering*, 1994, pp. 81–90.

[19] M. M. Lehman, "Approach to a theory of software process and software evolution: Position paper," in *FEAST 2000 Workshop, Imperial College, London, July 10–12, 2000*, 2000.

[20] H. Mannaert, J. Verelst, and K. Ven, "Exploring the concept of systems theoretic stability as a starting point for a unified theory on software engineering," in *Proceedings of the Third International Conference on Software Engineering Advances (ICSEA 2008),* Sliema, Malta, October 26-31, 2008, H. Mannaert, T. Ohta, C. Dini, and R. Pellerin, Eds. Los Alamitos, CA: IEEE CS Press, 2008, pp. 360–366.

[21] ——, "Exploring concepts for deterministic software engineering: Service interfaces, pattern expansion, and stability," in *International Conference on Software Engineering Advances*, Cap Esterel, France, Aug. 25–31, 2007.

[22] ——, "Towards rules and laws for software factories and evolvability: A case-driven approach," in *International Conference on Software Engineering Advances*, Tahiti, French Polynesia, Nov. 1–2, 2006.

[23] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[24] D. Van Nuffel, H. Mannaert, C. De Backer, and J. Verelst, "Towards a deterministic business process modelling method based on normalized systems theory," *International Journal on Advances in Software*, vol. 3, no. 1/2, pp. 54–69, 2010.

[25] J. Holmström, M. Ketokivi, and A.-P. Hameri, "Bridging practice and theory: A design science approach," *Decision Sciences*, vol. 40, no. 1, pp. 65–87, February 2009.

[26] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007.

[27] S. T. March and G. F. Smith, "Design and natural science research on information technology," *Decision Support Systems*, vol. 15, no. 4, pp. 251–266, 1995.

[28] A. Newell and H. Simon, *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

[29] H. A. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge, Massachusetts: MIT Press, 1996.

[30] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[31] A. Hevner and S. Chatterjee, *Design Research in Information Systems: Theory and Practice*, ser. Integrated Series in Information Systems. Springer, 2010, vol. 22.

Metrics for Evaluating Service Designs Based on SoaML

Michael Gebhart, Sebastian Abeck
Research Group Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{gebhart | abeck} @kit.edu

***Abstract*—In the context of service-oriented architectures, quality attributes, such as loose coupling and autonomy, have been identified that services should fulfill. In order to influence services with regard to these quality attributes, an evaluation is necessary at an early development stage, i.e. during design time. Existing work mostly focuses on a textual description of desired quality attributes, formalizes metrics that require more information than available during design time, or bases on a theoretical model that hampers the practical applicability. In this article, quality indicators for a unique categorization, loose coupling, discoverability and autonomy are identified. For each quality indicator formalized metrics are provided which enable their measurement and application on service candidates and service designs based on the Service oriented architecture Modeling Language as standardized language for modeling service-oriented architectures. To illustrate the metrics and to verify their validity, service candidates and service designs of a campus guide system as developed at the Karlsruhe Institute of Technology are evaluated.**

Keywords-service design; soaml; evaluation; metric; quality attribute; quality indicator

I. INTRODUCTION

With the shift to service-oriented architectures, goals concerning the information technology (IT) of companies, such as an increased flexibility, are expected to be attained. In order to support this attainment, quality attributes have been identified, services within a service-oriented architecture as building-blocks [28] should fulfill. Widespread quality attributes are loose coupling, unique categorization, discoverability, and autonomy [3, 6, 8, 11, 15, 28].

Since the design of services heavily influences the services and thus their quality attributes, it is necessary to perform the design phase with care. The quality attributes of services have to be determined during design time and based on this evaluation the service designs have to be revised if necessary. For this purpose, the quality attributes have to be described in a way that the IT architect can comprehensibly apply them on service designs. This requires a formalization of the quality attributes and additionally an involvement of a standardized language for service designs that can be used in real-world projects. The Service oriented architecture Modeling Language (SoaML) [20] has evolved as increasingly accepted and employed language to model service-oriented architectures, respectively their elements. Thus, we claim quality attributes being measureable on service designs based on SoaML or comparable languages. This enables their application without additional interpretation or transformation effort.

In existing work either a textual description of quality attributes or the formalization of metrics that measure certain aspects is focused. Textual descriptions are introduced by Erl [3], Reussner et al. [6], Josuttis [7], Engels et al. [8], Cohen [10], and Maier et al. [11, 12, 13]. They introduce a comprehensive set of quality attributes that should be considered when developing services. However, due to the textual descriptions, an application of these quality attributes is hampered and requires a prior interpretation. Each of the described quality attributes covers a lot of different aspects. Some of these aspects are already relevant during design time and others are only of interest in subsequent phases, such as the implementation phase. The IT architect has to analyze the quality attributes and identify relevant and measurable quality indicators first. Then he has to interpret them so that they can be applied on a modeled service design. Other work, as introduced by Perepletchikov et al. [14, 15, 16], Humm et al. [9], Rud et al. [17], Hirzilla et al. [18], and Choi et al. [19] focus on the formalization of metrics. Some of these formalizations base on theoretical models. This hampers the application on service designs that have been modeled using a standardized and widespread modeling language, such as SoaML, for a prior mapping of the different concepts is required. Additionally, the metrics are mostly abstractly and conceptually described which requires a prior interpretation again. For example concepts, such as “number of clients”, are used and the IT architect has to interpret if services or operations are meant and if duplicates should be counted or not. This interpretation may result in mistakes and consequently wrong evaluations. Finally, the metrics are often not related to widespread quality attributes that support the attainment of goals concerning the IT. Metrics that are not related to these widespread quality attributes are not motivated.

In this article, we derive quality indicators for the quality attributes unique categorization, loose coupling, discoverability, and autonomy. These quality attributes were chosen for they contain a representative set of aspects that can be measured during design time as shown in [1]. The quality indicators are derived directly from common and widespread descriptions of quality attributes in order to preserve the relation of the quality indicators to quality

attributes and thus motivate their measurement. Additionally, each quality indicator is formalized in form of metrics using the notation as introduced by Perepletchikov [13] in order to enable a comprehensible measurement. The formalization is adjusted to the elements of service candidates or service designs and their elements in SoaML. This allows a direct application of the quality indicators and their formalizations without any interpretation effort and thus reduces potential interpretation mistakes.

To illustrate the metrics and their validity, they are exemplarily applied on service candidates and service designs for a service-oriented system that guides students across the campus of the Karlsruhe Institute of Technology (KIT) called KITCampusGuide. This system has its origin in the NEST system, a service-oriented surveillance system developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [37, 38, 39]. As requirement, the services of the KITCampusGuide are expected to fulfill the quality attributes unique categorization, loose coupling, discoverability, and autonomy. The service candidates as preliminary services and the final service designs are modeled using SoaML.

The article is organized as follows: In Section 2, the fundamentals in the context of modeling service candidates and service designs using SoaML and the evaluation of services are presented. Section 3 introduces the derived quality indicators and their formalizations. The quality indicators are exemplarily applied on service candidates and service designs of the service-oriented KITCampusGuide. Section 4 concludes the article and introduces suggestions for future research.

II. FUNDAMENTALS

This article focuses on the evaluation of service designs. This requires a prior understanding of the concept of a service design that is introduced in the following section. Afterwards existing work in the context of evaluating services is analyzed with regard to their application on service designs. Finally, the existing work is discussed which constitutes the motivation for this article.

A. Modeling Service Designs

In order to analyze existing work with regard to their applicability for evaluating service designs, in the following, the service design artifact and its elements in SoaML are introduced. According to Erl [4, 5] and the Rational Unified Process (RUP) [24] for Service-Oriented Modeling and Architecture (SOMA) [21, 22, 23], the design process consists of two phases, the identification phase and the specification phase.

1) Identification Phase: The first phase, the identification phase, focuses on the determination of so-called service candidates [25]. They represent preliminary services as an abstract group of capabilities the service provides. A service candidate includes operation candidates as capabilities of the service candidate and preliminary operations. Additionally, the dependencies between service candidates are modeled. They describe that an operation candidate within a certain service candidate requires an operation candidate of another service candidate. In SoaML, service candidates are modeled using the Capability element in form of a stereotyped UML class. The operation candidates are added as UML operations. Dependencies can be modeled with usage dependencies in UML. The application of the Capability elements for service candidates is also confirmed by IBM. Within RUP SOMA, IBM uses its proprietary UML profile for modeling service candidates, the UML 2.0 profile for software services [33]. However, in newest work [34] they demonstrate the modeling of service candidates on the basis of SoaML which correlates with our understanding. The following Figure illustrates a set of service candidates and their dependencies.

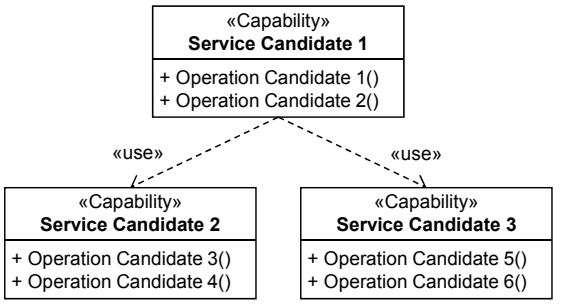


Figure 1. Modeling service candidates

2) Specification Phase: During the second phase, the specification phase, for each of the service candidates that constitute the basis for the implementation phase detailed service designs are created [26]. According to Erl [4, 27] and IBM [21, 22], a service design consists of a specified service interface that describes the service and a specification of the service component that fulfills the functionality. Latter includes the provided and required services and the internal logic in form of an orchestration of required services.

For modeling a service interface in SoaML the ServiceInterface element is provided that can be modeled using a stereotyped UML class. It includes a description about participating roles as UML Parts, realizes a UML interface that contains the provided operations, and uses another UML interface that includes operations a service consumer has to provide in order to receive callbacks. Additionally, the interaction protocol can be specified that describes the order of operations for gaining a valid result. The interaction protocol can be described using a UML Activity that is added as OwnedBehavior. Within this Activity, for each participating role a Partition is added containing the operations of the according UML interfaces. Figure 2 shows a service interface in SoaML. According to this service interface, the described service provides one operation called operation1. The service consumer has to provide one operation callbackOperation1 in order to receive a callback. The interacting roles are referred to as provider and consumer. The interaction protocol determines that for a valid result first the operation1 has to be called that is provided by the service. Afterwards, the callbackOperation1 of the service consumer is called by the service provider.

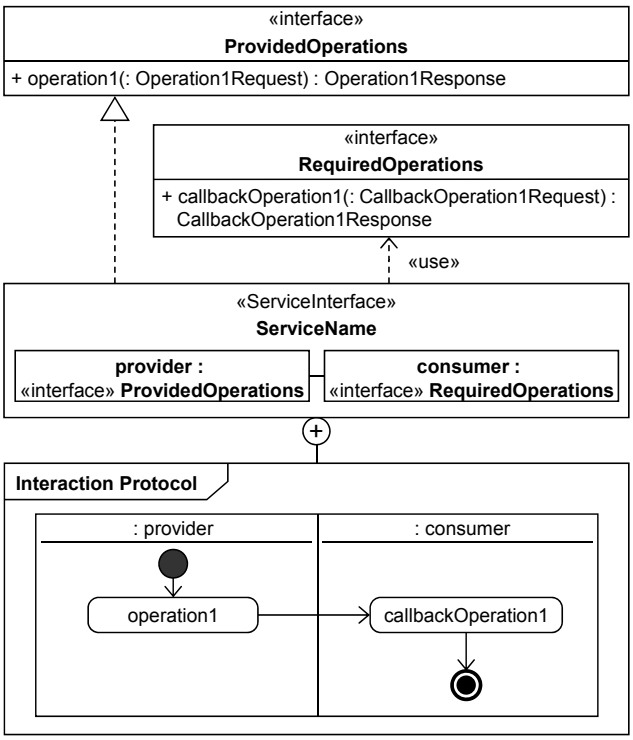


Figure 2. Modeling a service interface

Since each of the operations within the interfaces include messages being exchanged, a specification of these messages is necessary. For this purpose, the MessageType element is provided by SoaML that extends the UML data Type. It represents a document-centric message and can contain other data types. The following figure shows an excerpt of the message types required within the service interface above.

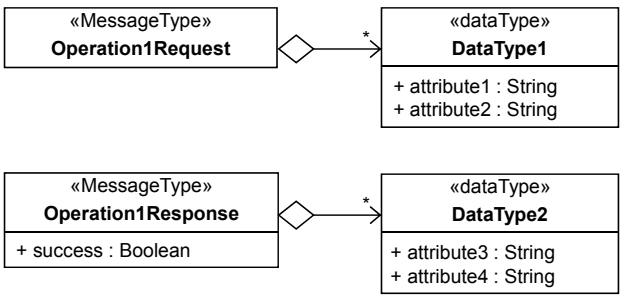


Figure 3. Modeling message types

The service component as part of the service design describes the component that fulfills the functionality of a service. For this purpose in SoaML the Participant element exists. It describes an organization, system, or software component. The service component is modeled using a stereotyped UML component. For each provided service, a ServicePoint is added to the service component that is typed by the describing ServiceInterface. Similarly, for each required service, a RequestPoint is specified that is also typed by the according ServiceInterface. The following figure illustrates a service component in SoaML.

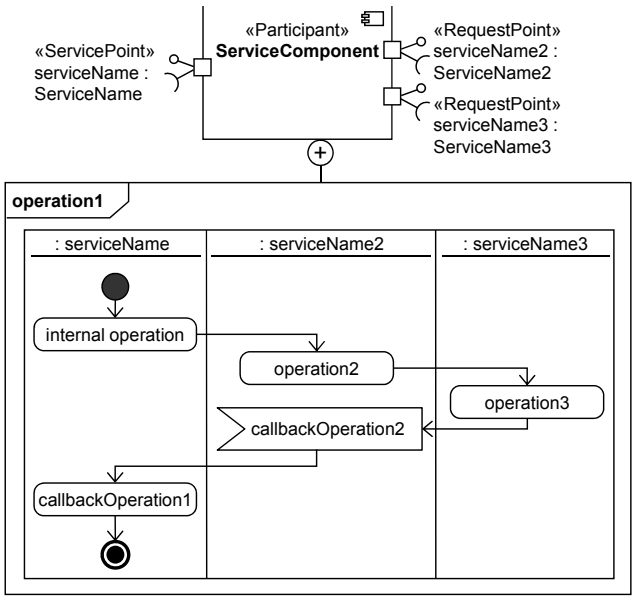


Figure 4. Modeling a service component

The internal logic is modeled using one UML Activity for each operation that is performed by the service component. The Activity is added as OwnedBehavior and named after this operation. It contains one UML Partition for each ServicePoint and RequestPoint. If an operation of a required service is called, a CallOperationAction is added to the according Partition. For receiving callbacks an AcceptEvent is used. If the service component performs functionality by itself, i.e. functionality that is not provided by an external service, an OpaqueAction is added to the Partition that represents the ServicePoint.

B. Evaluating Services

According to Boehm [29] and McCall et al. [30], within the Factor Criteria Metrics (FCM) quality model, the software quality as factor can be broken down into several criteria that can be further be described by metrics. When considering the flexibility, maintainability etc. as factor, the quality attributes considered within this article, such as loose coupling, can be thought of as criteria. Since the criteria are not measurable, they have to be further refined into metrics that equal quality indicators as introduced by the International Organization for Standardization (ISO) [31]. Thus, in order to evaluate service designs with regard to quality attributes that have been identified as criteria for services of high quality, quality indicators have to be determined and formalized as metrics. Within existing work either a description of quality attributes or metrics that enable the measurement of quality indicators are focused.

1) Description of Quality Attributes: In [3, 4], Erl introduces a comprehensive set of design principles and design patterns for services that can be thought of as quality attributes. The design principles are described in detail, however only textual information is provided. Quality indicators that enable an evaluation of service designs with regard to quality attributes are not introduced. Only some

so-called service characteristics are described that represent the impact of the design principles. However, these service characteristics are mostly not directly measurable and an explanation how to evaluate service designs with regard to these service characteristics is missing. Also some of these characteristics can only be evaluated if implementation details and deployment information are available.

For the Rational Unified Process for Service Oriented Modeling and Architecture (RUP SOMA), IBM lists quality attributes that have to be considered [21, 22, 23]. However, the quality attributes are only listed and considered as important, but a detailed description about these attributes and how to measure them is not provided likewise.

Engels et al. describe in [8] a method to design an application landscape and focus on the development process. Necessary steps to derive services from a prior analyzed business are explained and during the design phase several quality attributes are named. Also in this case, the quality attributes are only described textually. Quality indicators or metrics are not included. Similarly, Reussner et al. [6], Josuttis [7] and Maiers et al. [11, 12, 13] introduce quality attributes for services within service-oriented architectures. Also in this work, the textual description is focused and quality indicators or metrics are missing.

2) Formalized Metrics: Other work focuses on the formalization of metrics to evaluate services. Perepletchikov et al. [14, 15, 16] introduce metrics to measure cohesion and coupling of services. The metrics base on an extension of the generic software model of Briand [32]. Rud et al. [17] show metrics for measuring the granularity of services and Hirzialla et al. [18] focus on the flexibility. Choi et al. [19] measure the reusability of services. All this work has in common that it is not mainly meant for evaluating service designs as introduced in the section before. In some cases information about the implementation of the service or its deployment is required. The metrics that concentrate on the design of services and the information available during this time base on own notations and own understandings about how to design a service. For using the metrics on a common and standardized language, such as SoaML, first the used concepts within the metrics have to be transferred into representations within the formalized service designs. For this, a prior interpretation of the metric and a detailed understanding about the used concepts is necessary in order to correctly apply the metric. Additionally, some of the introduced metrics are not related to common and widespread quality attributes which hampers their incentive. to measure them.

C. Discussion

As illustrated above, SoaML provides all necessary elements to model service designs. However, SoaML does not provide any information about how to design services with certain quality attributes. On the other side, work in the context of quality attributes and formalized metrics is mostly not directly applicable due to the fact that the textual descriptions are too abstract. Furthermore, the formalized metrics are often not related to widespread quality attributes and they use concepts and understandings of service designs that have to be mapped onto elements of concrete languages first. This step requires an interpretation of the metrics and may include interpretation mistakes. Thus, in this article the quality attributes and their textual descriptions are used for deriving quality indicators and formalized metrics that can be applied on a formalized service design based on SoaML. The metrics are described by means of a similar notation as introduced by Perepletchikov et al. [14]. Due to the direct derivation of quality indicators from quality attributes and the usage of SoaML, the motivation why to measure them is given and a direct application without any interpretation effort is enabled.

III. Metrics for Evaluating Service Designs Based on SoaML

In order to evaluate service designs based on SoaML, quality indicators have to be identified that give the IT architect hints about the current value of the quality attributes. In this article, the four quality attributes of a unique categorization, loose coupling, discoverability, and autonomy are considered. These quality attributes mostly require information that is available during design time compared to quality attributes, such as statelessness and idempotence, which require additional information [1].

To illustrate the quality indicators, in this article the human-centered environmental observation domain is considered. This domain refers to the network-enabled surveillance and tracking system as introduced by the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [38, 39]. Currently, at the Karlsruhe Institute of Technology (KIT) the KITCampusGuide, a system to provide a guide for students, lecturers and guest, is developed. A person can ask for another person or a room on the campus of the university using mobile devices, such as a mobile phone, and the KITCampusGuide calculates the route. The following figure shows the KITCampusGuide in action. A requirement for the KITCampusGuide is to create services for this scenario that fulfill the quality attributes of a unique categorization, loose coupling, autonomy and discoverability as introduced in [2]. This is why this scenario is chosen to illustrate the quality indicators and their formalizations.

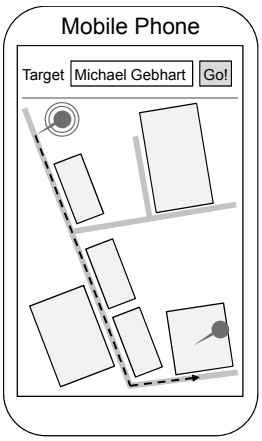


Figure 5. KITCampusGuide in action

Since some of the quality indicators require knowledge about the functional terms used within this scenario, the domain is modeled using an ontology based on the Web Ontology Language (OWL) [35]. As modeling tool Protégé [42] is applied. For illustrating the ontology we choose a notation that is similar to the OntoGraf in Protégé. Each concept is depicted by a rectangle and the relations between these concepts are represented by lines between these rectangles. In order to provide the name of the concepts and relations in various languages, the translations are added as labels. A suffix specifies the language, such as "@de" for German. The following figure shows an excerpt of the domain model for the human-centered environmental observation.

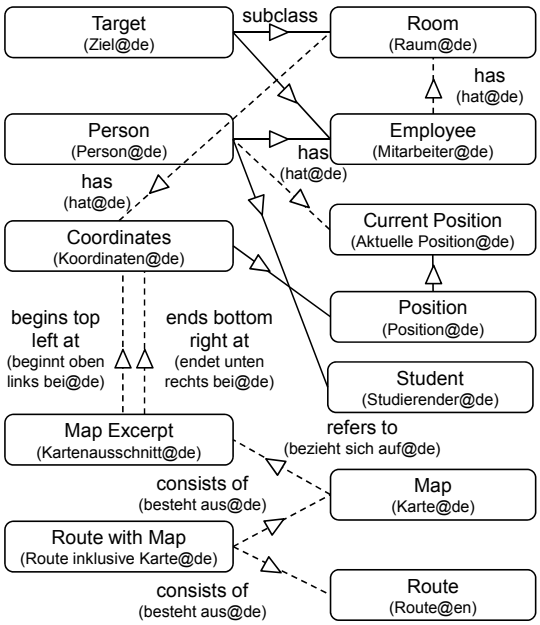


Figure 6. Excerpt of the domain model

For identifying quality indicators, the descriptions of the quality attributes are analyzed and aspects that can be measured within a service design are extracted. Each of these aspects represents a quality indicator. Afterwards, for each quality indicator a formalized metric is created. This formalization uses elements of a service design which enables its direct applicability. To interpret the value of the metric correctly, a scale is assigned. An exemplarily application of the metric on a service design of our scenario helps to illustrate the metric and verifies its validity.

A. Unique Categorization

Erl [3], Cohen [10], Engels et al. [8], and Maier et al. [12] introduce approaches for a unique categorization. The categorization has its origin in splitting different and bundling similar kinds of functionality. Thus, the unique categorization corresponds to the concept of cohesion [3]. Common categories are entity services that are responsible for managing business entities, task services that provide mostly process-specific functionality beyond the scope of one business entity, and utility services for cross-cutting technical functionality. This categorization can be broken down into different aspects that represent quality indicators.

1) Division of Business-Related and Technical Functionality: According to Reussner et al. [6], functionality that changes in different time intervals, such as business-related and technical functionality, should be split into several modules, in this case services because this increases their maintainability. Business-related functionality refers to the logic of the business domain, whilst technical functionality includes cross-cutting technical functionality, as for instance functionality of logging systems or security systems. The division of these different kinds of functionality categorizes services into entity and task services on the one side and utility services on the other side.

On the basis of service candidates the division can be verified by means of the contained operation candidates. Appropriate knowledge about the purpose of these operation candidates assumed, the following metric can be formalized.

$$DBTF(sc) = \frac{|\,BF(OC(sc))\,|}{|\,OC(sc)\,|}$$

In case of service designs, instead of the operation candidates the operations of the realized interface are considered.

$$DBTF(s) = \frac{\left|\,BF\left(O\left(RI(SI(s))\right)\right)\right|}{\left|\,O\left(RI(SI(s))\right)\right|}$$

The metrics are only valid if there exists any operation candidate respectively operation. Within the metrics, the following variables and functions are used.

TABLE I. VARIABLES AND FUNCTIONS USED FOR DBTF

Element	Description
DBTF	Division of Business-related and Technical Functionality
sc	service candidate: the considered service candidate
s	service: the considered service that is provided or required, represented by an ServicePoint or RequestPoint in SoaML
BF(oc)	Business Functionality: operation candidates providing business-related functionality out of the set of operation candidates oc
BF(o)	Business Functionality: operations providing business-related functionality out of the set of operations o
OC(sc)	Operation Candidates: operation candidates of the service candidate sc
SI(s)	Service Interface: service interface of the service s. In SoaML it is the type of the ServicePoint or RequestPoint s.
RI(si)	Realized Interfaces: realized interfaces of the service interface si
O(i)	Operations: operations within the interface i
| oc |	Number of operation candidates oc
| o |	Number of operations o

Thus, the metrics return values from 0 to 1, which have an order. Accordingly, the results are interpreted within the ordinal scale. The following table shows the interpretation of values for DBTF.

TABLE II. INTERPRETATION OF VALUES FOR DBTF

Value	Interpretation
0	Only technical functionality is provided
Between 0 and 1	Both business-related and technical functionality is provided
1	Only business-related functionality is provided

According to this table, for DBTF a value of 0 or 1 is desired because these values represent a division of business-related and technical functionality. A value between 0 and 1 should be avoided. The following figure depicts one service candidate and one interface realized by a service interface. The former includes both operation candidates with business-related and technical functionality. The latter includes only business-related functionality. Thus, the design of the service candidate should be revised, whilst the design of the interface follows the criteria for a service that can be uniquely categorized. This circumstance is also confirmed by the metric DBTF. For the service candidate a value of 0.5 and for the service interface a value of 1 is returned.

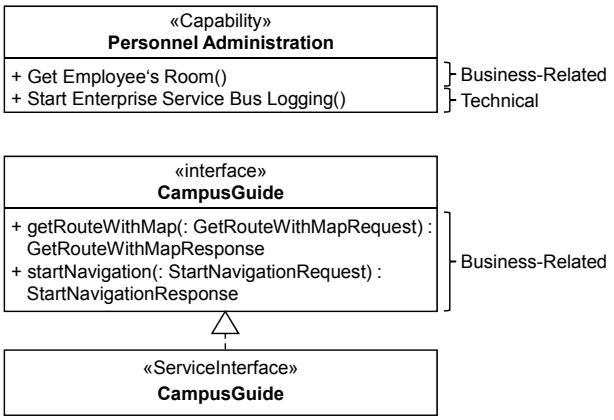


Figure 7. Example for division of business-related and technical functionality

2) Division of Agnostic and Non-agnostic Functionality: In order to increase the reusability of services, agnostic functionality that is agnostic should be divided from non-agnostic functionality [3]. Agnostic functionality is highly reusable and not process-specific, whilst non-agnostic functionality is less reusable and mostly process-specific. A usage of non-agonstic functionility in several processes is not expected. The division of this functionality results in the distinction of agnostic services, such as entity services, and non-agnostic services, such as task services.

Similarly to the division of business-related and technical functionality, on the basis of service candidates the division of agnostic and non-agnostic functionality can be evaluated by considering the operation candidates. Equivalently to DBTF, the following metric measures the ratio of operation candidates providing agnostic functionality to all operation candidates.

$$DANF(sc) = \frac{|\,AF(OC(sc))\,|}{|\,OC(sc)\,|}$$

If service designs are supposed to be evaluated, instead of the operation candidates the operations of the realized interface are used within the metric.

$$DANF(s) = \frac{\left|\,AF\left(O\left(RI(SI(s))\right)\right)\right|}{\left|\,O\left(RI(SI(s))\right)\right|}$$

The metrics are only valid if there is at least one operation candidate respectively one operation provided. Variables and functions that are used additionally to those introduced for DBTF are listed below.

TABLE III. VARIABLES AND FUNCTIONS USED FOR DANF

Element	Description
DANF	Division of Agnostic and Non-agnostic Functionality
AF(oc)	Agnostic Functionality: operation candidates providing agnostic functionality out of the set of operation candidates oc
AF(o)	Agnostic Functionality: operations providing agnostic functionality out of the set of operations o

The metrics return values from 0 to 1. Also in this case, the results are interpreted within the ordinal scale.

TABLE IV. INTERPRETATION OF VALUES FOR DANF

Value	Interpretation
0	Only non-agnostic functionality is provided
Between 0 and 1	Both agnostic and non-agnostic functionality is provided
1	Only agnostic functionality is provided

In order to increase the unique categorization, a value of 0 or 1 is desired. The following figure shows a service candidate with both agnostic and non-agnostic operation candidates. In order to increase the reusability, these operation candidates should be divided. This is also confirmed by the value 0.5 that is returned for DANF.

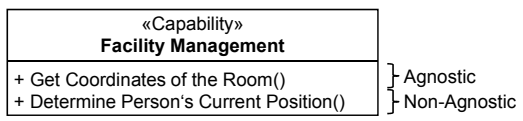


Figure 8. Example for division of agnostic and non-agnostic functionality

3) Data Superiority: If a service manages a certain business entity, this service should be explicitly managing this entity. For example, if a service is responsible for creating, deleting or changing a business entity, no other service should provide similar functionality. This concept is

called data superiority [8]. According to Erl [3] and Cohen [10], a service that fulfills the data superiority corresponds to an entity service.

On the basis of service candidates, the business entities managed by the contained operation candidates have to be assumed and compared with the business entities managed by operation candidates of other services. Optimally, there should be no overlap.

$$DS(sc) = 1 - \frac{\left| MBE\left(OC(sc)\right) \cap MBE\left(OC(ALL_{SC} \setminus sc)\right) \right|}{\left| MBE\left(OC(sc)\right) \right|}$$

On the basis of service designs the metric can be formalized as follows.

$$DS(s) = 1 - \frac{\left| \begin{array}{c} MBE\left(O\left(RI\left(SI(s)\right)\right)\right) \cap \\ MBE\left(O\left(RI\left(SI\left((ALL_S \setminus s)\right)\right)\right)\right) \end{array} \right|}{\left| MBE\left(O\left(RI\left(SI(s)\right)\right)\right) \right|}$$

TABLE V. VARIABLES AND FUNCTIONS USED FOR DS

Element	Description
DS	Data Superiority
M1 \ M2	Elements of set M1 without elements of set M2 or the element M2
ALL_{SC}	All existing service candidates
ALL_S	All existing services
MBE(oc)	Managed Business Entities: business entities that are managed by operation candidates oc
MBE(o)	Managed Business Entities: business entities that are managed by operations o

The metrics require that at least one business entity is managed by the service candidate respectively service. The metrics return values from 0 to 1. Based on the ordinal scale the results can be interpreted as follows.

TABLE VI. INTERPRETATION OF VALUES FOR DS

Value	Interpretation
Less than 1	No data superiority regarding the managed business entities
1	Data superiority regarding the managed business entities

For the metrics a value of 1 is desired. The following figure shows three service candidates that manage business entities. Assumed that there exist no other service candidates, for the service candidate Personal Management the value of DS is 1 because it manages explicitly the business entity Employee. For the service candidates Facility Management and Room, the value of DS is 0, for both manage the same business entities. In order to increase DS, the IT architect should consider a merger of these service candidates.

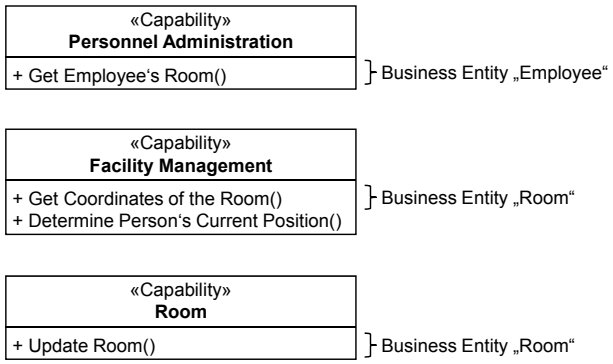


Figure 9. Example for data superiority and common business entity usage

4) Common Business Entity Usage: Additionally, the provided operations should use common business entities [PR+07] for ensuring that, for instance, an entity service focuses on one business entity only. This means that the business entities that are used as input parameters within operation candidates respectively operations should either be identical or should be dependent. A business entity depends from another business entity if it cannot exist for its own. This concept is comparable to the composition within UML [43].

On the basis of service candidates the common business entity usage can be evaluated using the operation candidates and their propably used business entities. First, all used business entities of the operation candidates and the within one operation candidate mostly often used business entities are determined. From these two sets of business entities the biggest set of common, i.e. depending business entities is created. Afterwards, the operation candidates that use these business entitites are identified and related to all operation candidates.

$$CBEU(sc) = \frac{\left| OCUBE\left(OC(sc), CMP\left(\begin{array}{c} MOUBE\left(OC(sc)\right), \\ UBE\left(OC(sc)\right) \end{array}\right)\right) \right|}{\left| OC(sc) \right|}$$

On service designs, the operations within the realized interface instead of operation candidates are used.

$$CBEU(s) = \frac{\left| OUBE\left(\begin{array}{c} O\left(RI\left(SI(s)\right)\right), \\ CMP\left(\begin{array}{c} MOUBE\left(O\left(RI\left(SI(s)\right)\right)\right), \\ UBE\left(O\left(RI\left(SI(s)\right)\right)\right) \end{array}\right) \end{array}\right) \right|}{\left| O\left(RI\left(SI(s)\right)\right) \right|}$$

The metrics require that there exists at least one operation candidate respectively one operation. Within these metrics, the following additional variables and functions are used.

TABLE VII. VARIABLES AND FUNCTIONS USED FOR CBEU

Element	Description
CBEU	Common Business Entity Usage
CMP(be1, be2)	Composition: biggest set of business entities out of be2 that depend on business entitites be1
UBE(oc)	Used Business Entities: business entities that are used within operation candidates oc as input
UBE(o)	Used Business Entities: business entities that are used within operations o as input
MOUBE(oc)	Mostly Often Used Business Entities: business entities that are mostly often used within one operation candidate out of operation candidates oc
MOUBE(o)	Mostly Often Used Business Entities: business entities that are mostly often used within one operation out of operations o
OCUBE(oc, be)	Operation Candidates Using Business Entities: operation candidates out of operation candidates oc that only use business entities out of be
OUBE(o, be)	Operations Using Business Entities: operations out of operations o that only use business entities out of be

The metrics return results from 0 to 1. The interpretation is listed below based on the ordinal scale.

TABLE VIII. INTERPRETATION OF VALUES FOR CBEU

Value	Interpretation
Less than 1	There exist operation candidates respectively operations that use non-common business entitites
1	All operation candidates respectively operations use common business entitites

For CBEU a value of 1 is desired because this value represents the case that all operation candidates or operations use common business entities. Applied on Figure 9, the service candidate Facility Management contains one operation candidate that uses the Room entity and one operation candidate that uses the Person entity as input. Since a room can exist without a person and vice versa, the metric returns a value of 0.5. A merge of the service candidate Room with the service candidate Facility Management as already proposed to improve the data superiority quality indicator would increase the value for CBEU. An optimal value could be achieved if the two contained operation candidates would be divided into two different service candidates.

B. Discoverability

Since reusability of existing functionality is one major aspect when establishing a service-oriented architecture, services are required to be discoverable. The discoverability is already positively influenced by a unique categorization. However, there are other aspects, such as naming conventions, which have to be considered. These aspects constitute quality indicators that influence the discoverability.

1) Functional Naming: The first quality indicator focuses on the functional naming of the created artifacts. According to Josuttis [7], in order to understand the functionality a service provides, the description of a service has to follow functional terms that have been determined for the considered domain. This means that the elements of a service, such as its interface, the roles, and the operations, have to be named after functional terms as they are determined within a domain model. Thus, the functional naming of a service can be further broken down into a functional naming of its externally visible artifacts, i.e. of the service interface, the roles, the operations, the parameters and the data types.

Since the service candidates are mainly meant to describe the architecture, i.e. the services and their dependencies in an abstract manner, the naming of artifacts constitutes a quality indicator only of interest on the basis of service designs. Thus, the following metrics for evaluating the functional naming are only measurable on service designs. The metrics determine the ratio of functionally named artifacts compared to all artifacts. For evaluating the functional naming of the service interface, the following metric can be formalized:

$$FNSI(s) = \frac{|FN(SI(s))|}{|SI(s)|}$$

The metrics for evaluating the functional naming of roles, operations, parameters, and data types can be formalized equivalently.

$$FNR(s) = \frac{\left|FN\left(R(SI(s))\right)\right|}{\left|R\left(SI(s)\right)\right|}$$

$$FNO(s) = \frac{\left|FN\left(O\left(RI(SI(s))\right)\right)\right|}{\left|O\left(RI(SI(s))\right)\right|}$$

$$FNP(s) = \frac{\left|FN\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right|}{\left|P\left(O\left(RI(SI(s))\right)\right)\right|}$$

$$FNDT(s) = \frac{\left|FN\left(DT\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right)\right|}{\left|DT\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right|}$$

TABLE IX. VARIABLES AND FUNCTIONS USED FOR FNSI, FNR, FNO, FNP, AND FNDT

Element	Description
FNSI	Functional Naming of Service Interface
FNR	Functional Naming of Roles
FNO	Functional Naming of Operations
FNP	Functional Naming of Parameters

FNDT	Functional Naming of Data Types
FN(me)	Functional Naming: set of functionally named elements out of the set of modelling elements me
P(o)	Parameters: parameters of the operations o and in case of messages the contained parameters
DT(p)	Data Types: used data types (recursively continued) of parameters p
R(si)	Roles: roles of service interface si

Each of the metrics is only valid if there is at least one role, one operation, one parameter, respectively one data type specified within the considered service design. Otherwise, the metric cannot be applied. As result, values from 0 to 1 are returned. The interpretation based on the ordinal scale is shown below.

TABLE X. INTERPRETATION OF VALUES FOR FNSI, FNR, FNO, FNP, AND FNDT

Value	Interpretation
Less than 1	There are elements that are not functionally named
1	All elements are functionally named

For the metrics FNSI, FNR, FNO, FNP, and FNDT, a value of 1 is desired. For example, based on the domain model as depicted in Figure 6, the FNO for the CampusGuide in Figure 7 is 0.5, for the term navigation is not part of the domain model. The IT architect has to decide if the term is functional and was accidently not added to the domain model. In this case the domain model should be revised. Otherwise the name of the operation should be changed. Thus, this metric does not only evaluate service designs. It also helps IT architect to validate the prior created domain model.

2) Naming Convention Compliance: The second quality indicator for the discoverability addresses the compliance with naming conventions. According to Maier et al. [13], the compliance with naming conventions increases the discoverability of a service. Typical naming conventions are the usage of the english language, verbs possibly followed by nouns for the names of operations, and upper case letters at the beginning of the name of service interfaces and data types. These naming conventions are necessary to create consistently named artifacts.

Similarly to the functional naming, this quality indicator is only of interest for service designs. The quality indicator can be further broken down into a naming convention compliance of the service interface, the roles, the operations, the parameters, and the data types. The metrics are similarly formalized to the metrics for the functional naming and determine the ratio of artifacts named regarding naming conventions to all artifacts. The metrics are only defined if the particular artifacts, i.e. the service interface, the roles, provided operations, used parameters, and data types are specified. Otherwise the metrics cannot be applied on the considered service design.

$$NCCSI(s) = \frac{|NCC(SI(s))|}{|SI(s)|}$$

$$NCCR(s) = \frac{\left|NCC\left(R(SI(s))\right)\right|}{\left|R\left(SI(s)\right)\right|}$$

$$NCCO(s) = \frac{\left|NCC\left(O\left(RI(SI(s))\right)\right)\right|}{\left|O\left(RI(SI(s))\right)\right|}$$

$$NCCP(s) = \frac{\left|NCC\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right|}{\left|P\left(O\left(RI(SI(s))\right)\right)\right|}$$

$$NCCDT(s) = \frac{\left|NCC\left(DT\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right)\right|}{\left|DT\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right|}$$

TABLE XI. VARIABLES AND FUNCTIONS USED FOR NCCSI, NCCR, NCCO, NCCP, AND NCCDT

Element	Description
NCCSI	Naming Convention Compliance of Service Interface
NCCR	Naming Convention Compliance of Roles
NCCO	Naming Convention Compliance of Operations
NCCP	Naming Convention Compliance of Parameters
NCCDT	Naming Convention Compliance of Data Types
NCC(me)	Naming Convention Compliance: set of elements out of the set of modelling elements me that follow specified naming conventions

The metrics return values from 0 to 1, interpreted on the basis of the ordinal scale.

TABLE XII. INTERPRETATION OF VALUES FOR NCCSI, NCCR, NCCO, NCCP, AND NCCDT

Value	Interpretation
Less than 1	There are elements that do not follow naming conventions
1	All elements follow naming conventions

The following figure depicts an interface realized by a service interface and the evaluation regarding naming conventions. According to this figure and the available information, the NCCSI is 0, NCCR is 0.5, NCCO is 0.5, NCCP is 0.75, and NCCDT is 0. Thus, the IT architect should revise the service designs and especially the names of the artifacts in order to increase the discoverability of the resulting service.

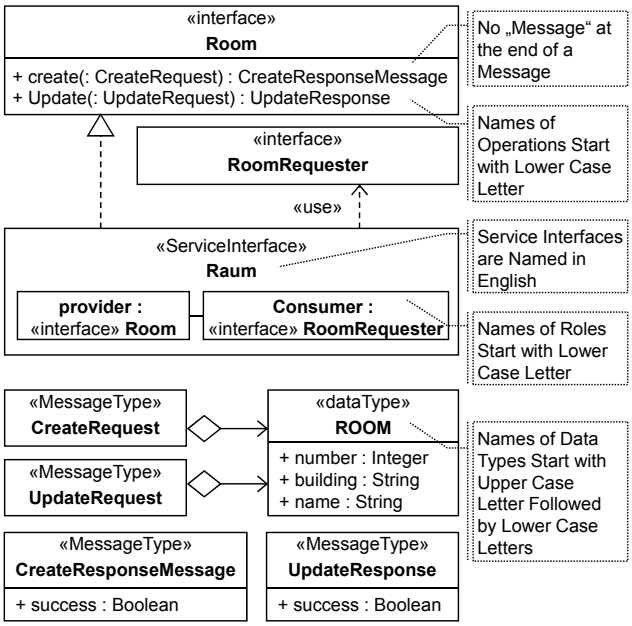


Figure 10. Example for naming convention compliance

3) Information Content: The service interface describes a service from an external point of view for potential service consumers. According to Erl [3], the extent of this information influences the discoverability of a service.

Transferred to SoaML, a service interface should contain as much information as possible. This means that the service interface, the contained roles, the realized interface, the used interface and the interaction protocol should be formalized. As metric the extent of the information content can be described as follows:

$$IC(s) = \frac{EX(SI(s)) + EX\left(RI(SI(s))\right) + EX\left(UI(SI(s))\right) + EX\left(R(SI(s))\right) + EX\left(IP(SI(s))\right)}{5}$$

TABLE XIII. VARIABLES AND FUNCTIONS USED FOR IC

Element	Description
IC	Information Content
EX(e)	Exists: returns 1 if the element e exists, else 0
IP(si)	Interaction Protocol: interaction protocol of the service interface si
UI(si)	Used Interfaces: used interface provided by the service consumer

As result, values from 0 to 1 are returned. The interpretation based on the ordinal scale is shown in the following table. For IC a value of 1 is desired for this value represents the case that all possible information is available within the service design.

TABLE XIV. INTERPRETATION OF VALUES FOR IC

Value	Interpretation
Less than 1	Within the service design not all possible information is available
1	All possible information is available

For example, the information content, thus the metric IC for the service interface depicted in Figure 10 is 0.8. The value of IC can be increased and maximized by adding the interaction protocol to the service interface.

C. Loose Coupling

The loose coupling focuses on the reduction of dependencies between services within a service-oriented architecture and represents one of the most widespread aspects. A loose coupling promotes the scalability, fault tolerance, flexibility, and maintainability of the architecture [6, 7, 8, 16, 20]. Once a service requires another service to fulfill its functionality, a certain kind of coupling exists. However, in order to decrease the coupling, the following aspects can be considered that represent quality indicators for a loose coupling.

1) Asynchronity: According to Josuttis [7] and Maier et al. [11], long-running operations should be performed asynchronously. This means that the service consumer is being informed when the service provider has performed the called operation. This decouples the service consumer from the service provider during the execution of the operation.

In SoaML the communication mode is determined during the specification phase, i.e. this quality indicator can be evaluated on the basis of service designs. Within the interaction protocol, the IT architect can decide whether to provide an operation synchronously or asynchronously. For this purpose the attribute “IsSychronous” of the CallOperationActions within the Activity that represents the interaction protocol can either be set true or false. Thus, to determine this quality indicator the rate of long-running operations that are also asynchronous has to be measured.

$$ASYNC(s) = \frac{\left|ASO\left(IP(SI(s))\right) \cap LRO\left(O\left(RI(SI(s))\right)\right)\right|}{\left|LRO\left(O\left(RI(SI(s))\right)\right)\right|}$$

TABLE XV. VARIABLES AND FUNCTIONS USED FOR ASYNC

Element	Description
ASYNC	Asynchronity
ASO(ip)	Asynchronous Operations: asynchronous operations within the interaction protocol ip
LRO(o)	Long Running Operations: long-running operations out of the set of operations o

The metric is only valid if there is at least one long-running operation. The following table shows the interpretation of the values based on the ordinal scale. The metric returns values from 0 to 1.

TABLE XVI. INTERPRETATION OF VALUES FOR ASYNC

Value	Interpretation
Less than 1	There are long-running operations that are not provided asynchronously
1	All long-running operations are provided asynchronously

The following figure shows the interaction protocol for the service interface depicted in Figure 10. Assumed that the Update operation is a long-running operation, ASYNC returns 0 as Update is not provided asynchronously.

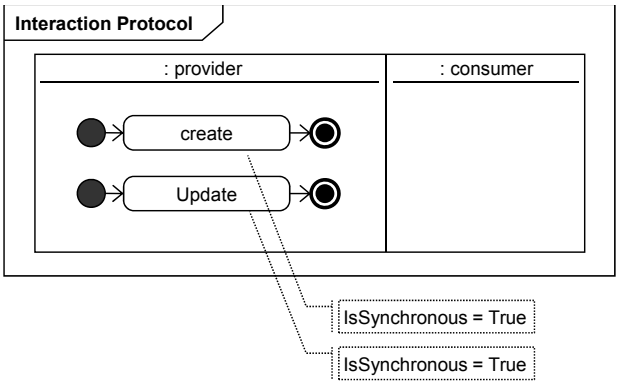


Figure 11. Example for asynchrony

2) Common Data Types Complexity: The usage of common data types across several services increases the coupling between them. If a service provider aspires to change the data types used by provided services, all other services that require this service have to be adapted too. Thus, Josuttis [7] advises to only use common data types if they are simple data types, such as String or Integer. Otherwise, the services should use own data types and the infrastructure should handle the transformation.

For this purpose, in SoaML the service designs have to be considered. The used parameters within the operations of a certain service and the contained data types should not be identical to the data types used by other services. A typical way to avoid this issue is to create identical data types, however within different UML packages for each created service. The following metric measures the ratio of common and simple data types to all used data types. For an optimal value, either there is no common data type or all commonly used data types are simple. The metric is only valid if the required artifacts, such as operations and data types, exist within the service design.

$$CDTC(s) = \frac{\left| SDT\left(\begin{array}{c} DT\left(P\left(O\left(RI\left(SI(s)\right)\right)\right)\right) \cap \\ DT\left(P\left(O\left(RI\left(SI(ALL_s \setminus s)\right)\right)\right)\right) \end{array} \right) \right|}{\left| DT\left(P\left(O\left(RI\left(SI(s)\right)\right)\right)\right) \right|}$$

TABLE XVII. VARIABLES AND FUNCTIONS USED FOR CDTC

Element	Description
CDTC	Common Data Types Complexity
SDT(p)	Simple Data Types: simple data types within the parameters pt

The metric CDTC returns values from 0 to 1. The interpretation based on the ordinal scale is shown in the following table.

TABLE XVIII. INTERPRETATION OF VALUES FOR CDTC

Value	Interpretation
0	There are no common data types used
Between 0 and 1	There are common and complex data types used
1	The commonly used data types are simple

The optimal value is represented by 0 or 1. A value between 0 and 1 should be avoided.

3) Abstraction: To decrease the coupling between service consumer and service provider, a service consumer should be able to use provided functionality without knowledge about the internal behavior of the service provider. This enables the invocation of functionality without knowledge about the implementation and an easier replacement of the implementation or the service provider. According to Erl [3], Josuttis [7] and Maier et al. [11], the operations should be designed in an abstract manner and should hide internal details. Thus, the quality indicator can be broken down into two quality indicators: First, the operations provided by the service should be abstract, i.e. the name and their purpose should be abstract. Additionally, the used parameter should be abstract, i.e. implementation details should not be exchanged when invoking an operation.

Thus, in SoaML the first quality indicator focuses on the operation of the interface that is realized by the service interface. The abstract operations are related to all provided operations. The second quality indicator regards the parameters that are used within the operations. The metric measures the ratio of abstract parameters to all parameters.

$$AO(s) = \frac{\left| A\left(O\left(RI\left(SI(s)\right)\right)\right) \right|}{\left| O\left(RI\left(SI(s)\right)\right) \right|}$$

$$AP(s) = \frac{\left| A\left(P\left(O\left(RI\left(SI(s)\right)\right)\right)\right) \right|}{\left| P\left(O\left(RI\left(SI(s)\right)\right)\right) \right|}$$

TABLE XIX. VARIABLES AND FUNCTIONS USED FOR AO AND AP

Element	Description
AO	Abstraction of Operations
AP	Abstraction of Parameters
A(o)	Abstract: set out of operations o that are abstract
A(p)	Abstract: set out of parameters p that are abstract

For AO and AP values from 0 to 1 are returned that can be interpreted on the basis of the ordinal scale as follows.

TABLE XX. INTERPRETATION OF VALUES FOR AO AND AP

Value	Interpretation
Less than 0	There exist operations respectively parameters that are not abstract
1	All operations respectively parameters are abstract

According to the table above, a value of 1 is desired for both metrics because this represents that all operations respectively parameters are abstract which promotes the loose coupling. The following figure shows an interface that is realized by a service interface. One of the provided operations is not abstract, thus the value for AO is 0.5. The value of AP is 0.4 for five parameters are specified and only two of them are abstract.

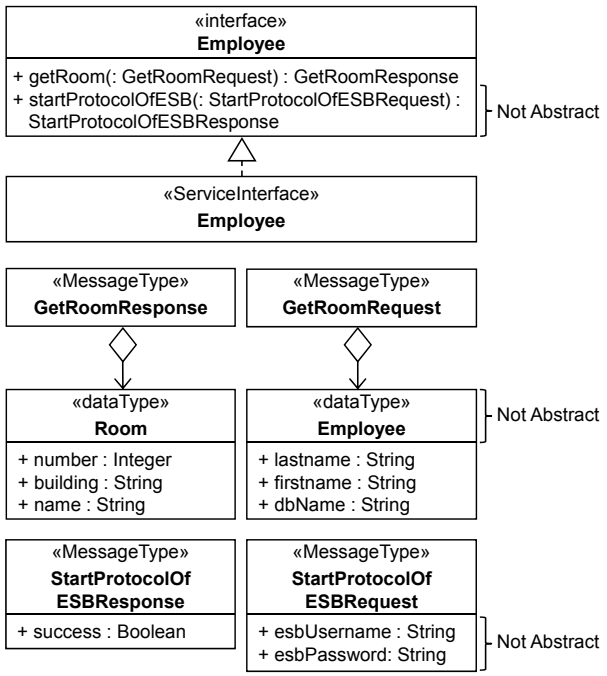


Figure 12. Example for abstraction

4) Compensation: According to Josuttis [7] and Engels et al. [8], in order to undo operations of a service that perform state-changing functionality, appropriate compensation operations should be provided. This enables the application of this service within transaction contexts, which requires an undo even if other services are the reason for a failure.

This quality indicator can already be measured on the basis of service candidates. For every operation candidate that represents a state-changing operation, an appropriate compensating operation candidate should exist. The metric first determines operation candidates that are not mainly compensating and change a state. Afterwards, out of this set the operation candidates are identified for those a compensating operation candidate exists. This set is related to the set of all non-compensating and state-changing operation candidates.

$$CF(sc) = \frac{\left|CFP\left(SC\left(NC(OC(sc))\right)\right)\right|}{\left|SC\left(NC(OC(sc))\right)\right|}$$

On the basis of service designs, instead of operation candidates the operations within the realized interface are considered.

$$CF(s) = \frac{\left|CFP\left(SC\left(NC\left(O\left(RI(SI(s))\right)\right)\right)\right)\right|}{\left|SC\left(NC\left(O\left(RI(SI(s))\right)\right)\right)\right|}$$

TABLE XXI. VARIABLES AND FUNCTIONS USED FOR CF

Element	Description
CF	Compensating Functionality
NC(oc)	Non-Compensating: non-compensating operation candidates out of the set of operation candidates oc
NC(o)	Non-Compensating: non-compensating operations out of the set of operations o
SC(oc)	State Changing: operation candidates out of the set of operation candidates oc that provide state-changing functionality
SC(o)	State Changing: operations out of the set of operations o that provide a state-changing functionality
CFP(oc)	Compensating Functionality Provided: operation candidates out of the set of operation candidates oc a compensating operation candidate exists for
CFP(o)	Compensating Functionality Provided: operations out of the set of operations o a compensating operation exists for

The metric CF returns values from 0 to 1. The interpretation on the basis of the ordinal scale is shown below.

TABLE XXII. INTERPRETATION OF VALUES FOR CF

Value	Interpretation
Less than 0	There exist state-changing operation candidates respectively operations without compensating operations candidates respectively operations
1	For all operation candidates respectively operations that provide state-changing functionality a compensating operation candidate respectively operation exists

For the service interface depicted in Figure 10 the value of CF is 0.5 because for Update there exists a compensating operation, the Update operation itself. However, for the create operation there is no compensating operation. In order to increase the value of CF and thus the loose coupling, a delete operation should be added that enables the deletion of a prior created room.

D. Autonomy

The autonomy of a service addresses its independence from other services [3, 7]. For increasing the autonomy of a service dependencies to other services have to be reduced. For the autonomy the following quality indicators can be identified.

1) Service Dependency: According to Erl [3], the direct dependencies to other services should be decreased. If a service depends from other services also its reliability, performance and predictability is influenced by these services.

In SoaML the direct dependencies can be evaluated on the basis of the usage dependencies between service candidates and the RequestPoints within service components. Both represent the dependencies of a service to other services in order to fulfill its functionality.

$$SD(sc) = |\,RS(sc)|$$

$$SD(s) = \big|\,RS\big(SCT(s)\big)\,\big|$$

TABLE XXIII. VARIABLES AND FUNCTIONS USED FOR SD

Element	Description
SD	Service Dependency
RS(sc)	Required Services: service candidates the service candidate sc depends on
SCT(s)	Service Component: service component of the service s
RS(sct)	Required Services: services the service component sct depends on

SD returns values from 0 to unlimited. The interpretation is based on the absolute scale.

TABLE XXIV. INTERPRETATION OF VALUES FOR SD

Value	Interpretation
0	the service candidate or the functionality fulfilling service component depends on no other service candidate respectively service
n (n > 0)	the service candidate or service component requires n other services to fulfill its functionality

For a maximal autonomy the value of SD should be 0. However, especially in the context of service-oriented architectures the reuse of existing functionality should be considered, which decreases the autonomy. Additionally, for improving other quality attributes, such as the unique categorization, the autonomy often has to be reduced too.

The following figure shows a service component that provides the service CampusGuide. Due to the five RequestPoints, the metric SD for CampusGuide returns 5.0.

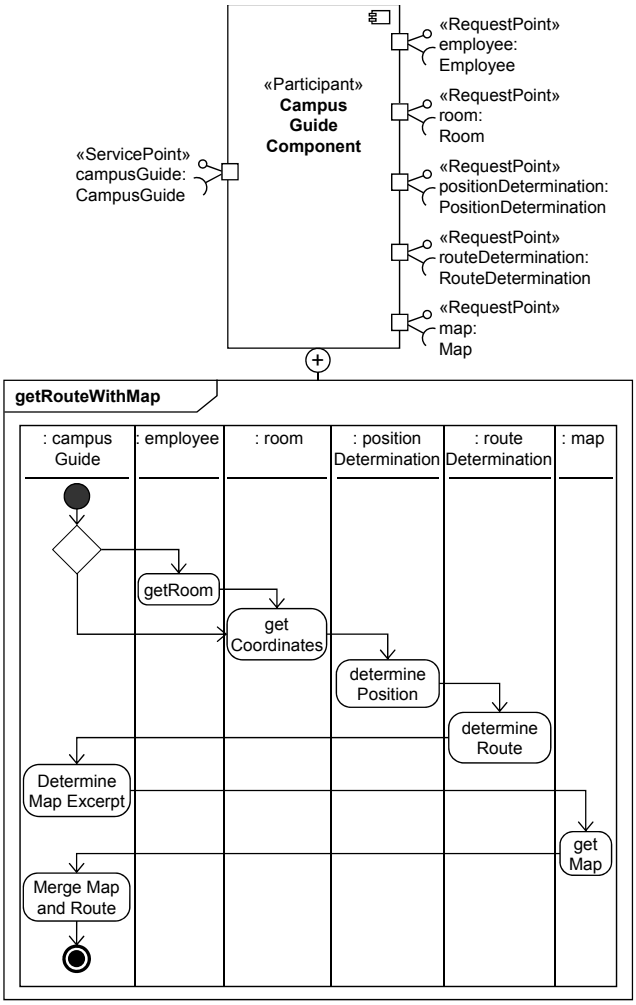


Figure 13. Example for service dependency

2) Functionality Overlap: According to Erl [3], a clear specification of the functional boundary of a service increases its autonomy. This means that the functionality of a service should not overlap with functionality of other services. Background of this requirement is that an overlap often results in dependencies between these services. In order to use functionality of a certain service, due to the overlap, also the functionality of the other services is necessary. Thus, the service with overlapping functionality can only be used together with other services. For avoiding functionality overlaps, services should be normalized [4].

The functionality overlap can be determined both on service candidates and service designs. For evaluating a service candidate its operation candidates have to be compared with the operation candidates of other service candidates. Afterwards, the set of operation candidates that provide redundant functionality are related to all provided operation candidates.

$$FO(sc) = \frac{|\, RF(OC(sc), OC(ALL_{sc} \setminus sc)) \,|}{|\, OC(sc) \,|}$$

On the basis of service designs, the functionality overlap is determined by means of the operations within the realized interface.

$$FO(s) = \frac{\left| RF\left(O\left(RI(SI(s))\right), O\left(RI(SI(ALL_s \setminus s))\right)\right)\right|}{\left| O\left(RI(SI(s))\right)\right|}$$

TABLE XXV. VARIABLES AND FUNCTIONS USED FOR FO

Element	Description
FO	Functionality Overlap
RF(oc1, oc2)	Redundant Functionality: operation candidates out of the set of operation candidates oc1 with redundant functionality to the operation candidates oc2
RF(o1, o2)	Redundant Functionality: operations out of the set of operations o1 with redundant functionality to the operations o2

The metrics return values from 0 to 1. The following table shows the interpretation of the values based on the ordinal scale.

TABLE XXVI. INTERPRETATION OF VALUES FOR FO

Value	Interpretation
0	The operation candidates respectively operations of the considered service candidate or service do not provide functionality that overlaps with functionality of other service candidates or services
Between 0 and 1	The operation candidates respectively operations of the considered service candidate or service provide functionality that overlaps with functionality of other service candidates or services
1	The operation candidates respectively operations of the considered service candidate or service provide only functionality that overlaps with functionality of other service candidates or services

Based on the service candidates depicted in Figure 9, the following figure shows a service candidate with functionality overlap. The metric returns 0.5, for half of the provided operation candidates overlap with functionality provided by the Facility Management service candidate.

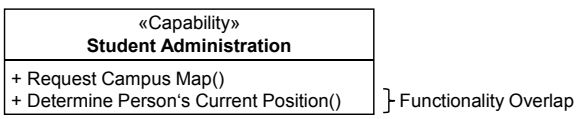


Figure 14. Example for functionality overlap

IV. CONCLUSION AND OUTLOOK

In this article we presented metrics for evaluating service designs based on SoaML. The approach uses the textual descriptions for quality attributes as introduced in existing work and analyses these quality attributes with regard to quality indicators, which are measureable on service candidates as preliminary services and fully specified service designs. The concept of a service design was derived from existing development processes. For each of the quality indicators formalizations were given that reuse the concepts of service candidates and service designs and their specification in SoaML. This enables the application of the formalization and thus the determination of the quality indicators on modeled service designs without additional interpretation effort.

The identification of quality indicators and their formalizations help IT architects to comprehensibly evaluate service designs with regard to common and widespread quality attributes. In this article the quality attributes of a unique categorization, loose coupling, discoverability, and autonomy were considered. Other quality attributes and potential quality indicators were informally introduced in [1]. The usage of SoaML as language to model service candidates and service designs enables the integration of the metrics into existing development tools. SoaML represents an emerging standard for modeling service-oriented architectures. Its availability as XMI [45] enables the usage in any UML-capable development tool.

To illustrate the metrics, service candidates and service designs of a service-oriented campus guide system as it is developed at the Karlsruhe Institute of Technology (KIT), called KITCampusGuide, have been introduced. Well-chosen excerpts of service candidates and service designs for this scenario were used to apply the metrics and thus to show their validity. The metrics were applied in practice to design services for the KITCampusGuide with a unique categorization, loose coupling, discoverability, and autonomy. Currently, the metrics also applied for the domain campus management in order to create a catalog of services for universities and their administrative processes according to the Bologna Process [36] with same quality attributes. In this context especially the compliance with naming conventions and the usage of a common domain model for a functional naming are of interest. Additionally, the metrics are applied at the Personalized Environmental Service Configuration and Delivery Orchestration (PESCaDO) project [40, 41], a project co-funded by the European Commission. Also in this case, service designs are supposed to be created that verifiably fulfill the four introduced quality attributes.

Additionally to the identification and formalization of metrics, we work on integrating the metrics into development tools in order to further support the IT architect during the design phase. A more detailed formalization based on the Object Constraint Language (OCL) [44], as already demonstrated in [1], enables the embedding of well-chosen metrics into UML tools and thus an automatic evaluation of service designs. For this purpose, additional semantic information may be necessary. Hence, we are also working on a determination and formalization of this additional information. Furthermore, we work on an integration of the metrics into existing development processes. The metrics are supposed to support the IT architect in creating service designs with certain quality attributes by identifying service

designs flaws [2]. If service design flaws could be determined, appropriate action alternatives that may result in improved service designs are provided to the IT architect. These action alternatives help the IT architect to revise the service designs with regard to certain quality attributes.

REFERENCES

[1] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck, "Evaluation of service designs based on soaml", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 7-13.

[2] M. Gebhart, M. Baumgartner, and S. Abeck, "Supporting service design decisions", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 76-81.

[3] T. Erl, SOA – Principles of Service Design, Prentice Hall, 2008. ISBN 978-0-13-234482-1.

[4] T. Erl, SOA – Design Patterns, Prentice Hall, 2008. ISBN 978-0-13-613516-6.

[5] T. Erl, Service-Oriented Architecture – Concepts, Technology, and Design, Pearson Education, 2006. ISBN 0-13-185858-0.

[6] R. Reussner, W. Hasselbring, Handbuch der Software-Architektur, dpunkt.verlag, 2006. ISBN 978-3898643726.

[7] N. Josuttis, SOA in der Praxis – System-Design für verteilte Geschäftsprozesse, dpunkt.verlag, 2008. ISBN 978-3898644761.

[8] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J.-P. Richter, M. Voß, and J. Willkomm, Quasar Enteprise, dpunkt.verlag, 2008. ISBN 978-3-89864-506-5.

[9] B. Humm, O. Juwig, "Eine normalform für services", GI Informatik 2006, Dresden, Germany, October 2006, pp. 99-110.

[10] S. Cohen, "Ontology and taxonomy of services in a service-oriented architecture", Microsoft Architecture Journal, 2007.

[11] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, T. Winterberg, „Lose kopplung – warum das loslassen verbindet", SOA-Spezial, Software & Support Verlag, 2009.

[12] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, T. Winterberg, „Die soa-service-kategorienmatrix", SOA-Spezial, Software & Support Verlag, 2009.

[13] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, T. Winterberg, „Was macht einen guten public service aus?", SOA-Spezial, Software & Support Verlag, 2009.

[14] M. Perepletchikov, C. Ryan, K. Frampton, and H. Schmidt, "Formalising service-oriented design", Journal of Software, Volume 3, February 2008.

[15] M. Perepletchikov, C. Ryan, K. Frampton, and H. Schmidt, "Cohesion metrics for predicting maintainability of service-oriented software", Seventh International Conference on Quality Software (QSIC 2007), 2007.

[16] M. Perepletchikov, C. Ryan, K. Frampton, Z. Tari, "Coupling metrics for predicting maintainability in service-Oriented design", Australian Software Engineering Conference (ASWEC 2007), 2007.

[17] D. Rud, S. Mencke, A. Schmietendorf, R. R. Dumke, „Granularitätsmetriken für serviceorientierte architekturen, MetriKon, 2007.

[18] M. Hirzalla, J. Cleland-Huang, A. Arsanjani, "A metrics suite for evaluating flexibility and complexity in service oriented architecture", ICSOC 2008, 2008.

[19] S. W. Choi, S.D. Kimi, "A quality model for evaluating reusability of services in soa", 10th IEEE Conference on E-Commerce Technology and the Fifth Conference on Enterprise Computing, E-Commerce and E-Services, 2008.

[20] OMG, "Service oriented architecture modeling language (SoaML) – specification for the uml profile and metamodel for services (UPMS)", Version 1.0 Beta 1, 2009.

[21] IBM, "RUP for service-oriented modeling and architecture", IBM Developer Works, http://www.ibm.com/developerworks/rational/downloads/06/rmc_soma/, 2006. [accessed: January 04, 2011]

[22] U. Wahli, L. Ackerman, A. Di Bari, G. Hodgkinson, A. Kesterton, L. Olson, and B. Portier, "Building soa solutions using the rational sdp", IBM Redbook, 2007.

[23] A. Arsanjani, "Service-oriented modeling and architecture – how to identify, specify, and realize services for your soa", IBM Developer Works, http://www.ibm.com/developerworks/library/ws-soa-design1, 2004. [accessed: January 04, 2011]

[24] P. Kroll and P. Kruchten, The Rational Unified Process Made Easy, a Practitioner's Guide to the RUP, Addison-Wesley, 2003.

[25] M. Gebhart and S. Abeck, "Rule-based service modeling", The Fourth International Conference on Software Engineering Advances (ICSEA 2009), Porto, Portugal, September 2009, pp. 271-276.

[26] P. Hoyer, M. Gebhart, I. Pansa, S. Link, A. Dikanski, and S. Abeck, "A model-driven development approach for service-oriented integration scenarios", 2009.

[27] T. Erl, Web Service Contract Design & Versioning for SOA, Prentice Hall, 2008. ISBN 978-0-13-613517-3.

[28] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA – Service-Oriented Architecture Best Practices, 2005. ISBN 0-13-146575-9.

[29] B. Boehm, Characteristics of Software Quality, Elsevier Science Ltd, 1978. ISBN 978-0444851055.

[30] J. McCall, P. Richards, and G. Walters, "Factors in software quality – volume 1", 1977.

[31] ISO/IEC, "ISO/IEC 9126-1:2001 software engineering: product quality – quality model", 2001.

[32] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software-engineering measurement", IEEE Transactions on Software Engineering, Vol. 22, No. 1, 1996.

[33] S. Johnston, "UML 2.0 profile for software services", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005. [accessed: January 04, 2011]

[34] J. Amsden, "Modeling with soaml, the service-oriented architecture modeling language – part 1 – service identification", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html, 2010. [accessed: January 04, 2011]

[35] W3C, "OWL 2 web ontology language (OWL)", W3C Recommendation, 2009.

[36] European Commission, "The bologna process - towards the european higher education area", http://ec.europa.eu/education/higher-education/doc1290_en.htm, 2010. [accessed: January 04, 2011]

[37] M. Gebhart, J. Moßgraber, T. Usländer, and S. Abeck, „SoaML-basierter entwurf eines dienstorientierten beobachtungssystems", GI Informatik 2010, Leipzig, Germany, October 2010, pp. 360-367.

[38] A. Bauer, S. Eckel, T. Emter, A. Laubenheimer, E. Monari, J. Moßgraber, and F. Reinert, "N.E.S.T. – network enabled surveillance and tracking", Future Security 3rd Security Research Conference Karlsruhe, 2008.

[39] J. Moßgraber, F. Reinert, and H. Vagts, "An architecture for a task-oriented surveillance system", 2009.

[40] The PESCaDO Consortium, "Service-based infrastructure for user-oriented environmental information delivery", EnviroInfo, 2010.

[41] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, "D8.3 Specification of the pescado architecture", Version 1.0, 2010.

[42] M. Horridge, "A practical guide to building owl ontologies using protégé 4 and co-ode tools", http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/, Version 1.2, 2009. [accessed: January 04, 2011]

[43] OMG, "Unified modeling language (UML), superstructure", Version 2.2, 2009.

[44] OMG, "Object constraint language (OCL)", Version 2.2, 2010.

[45] OMG, "XML metadata interchange (XMI) specification", Version 2.0, 2003.

Contextual Injection of Quality Measures into Software Engineering Processes

Gregor Grambow and Roy Oberhauser
Computer Science Dept.
Aalen University, Germany
{gregor.grambow, roy.oberhauser}@htw-aalen.de

Manfred Reichert
Institute for Databases and Information Systems
Ulm University, Germany
manfred.reichert@uni-ulm.de

***Abstract*—Despite improvements in software engineering processes and tools, concrete preventative and analytical software quality assurance activities are still typically manually triggered and determined, resulting in missed or untimely quality opportunities and increased project overhead. Quality goals, when defined, lack holistic environmental support for automated performance measurement and governance that is tightly integrated in the low-level operational software engineering processes. This results in higher quality risks and cost risks. Based on adaptive process management, an approach is presented that injects situationally-determined quality measure proposals into the concrete workflows of software engineers, using contextual semantic knowledge and multi-agent quality goal tracking and decision-making. Our evaluation shows the feasibility of the approach for automatically providing timely quality measure guidance to software engineers without disrupting their current activities. This supports process governance while reducing quality risks and costs during software development projects.**

Keywords-software quality assurance; process-centered software engineering; adaptive process management; semantic technology; agents; Goal-Question-Metric technique

I. INTRODUCTION

This article extends our previous work in [1], which described aspects of an approach for automated integration of software quality management and software engineering process management. Today IT-supported business process management (BPM) enjoys wide industrial adoption [2] and can support improved product quality by ensuring that quality-supporting processes are executed [3]. Process repetition and predictability lowers the recovery risk for necessary investments in process modeling, process management system support, and enterprise application integration [5]. Interestingly, BPM is also increasingly being used for product development [4].

In the software engineering (SE) domain, numerous obstacles inhibit automated SE process management (SEPM) at the operational level. These include the contextual dependency of the low-level activities (e.g., coding, debugging, testing), the high degree of change to the involved artifacts (e.g., source code files, test documentations), the informational and environmental dependencies (e.g., coordination, requirements, reports, tools), the uniqueness of each developer's concrete personal process (e.g., junior vs. senior engineers, information needs), activity coordination with the overall team process, the contextual and project influences on the processes (e.g., schedule, resource availability), and software quality assurance (SQA) dependencies (e.g., quality plan, reactive quality measures, metric dependencies).

Historically software development projects have also faced difficulties in meeting budget, schedule, functionality, and quality targets [6][7][8]. A more recent study in 2002 by the National Institute of Standards and Technology (NIST) found that most delivered software products are still stricken by bugs and defects [9]. While some of these difficulties might be ascribed to a misaligned planning environment in certain organizations [10], the project pressures and resulting issues will likely linger due to global competition and other influences [11]. Other difficulties can be attributed to SE's adolescence as a discipline and certain unique product properties that affect the SE development process, such as software's complexity, conformity, changeability, and invisibility [6]. Additionally, the extent (too little or too much) and timeliness of SQA significantly impacts overall project costs [12], making effective and efficient SQA vital. Yet it remains laborious to manage and apply the appropriate low-level SQA measures (actions) in a timely fashion during SE process enactment. In order to achieve software quality goals, these must be defined and concretely and contemporaneously measured [13]; yet this is often challenging for various SE organizations [14]. Especially small and medium sized companies often struggle to achieve high quality levels. This often results from the increased complexity of their growing organizational structures, the lack of process maturity and capabilities, and the lack of dedicated quality management personnel.

A. Problem Statement

While SE process models foster development efficiency [15], they are often defined rather abstractly and thus fail to provide low-level guidance for the activities actually executed at the operational level. Furthermore, processes are often defined rigidly beforehand. However, during their execution, reality often diverges from the planned process [16].

Automated guidance for combining SQA with SEPM is not yet prevalent. Challenges in software development projects are presented at both the product and process levels based on the nature of software artifacts and manually driven processes. Product intangibility hinders effective retrieval of

timely information about its product quality status. Additionally, the combination of abstract process definitions and intertwined, contextually information-dependent lower-level workflows make targeted process guidance for developers irrelevant, complex, or financially infeasible. Thus, artifact issues often cannot be detected promptly and, even if they are, the contemporary integration of quality measures into the workflow is not possible. At times, quality measures come into focus and are applied close to release, or, when the project is behind schedule, they may be jettisoned altogether; however, it is generally acknowledged that their application in earlier development stages saves time as well as money [12][17]. The proper application of quality measures is also problematic, since their effectiveness and efficiency depend on many factors, such as the applicability of the measure, the project timing, worker competency, and correct fulfillment [12]. For clarity, measure in this article is meant in the sense of a specific action intended to produce some effect - reactive actions are thus countermeasures.

To illustrate the problem as well as the proposed solutions, this paper uses a series of simplified practical scenarios dealing with a fictional company (called 'the Company').

Example 1 (Abstract Process): *The Company uses a SE process model for software development, the V-Model XT [18]. This process features detailed descriptions of activities, roles, milestones, and artifacts. Its application is based on the use of various documents with no automated governance or support. In the Company, activities for developers are planned and scheduled on a very coarse-grained level, leaving the coordination of what is to be done to the developers and manager. Therefore, the SE process does not really "touch" the developers, and their actual activities are difficult to trace. The quality of the source code is not monitored continuously and static code analysis tools are only used sparsely by the developers. Deterioration of the source code quality goes undetected and quality measures are only taken at the end of projects when there is time left or when concrete bugs exist.*

B. Contribution

Automated support for and governance of the coordinated integration of SQA in SEPM offers promising perspectives for addressing shortcomings in current SQA approaches. In the following, the terms *process* and *workflow* will be used extensively, and are delimited here against each other in alignment with existing definitions provided by the Workflow Management Coalition [19] and Gartner Research [20].

Definition: Business Process Management deals with the explicit identification, implementation, and governance of processes as well as their improvement and documentation. This incorporates different issues such as organizational or business aspects, or the strategic alignment of the activities. Workflow Management, in turn, deals with the automation of business processes. Hence, a workflow is the technical implementation of a business process or a part of it.

In our previous work, we introduced the CoSEEEK framework [21], which utilizes various technologies to provide automated, context-aware assistance for SEPM. In [22] and [23], we provided a solution to dynamically generate workflows according to the properties of various situations to support dynamic workflows extraneous to the SE process. We are currently also working on the integration of this dynamic generation with workflows belonging to the SE process and covering situations where pre-planned workflows are too rigid. In [24], we introduced an SE workflow language that provides extended modeling capabilities for SE workflows and improves the connection between abstractly defined processes and concretely executed activities. Finally, in [1] and [25], which provide the basis for this article, we described aspects of our overall approach for integrating SQM and SEPM. In particular, this paper provides a more comprehensive description of this approach with further extensions, in particular elucidating the following areas:

- automatic detection and management of source code related problems in a SE project,
- automatic assignment of quality measures to detected quality problems,
- automatic strategic prioritization and alignment of quality measures to project quality goals,
- tailoring of measure (action) proposals to the situation, and
- automatic integration of quality measures in the software engineer's workflow.

The remainder of this paper is organized as follows: Section II presents background information needed for the understanding of this paper and elicits fundamental requirements for our solution approach. Section III describes our solution approach. Section IV discusses realization aspects and Section V evaluates our solution. Finally, Section VI discusses related work and Section VII concludes the paper.

II. REQUIREMENTS

Requirements have been elicited based on various sources we found in literature. The identified requirements cover different areas to enable comprehensive system support for integrating SQA and SEPM.

Context-awareness. To enable automated decisions on quality measure assignments, any system support should be aware of its environment and the context of the current situation.

Requirement R:Ctx1 (Context integration): To be aware of problems in the SE project, the system must have a facility to integrate information on SE process or product problems from various sources (e.g., external tools measuring the state of the source code, bug tracking systems).

Requirement R:Ctx2 (Quality opportunity awareness): To enable automated integration of quality measures at run-time, the system must be aware of quality opportunities, meaning time points when a user can cope with a quality

measure. This requires knowledge about the users' schedule, meaning the abstract activities that have been scheduled and estimated for the user.

Process management: To enable the automated integration of quality measures, the system must be able to govern the SE process automatically. To foster this in SE, facilities must be in place to enable the system to match the workflow specification belonging to the process (or parts of it) with other facts representing the current situation. That way, contextual information can be used for SE process support.

Requirement R:Sepm1 (Vertical workflow connection): It should be possible to define flexible connections between different workflows (meaning the vertical connections between sub-workflows and their super-workflows). In traditional process management, a simple connection between sub- and super-workflow is possible. If an activity of a super-workflow refers to a sub-workflow, it will be only finished at run-time after completing the corresponding sub-workflow instance. However, in practice, more complex connections may be required as illustrated in Figure 1 and confirmed by processes from other domains like the automotive industry [26] and enterprise resource planning [27]. In this example (Figure 1), activities are grouped by work packages. In the planning phase, for example, the packages and their corresponding activities are planned. This means that the activity of planning a package depends on the completion of the planning of the contained activities. The same applies for the processing of a package. That way there are multiple connections between the super and the sub-workflow, and the completion of a certain activity does not necessarily depend on the completion of a whole sub-workflow, but on the completion of one or multiple activities in one or multiple other workflows.

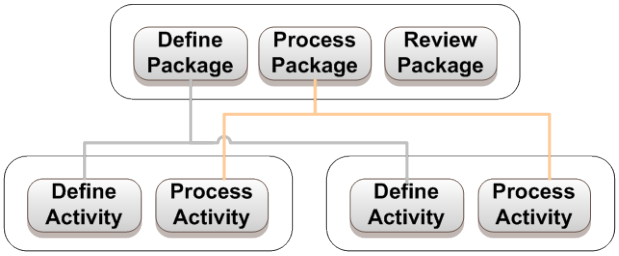


Figure 1. Sub-workflow Connections.

Requirement R:Sepm2 (Task Granularities): For the automated detection of quality opportunities, an explicit connection between abstract assignments, which have been planned and scheduled, and the related concretely executed activities is desirable. Traditional BPM features human tasks only with one single granularity. In fact, human tasks exist on different levels of abstraction that are often related to each other (see also [28][29]). Process management lacks sufficient support for this - just modeling tasks on different levels of abstraction does not adequately match reality since tasks usually have different properties. An example of those tasks is shown in Figure 2.

Example 2 (Task Granularities): *Task 1 is an abstract assignment, which was planned and estimated from the business side. That task implies Tasks 2 and 3, which are concretely planned and executed by the developer. Those tasks, in turn, also imply tasks on a more concrete level like Tasks 4, 5 and 6, which may have special connections to the environment, as they require, for example, certain tools.*

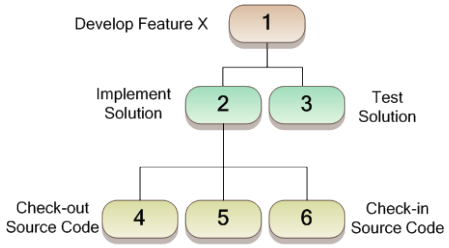


Figure 2. Task Granularities.

Requirement R:Sepm3 (Workflow adaptation): The concrete workflows should be adaptable. In particular, their specification should support automated adaptations during run-time to enable the system to automatically and dynamically insert quality activities into workflows where required and favorable.

Quality Measure Selection. The selection of appropriate quality measures during SE process execution constitutes another challenge. Various factors must be taken into account for effectiveness and efficiency.

Requirement R:Qmsel1 (Quality measure selection): Applied quality measures should be automatically chosen during run-time in alignment to project goals in order to match the defined strategy of the project.

Requirement R:Qmsel2 (Proactive measures): Quality measures should not only rely on detected problems, but also consider common quality enhancement. Thus, proactive and reactive measures should be available.

Requirement R:Qmsel3 (Situational measure tailoring): Context-sensitive tailoring of proposed measures is desirable considering different factors of the actual situation, e.g., properties of the applying person and application time point.

Requirement R:Qmsel4 (Measure assessment): The selection of measures should be aware of their effectiveness to optimally match with specific environments or situations in different companies. Therefore, continuous monitoring of the quality of the source code is essential to detect potential impacts of applied measures on the overall quality. In particular, a relation between the application of SQA measures and the evolution of source code quality should be established to assess the effectiveness of the measures.

III. SOLUTION APPROACH

Considering the aforementioned requirements, the concepts behind our solution approach are now described in detail. To automatically integrate quality measures into the SE process, our approach consists of a process (referred to here as a procedure to avoid confusion with SEPM) in conjunction with the architecture of the *Context-aware Software*

Engineering Environment Event-driven frameworK (CoSEEEK) [21].

A. Solution Procedure

Our solution procedure involves three fundamental phases: a *detection phase*, a *processing phase*, and a *post-processing phase* as shown in Figure 3. These phases as well as their different steps will be explained in the following subsections.

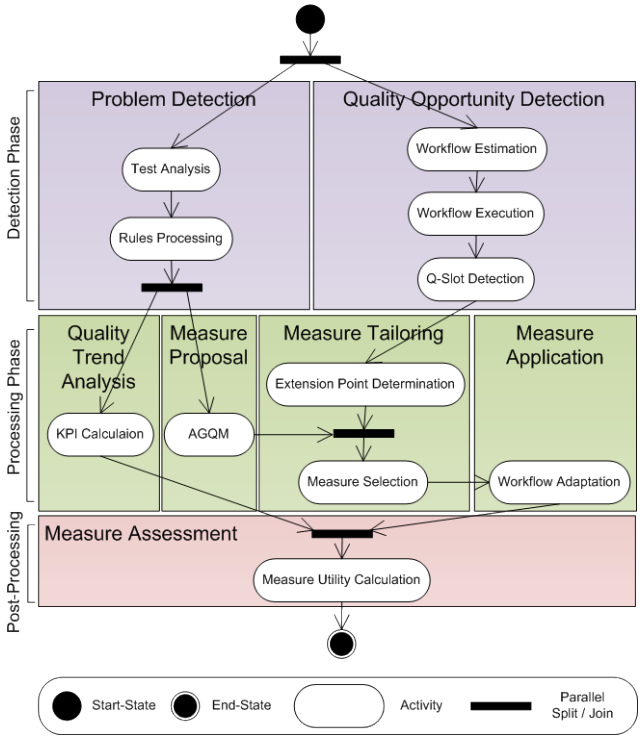


Figure 3. Conceptual Procedure.

The *Detection Phase* continuously enables an awareness of the current project situation to meet the requirements relating to context-awareness (cf. Section II.B). For integrating quality measures, two factors are of particular interest: the presence of problems (cf. Requirement R:Ctx1) - recognized via the 'Problem Detection' - and the availability of opportunities for quality measures in the users' schedule (cf. Requirement R:Ctx2) - recognized via 'Quality Opportunity Detection'. To enable such detection in an automated fashion, the SE process specification must be extended (cf. Requirement R:Sepm2). The applied extensions will be described in Section III.C.

The *Processing Phase* deals with the selection and proposal of the quality measures and involves four steps. Utilizing the GQM technique [30], quality measures (actions) are initially proposed in alignment with project goals to satisfy Requirement R:Qmsel1. This phase also adds proactive measures to the measure proposal process (cf. Requirement R:Qmsel2). To prepare these measures for their automated application, 'Measure Tailoring' incorporates information about the applying persons and the possible points in the users' schedule in which to apply the measure (cf. Requirement R:Qmsel3). This leads to a selection of appropriate points (so called *Q-Slots*) and to an automated integration of the quality measures into the concrete workflow of the chosen person. The application of measures can be also done automatically utilizing extensions made to the SE process specifications (cf. Requirement R:Sepm3). These enable the system to be aware of matching extension points (e.g., in the workflows). These are illustrated in Figure 4 by a small abstract workflow containing the activities 'A1' to 'A5'. 'A2' and 'A4' have an associated extension point, meaning that an automated insertion of a new activity is possible subsequent to these activities.

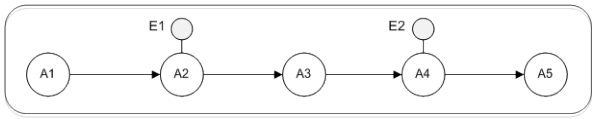


Figure 4. Extension Points.

Finally, to be able to track the quality of the project continuously, in the *Post-Processing Phase* (cf. Figure 3) a 'Measure Assessment' is performed via a quality trend analysis. This analysis supports an awareness and automatic assessment of the potential utility of the applied measures, fostering quality (cf. Requirement R:Qmsel4).

Since each project is unique, the applicability and effectiveness of measures can vary with respect to different projects. Therefore, the system executes an assessment phase to rate the applied measures and to incorporate their impact in the given project.

B. Conceptual Architecture

CoSEEEK provides the necessary infrastructure for realizing the solution procedure presented in the previous sub-section. Its conceptual architecture is shown in Figure 5.

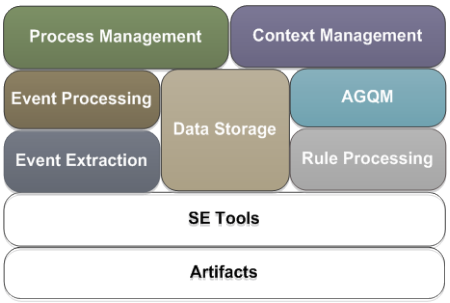


Figure 5. CoSEEEK Conceptual Architecture.

SE Tools is a placeholder for all tools used in a SE project of which CoSEEEK is aware, such as source control systems or IDEs. Artifacts are those things produced in a SE project using the *SE Tools*. This includes source code artifacts, documents, and models.

Awareness of changes to the state of tools as well as artifacts is supported by the *Event Extraction* module. It utilizes sensors that are typically integrated into the tools or otherwise monitor the tools. These sensors generate events in in response to various situations (e.g., 'switch to debug

perspective' in an IDE). The *Data Storage* module encapsulates the storage mechanisms for events and shared data. Communication is event-based and loosely coupled to support integration and exchangeability of different modules.

The events generated and collected in the *Event Extraction* module are basic and low-level. The *Event Processing* module utilizes *complex event processing* (CEP) [31] to process these events, providing high-level events with enriched semantic value. The *Rules Processing* module uses rule-based computing to analyze tool data such as static analysis reports or metrics, and it triggers follow-up actions as necessary (e.g., quality measures for violated metrics). These triggered measures are subsequently filtered by the *AGQM* (Automated Goal Question Metric) module, which automates and extends the GQM approach via multi-agent computing to analyze the project quality state in alignment with strategic project goals and to propose appropriate proactive and reactive quality measures.

The *Process Management* module applies dynamic and adaptive process management technology to govern the activities of the involved project participants. This enables the system to match workflows to real project situations (instead of rigidly prescribing certain activities and their orders) and thus provides real situational guidance. This becomes possible by utilizing the cumulated knowledge contained in the *Context Management* module. In that module, high-level information of all project areas is collected, as, for example, the skills of users or information about the quality state of the project. Using semantic technology, this information can be used to reason about the project state and contextual influences (causes and effects) and thus provide automated decisions and workflow adaptations such as the automated and dynamic integration of quality measure activities during SE process execution.

C. Context-aware Business Process Management

To support a high degree of automated and context-aware assistance, a tight coupling of the *Context Management* and the *Process Management* module is required, which will be referred to as *Context-aware Process Management* (CPM). This addresses many of the shortcomings of traditional BPM (as listed in Requirements *R:Sepm1 to R:Sepm3*) and facilitates the comprehensive utilization of the awareness capabilities in CoSEEEK. Fundamentally, process management concepts are enhanced with semantic information. This additional information is stored in the *Context Management* module, while the workflows are managed by the *Process Management* module. Since *Context Management* unifies all project knowledge, it can be also used as a management layer around the *Process Management* module, facilitating context-based process management. Thus, all process-related actions are addressed by the *Context Management* module, which, in turn, manages the actions of the *Process Management* module. Figure 6 illustrates these extensions to process management.

The *Process Management* module governs the workflows and their activities. These two concepts are mirrored in the *Context Management* module: the activity by the *Work Unit* and the workflow by the *Work Unit Container*. Thus, process management is separated into two areas that we call *vertical* and *horizontal process management*. *Horizontal process management* denotes the governing of the different activities of one workflow (also denoted as process orchestration) utilizing well-established workflow patterns like AND, SPLIT, or LOOP. This is done within the *Process Management* module. *Vertical process management,* in turn, deals with the management of the dependencies between different workflows on different levels of abstraction. Since process management only offers one kind of connection (an activity depends on a sub-workflow) here, this is handled by the *Context Management* module. This allows for the flexible definition of dependencies. The completion of a *Work Unit* can depend upon one or multiple *Work Unit Containers* or on one or multiple *Work Units* contained in other *Work Unit Containers*. A mixture of both is possible as well.

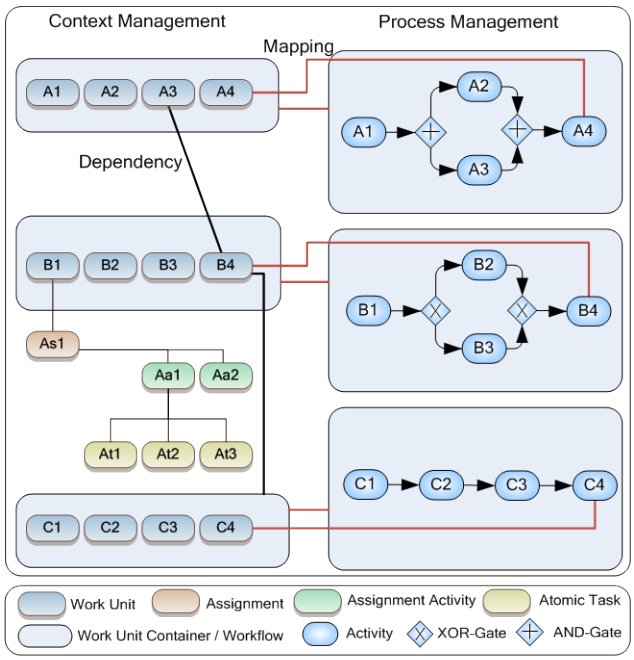


Figure 6. Context-aware Business Process Management.

Work Units are connected to three other concepts, enabling advanced task management (cf. Requirement R:Sepm1). The *Assignment* is used as a coarse-grained top-level task, which is also estimated and scheduled from the business side in a project, exemplified in Figure 2 as "Develop feature X". The *Assignment Activity* then describes the tasks that are necessary to accomplish the *Assignment*, e.g., "Design Solution" or "Write Developer Tests" (cf. Figure 2). The most fine-grained level is described by atomic tasks like "Check out" or "Build Code".

Combining the *Context Management* and the *Process Management* modules enables the automatic adaptation of running workflows based on the current context. This has been used to automatically build workflows for issues extraneous to SE process models, like bug fixing or refactoring as described in [22].

D. Quality Opportunity Detection

To enable the automated detection of quality opportunities, an awareness of the activities that have been planned and scheduled becomes necessary. These are captured by the *Assignment*, which has certain properties to capture estimated durations. These assignments can be created, estimated, and scheduled in CoSEEEK or imported from other tools. This is illustrated by Figure 7.

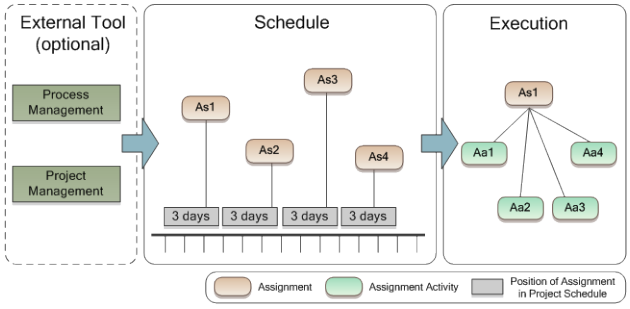


Figure 7. Quality Opportunity Detection.

The figure shows a simple example schedule containing the Assignments 'As1' – 'As4' that are estimated to take three days each. The scheduled Assignments are then taken for execution. The figure illustrates the connection of the Assignment 'As1' to four Assignment Activities for execution, which are 'Aa1' – 'Aa4'. Optionally, the schedule can be imported from an external tool. That way, the activities can be estimated and scheduled from the business side, e.g., utilizing a process management tool like microTOOL in-Step [32], and then be automatically used for execution in CoSEEEK.

In our approach, two triggers for a quality opportunity are implemented. The first one is early *assignment* completion. If a user finishes an assignment earlier than necessary, a quality measure can be assigned to him without delaying forthcoming activities. The second trigger is the quality overhead factor. It enables the *a-priori* specification of a certain percentage of the project workload that should be reserved for quality activities. If the user has not yet reached that amount during process execution, a quality measure may be applied. This can be combined with a quality function indicating how much time for quality should be spent in which project phase. Since it has been shown that it can be beneficial to adjust the work allocation for quality based on the stage of a project [12][17], this can improve both the effectiveness and efficiency of the quality efforts taken.

Example 3 (Quality Opportunity Detection): *To illustrate how automated quality opportunity detection could work, Figure 8 shows a simple schedule of the Company. This example, which demonstrates early assignment completion, assumes that the Company has already adapted the planning to create more fine-grained assignments not taking weeks but days. The schedule comprises four users having five assignments each. Each of these assignments, in turn, is estimated to take three days. Every user in this example finishes early on one Assignment, triggering the creation of a Q-Slot filling the hole in the schedule.*

Figure 8. Schedule with Q-Slots.

The concrete detection of the quality opportunities is done each time an assignment completes. This can be detected automatically by connecting the *Assignments* to the users' *Atomic Tasks* (cf. Section III.C). *Atomic Tasks,* in turn, are connected to the development environment via the sensor infrastructure provided by CoSEEEK, generating an awareness of their status. When all *Atomic Tasks* of an *Assignment Activity* are completed, the *Assignment Activity* completes as well. The same applies to the *Assignments* in relation to the *Assignment Activities*. This process is further described in [25].

E. Problem Detection

Problem detection makes use of the environmental awareness of CoSEEEK to identify potential problems, e.g., in the source code. In this context, external data from tools needs to be integrated. This information is utilized for calculating various metrics that can be customized to measure the quality state of the SE project. Metrics directly indicating problems in the source code are obtained from static code analysis tools like PMD [33] and FindBugs [34], while certain testing problems can be detected with test coverage tools [35] such as Cobertura [36] or EMMA [37].

However, not only code-related product-level problems threaten quality, but process-related factors should also be assessed to ensure quality. These assessments include functional testing, profiling, and load testing. Since CoSEEEK is aware of the execution of respective activities, it can ascertain their absence. Thus, process metrics can also include these facts. Facts available to CoSEEEK can be incorporated in metrics, which enables quality awareness through the presence of measurable, quantifiable information. To reduce the associated configuration effort, a set of pre-configured default metrics will be included with the system.

After detecting any problem, measures (actions) can be used to counter them. The *Rules Processing* module is utilized for triggering the automatic proposal of a measure when the threshold for a particular metric has been violated. Metric or violation reports are received and analyzed, and a list of the violated metrics including assigned quality measures is created by the module.

Example 4 (Source Code Monitoring): *Company policy includes a nightly built process on a build server that invokes static code analysis tools to enable continuous measurement. Thus, a deterioration of the source code quality can be detected by metrics such as its cyclomatic complexity. If complexity exceeds a threshold, the code becomes difficult to maintain and test, and there is a higher probability of introducing defects.*

F. Measure proposal

At this point in the procedure, problems as well as Q-Slots have been detected and an initial assignment of quality measures to metric violations has been done. The generation of a Q-Slot then triggers a measure selection and proposal process. The latter is coordinated by the *Context Management* module. First, the *Context Management* module triggers the *AGQM* module to prioritize the measures strategically in alignment to the quality goals of the SE project. Thereafter, the measures are tailored to the current situation.

The process of prioritizing measures is very dynamic due to different goals, presence of various metrics and violations and different project situations. Therefore, the *AGQM* module features a multi agent system (MAS) to prioritize measures in alignment to the project goals to be able to accommodate these various factors. The process is based on an extension to the GQM technique [30]. GQM consists of a hierarchical structure that starts with the definition of certain project goals. During configuration, for each goal various questions are defined. These answers should provide indicators for the level of goal fulfillment. Each question can be associated with certain metrics, establishing a connection from the abstract project goals to concrete facts in the project. The following example shows the application of the GQM technique.

Example 5 (GQM Plan): *As part of a GQM plan, a goal 'Maintainability' could be defined relating to the maintainability of the produced source code. For this goal, one question could be 'How understandable is the code?' For this question, in turn, different metrics could apply. One example is the metric 'Comment Ratio'.*

1) GQM Extensions

This subsection shows the basic concept on which the agents rely. Two main requirements have to be satisfied to facilitate automatic support for GQM execution. First, a GQM plan must exist that defines the relations between goals, questions, and metrics. Second, the metrics have to be integrated in the system, enabling the automatic extraction of corresponding values and thus the automatic receipt of possible deviation information.

Some extensions to the GQM technique became necessary to support automation. Different abstractions of key performance indicators (KPIs) were introduced to enable the automatic calculation of goal deviations. Furthermore, metrics are encapsulated in KPIs to enable consolidation and simplified deviation calculation. Since multiple metrics may be utilized for a single question in GQM, a QKPI (Question KPI) was created for consolidation of the metric values at the question level. Similarly, multiple questions may apply to a single goal, thus a GKPI (Goal KPI) is used for goal deviation calculations. For each of the KPIs, formulas specify how metrics are combined. To support automated multiple goal attainment, each defined goal was assigned one agent responsible for monitoring and fulfillment of that goal.

The calculation of the different KPIs is conducted by the Context Management module as part of the quality trend analysis described in Section I. Figure 9 shows the relation between the different conceptual elements.

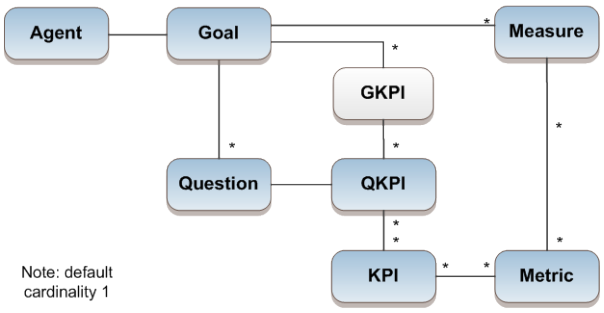


Figure 9. Extended GQM Structure

To prescribe appropriate countermeasures for (potential) quality deterioration, measures were categorized as follows: *reactive (or analytical) measures*, which are directly associated with concrete metrics or violations, and *proactive (or preventative) measures*, which hereby are categorized as supporting certain quality goals at an abstract level and may not be readily associated with a concrete problem. Proactive measures are assigned to GKPIs and can be triggered either when a GKPI deviation occurs (a supportive role) or in the absence of reactive measures. This differentiation is pragmatic since reactive measures can be based on concrete existing problems and can thus be more fine-grained, whereas proactive measures support a goal in general.

2) AGQM Process

At the beginning of a project or a phase or iteration, a quality manager assigns points to each goal (implying its importance) and chooses a bidding strategy for the agent managing that goal. These points are used by agents for negotiating proposed measures. The AGQM process invokes a proactive as well as reactive selection mechanism that results in a measure proposal.

Quality goals can be conflicting, and determining the appropriate balance is project-specific. Thus, a competitive bidding process among agents is chosen for enabling proactive measures, whereas a cooperative voting process is applied for enabling reactive measures. The competitive bidding allows agents with greater importance to definitively have opportunities to support their goal with measures, in contrast to voting where agent majorities might win. That way, a group of lower-priority goal agents does not hinder a higher priority goal from ever asserting influence. The bidding strategies enable agents to win opportunities earlier or later in an iteration cycle.

The reactive voting process is cooperative since a potentially large number of concrete reactive measures based on metric violations become possible for a limited number of quality opportunity slots (Q-slots), and those measures that will have the greatest overall quality impact across all goals are favored. The agents cooperatively vote on the measures list received from the *Rule Processing* module. Via the structure shown in Figure 9, each agent determines for each measure whether a measure belongs to a metric being related to the agent's goal. The points of an agent are then

distributed (currently uniformly) across all measures associated to its goal. A prioritized list of reactive measures is output, while the ultimate choice will be applied by *Context Management* based on the situation.

The proactive section of the *AGQM* module utilizes metrics for calculating the different KPIs, QKPIs, and GKPIs. If there are any deviations at the goal level of an agent (GKPI), it may participate in the bidding session (this favors those goals known to be at risk). Each agent bids and the highest bid wins, elevating its proactive measure set to a proposal. In this process, not just the points differentiate between the goals, but also the strategy chosen by the agents. The strategies influence how an agent increases or decreases its bids after winning or losing for the next bidding process. Choosing a defensive strategy for an agent will increase the likelihood that a proposal of its associated measures will occur in later phases of the iteration. This behavior occurs because in early sessions the agents with more aggressive strategies will place much higher bids. The defensive agent can then place winning bids later when the aggressive agents run out of points.

To define an appropriate proportion between proactive and reactive measures, a proactive-to-reactive ratio can be defined. This determines how often reactive vs. proactive measure sets are provided by the *AGQM* module. Section V will evaluate a concrete scenario utilizing this approach. If no metrics and thus no reactive measures are yet available, then no question or goal deviation is detectable since there is no basis for their calculation. In this case, all agents participate in proactive bidding so that any Q-Slots can be used for proactive measures.

G. Measure Tailoring

The AGQM module has created a list of prioritized measures according to project goals. However, the final selected measure should depend on environmental factors for the most effective and efficient measure application. These include properties of the measure itself, properties of the applying person, and properties of the current situation.

The properties of the measure are defined in the *Context Management* module and include, for example, the type of the measure or the applicable number of users involved (e.g., a code review involves multiple persons). One property of the person that can be of interest is the skill level. The properties of the current situation are modeled based on the concepts of the *Q-Slot* and the *Extension Point*. The *Extension Point*, as part of the semantic enhancements to the process, is a pre-specified point in the process where the integration of a quality measure is feasible, as illustrated in Figure 4. That involves an abstraction level and applicable measure type since, for example, at the end of a project phase other measures might be applicable compared to a time point after directly implementing new functionality. The *Q-Slot* captures a time category indicating how much time is left for a quality measure. Via these properties, a measure fitting to the current situation can be chosen. This process is further explained in [25].

H. Measure Application

To enable a high degree of automated guidance, the user is not only informed about the measure to be applied, but the measure is also directly integrated into the users' workflow (not necessarily visible to the user, but tracked by the system). Both semantic enhancements to process management and the capabilities of the adaptive process management system, which enables the dynamic change of running workflow instances, are used. Details can be found in [25].

Example 6 (Measure Selection): *To enable automated support for quality measures, the Company introduced the facilities for automated problem and quality opportunity detection. A GQM plan was created with maintainability and reliability as well as the creation of new functionality as goals. If developers now finish early on Assignments, the system can automatically assign them quality measures that fit the project goals and are appropriate to the personal situation. As example for this, consider refactoring of complex code. This measure was triggered as problems in the source code were detected, for example by applying the cyclomatic complexity metric. The measure was prioritized as high since it was judged as important and applicable to both the maintainability and reliability goals of the project. Finally, the system can choose the matching person for the measure based on properties like the skills of the person, their familiarity with that code section, or the amount of time they can spend in accomplishing a measure vs. the expected time needed for the measure.*

I. Quality Trend Analysis

The continuous monitoring of the quality of the source code is essential to be able to detect any impact of applied quality measures. Therefore, the list of quality measures from the *Rule Processing* module is utilized by the *Context Management* module. The list not only contains proposed measures, but also the metric belonging to each measure and the value of the metric. To enable automated evaluation of quality trends in conjunction with the GQM technique, different levels of key performance indicators (KPIs) were introduced as depicted in Figure 9.

KPIs are composite metrics unifying the values of other metrics or KPIs to enhance their expressiveness and significance. Each KPI is based on a formula that prescribes how the values of the encapsulated metrics are used to compute the KPI value. KPIs are utilized not only for quality trend analysis on different levels of abstraction, but also for automated goal deviation monitoring with respect to the GQM technique. Therefore, three levels of KPIs were introduced: on the most concrete level, the KPI unifies one or more metrics for clarity since different metrics may be utilized by the system. The QKPI represents a Question of the GQM technique as a value to facilitate automated deviation calculation, which is automatically computed from attributed KPIs and base metrics. The same applies for the GKPI, which unifies the values of the questions belonging to one project goal.

Compared to our initial approach described in [1] and [25], the calculation of the KPIs has been refined and moved

to the *Context Management* module. Therefore, the KPI structure and the various calculated KPI values are stored in the ontology for better information processing and access. The values can then be incorporated in reasoning procedures and the data can be easily provided to other external applications.

The calculations are now done in a uniform way for all KPIs, applying a weighted average of all values a *KPI* aggregates as depicted in Formula (1), where M_i are the concrete values that are aggregated and W_i are the attributed weights of the n metrics or KPIs being aggregated. All received metric values are normalized to a range from 0 to 1.

$$KPI = \frac{\sum_{i=1}^{i=n} W_i M_i}{\sum_{i=1}^{i=n} W_i} \quad (1)$$

J. Measure Assessment

Regarding different companies with different people, tools, and processes, applied measures may show different degrees of effectiveness. To reflect that and to improve future measure proposals, a so-called measure utility is introduced to indicate the usefulness of the applied measure. That property is neutrally initialized and updated after each application of the measure. The delta of the *KPI* related to the measure right after its application is compared to the value prior. Since some measures may not have an immediate effect, multiple future deltas can be taken into account. This process is further described in [25].

Example 7 (Measure Assessment): *The special refactoring proposed in Example 6 can now be automatically assessed for the Company since it has applied continuous quality measurement. If the refactoring is successful and the complexity of the code is reduced, this is indicated by a subsequent measurement showing a lower value for the cyclomatic complexity metric. This value will then also affect the value of a KPI related to the maintainability goal defined by the Company. The KPI value, in turn, will affect the utility value of the proposed refactoring measure, which will, if successfully applied, have a higher probability of selection by Context Management in the future.*

IV. REALIZATION

This section provides implementation details for the components described in Section III and reflects their current implementation status.

A. Architecture

The technical architecture is depicted in Figure 10, whereby the modules are deployed as web services. To support loose coupling of the deployed services in CoSEEEK, event-driven computing and space-based computing are leveraged for service interaction [21]. The communication with the tuple space is technically realized using the web service framework Apache CXF.

For *SE Tools* and *Artifacts*, the actual instances are dependent on the SE environment and the current configuration. In the context of this paper, *SE Tools* includes static analysis tools like PMD, version control tools like Subversion, and IDEs (Integrated Development Environments) like Eclipse. Other relevant tools are, for example, external project management tools from which processes can be imported, like microTOOL inStep [32].

The primary *Artifacts* relevant to the scenario presented in this paper are source or test code files that are processed using these tools.

The *Event Extraction* module utilizes the Hackystat framework [38]. This framework provides a rich set of sensors that can be integrated into various SE tools. The sensors enable the *Event Extraction* module to automatically generate events in different situations, as, e.g., checking in a source code file in Subversion or switching to the debug perspective in Eclipse.

Most of the extracted events are rather atomic and thus combined by the *Event Processing* module to provide more semantic value. This is done utilizing Esper [39] for CEP. Esper provides a facility to define patterns that govern how certain events are combined to derive other higher-level events, which are then again written to the *Data Storage* module as all other events.

The *Data Storage* module, in turn, is realized via an implementation of the tuple space paradigm [40] on top of the XML database eXist [41] for shared XML data, whereas the Hackystat SensorBase is used for high volume event data. The specific CoSEEEK tuple space implementation that uses eXist consists of multiple so-called collections that structure the stored information. Examples include 'Context Management' as well as 'Process Management'. Each CoSEEEK module can write tuple events in these collections or subscribe to be automatically informed about new events in a certain collection.

The *Rules Processing* module automatically processes static rules to assign certain quality measures to certain violated metrics utilizing JBoss Drools [42].

The *AGQM* module, which is in charge of strategic quality measure prioritization, employs a multi-agent system (MAS) with different behavior agents. It is implemented utilizing the FIPA- compliant [43] Jade framework [44].

The *Context Management* module employs semantic technology to enable high-level utilization of all project knowledge. Technology advantages include enhanced interoperability between different applications, extending reuse possibilities, and the option for advanced content consistency checking [45]. It also provides a vocabulary for the modeled entities including taxonomies and logical statements about the entities. Ontologies also provide the capability of reasoning about the contained data and inferring new facts. As an ontology language, OWL-DL (Web Ontology Language Description Logic) [46] is used due to its proliferation and standardization. For simple RDF [47] based queries to the ontology, SPARQL [48] is used. Operations that are more complex are executed using the reasoner Pellet [49]. Programmatic access via DAO objects to the ontology is provided by the Jena framework [50]. Thus, different semantic concepts can be created and manipulated as needed.

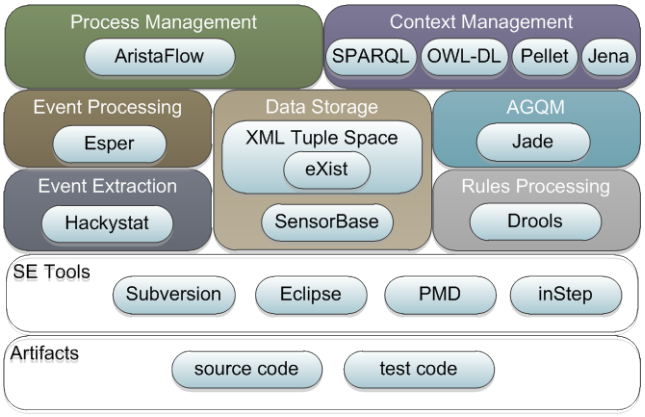


Figure 10. CoSEEEK Technical Architecture

The *Process Management* module builds upon the AristaFlow BPM Suite [51][52]. AristaFlow provides process management technology that is notable with respect to the flexible support of adaptive and dynamic workflows. New workflow templates can be composed out of existing application services in a plug-&-play like fashion, and then serve as schema for the robust and flexible execution of related workflow instances. In particular, during run-time, selected workflow instances can be dynamically and individually adapted in a correct and secure way; e.g., to deal with exceptional situations or evolving business needs [53]. Examples of workflow instance changes supported by AristaFlow include the dynamic insertion, deletion, or movement of single workflow activities or entire workflow fragments respectively (for a discussion of the adaptation patterns supported by AristaFlow, see [54]). For integrating these change functions and other AristaFlow services (e.g., for managing user work lists or for defining workflow templates) with domain- or application-specific process-aware information systems as in our case, the AristaFlow Open Application Program Interface (Open API) can be utilized [55][56]. For example, for dynamically inserting activities at the workflow instance level, the application developer can make use of the following system functions provided by the AristaFlow Open API:

- Querying the activity repository for available activity templates,
- Marking those activities of the workflow instance after which the selected activity shall be inserted (i.e., after completing these activities the newly added one shall be enabled),
- Retrieving the set of activities selectable as “end” activities for this insertion,
- Marking the activity (or set of activities) which shall serve as end activity (activities),
- Performing (tentatively) the insertion based on this information,
- Checking the AristaFlow report on detected errors (e.g., missing values for input parameters), and
- Making the instance change persistent.

Note that dynamic workflow instance changes can be conducted at a high level of abstraction. In particular, all complexity relating to dynamic workflow instance changes (e.g., correct transformations of the workflow schema, correct mapping of activity parameters, state adaptations) are hidden to a large degree from end users and application developers respectively [57]. Furthermore, AristaFlow provides techniques for learning from past experiences and workflow instance adaptations, respectively, and for evolving workflow schemes accordingly [58][59][60].

B. Context-aware Business Process Management

As mentioned in Section III, CPM (Context-aware Process Management) is enabled by correspondingly modeling the workflows in the ontology. Figure 11 illustrates this with the 'Develop Solution Increment' workflow of the OpenUP process [61].

In traditional process management, as mentioned in Requirement R:Sepm2, some aspects of task management have not been sufficiently covered. On the one hand, coarse-grained user assignments that have been planned for the current iteration or phase are not explicitly included since they are too abstract. In CoSEEEK, this is done explicitly in the *Context Management* module, as illustrated in Figure 11, by the Assignment 'Develop Feature X'. The latter is connected to the *Work Unit Container* that, in turn, contains all *Work Units* representing the activities of the workflow. If human tasks shall be executed via those activities, they are implicit parts of these activities.

In CPM, modeling it is done more explicitly. *Work Unit*s representing activities are only used for workflow governance. If they shall imply human activities, they have to be connected to Assignment Activities. The latter are also connected to the Assignment, making the connection of the abstract assignment to the concretely executed activities more explicit. However, activities like 'Implement Solution', in turn, consist of a number of smaller tasks. These tasks are also explicitly modeled in CPM. Figure 11 shows the *Atomic Tasks* related to the 'Implement Solution' *Assignment Activity*. These tasks can be automatically detected by the sensors of CoSEEEK’s *Event Extraction* module. Thus, it is possible that the system is automatically aware of the completion of an activity through the detected completion of all related tasks and thus can automatically finish that activity and propose the next one. This relieves the user from the burden of always explicitly informing the system about activity completion. The same applies to the Assignment, which can be automatically finished by the completion of all related Assignment Activities.

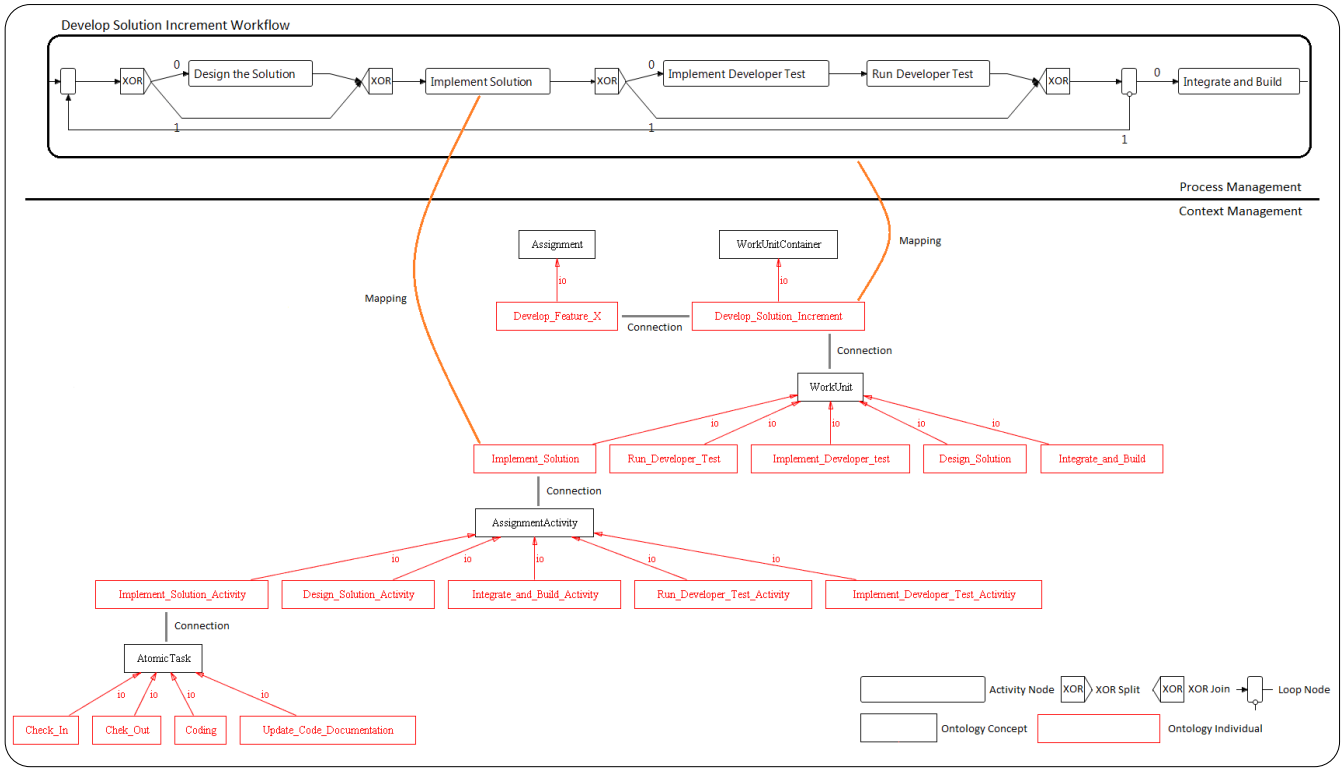


Figure 11. Concepts in Process Management and in the Ontology.

The enrichment of process management concepts via the ontology further enables the aforementioned extended vertical relations between workflows and the clear separation between *horizontal* and *vertical process management* (cf. Section III.C). The separation into two areas governed by two different modules was chosen despite the high coordination effort it imposes. The main reasons for this are as follows: mirroring process management components in the ontology not only enables extended vertical process management, but also allows for a tighter integration of the processes in the projects using different task levels for humans and CoSEEEK's environmental awareness capabilities. Since the decision on whether or not a *Work Unit* and the respective node can be completed is done in the *Context Management* module, the vertical dependencies are integrated into that procedure as well. Thus, multiple dependencies are all managed at one point, fostering contextual integration of process management and dependency extensibility. A *Work Unit* cannot complete, for instance, if related user activities are not completed or if the *Work Unit* depends on activities by other users or teams. An example of new dependencies that can be easily integrated is that an artifact has to be in a certain state or that an external tool has signaled a certain event. The horizontal governing of a process structure stays with process management because this is a non-trivial task and mature process management systems such as AristaFlow (cf. Section IV.A) support many correctness checks and guarantees on process execution. The extensions to *vertical process management* made by CPM are detailed in the following. There exist three possible connections, all of which can occur multiple times and can be mixed:

- *Depends-on-Work Unit Container*: This is the classical connection between an activity and a contained sub-workflow. When the *Work Unit* is activated, the *Sub Work Unit Container* is started and the *Work Unit* must not complete until the *Sub Work Unit Container* is completed. This connection is illustrated in Figure 6 by the *Work Unit* 'B4' that depends on the *Work Unit Container* containing the *Work Units* 'C1' – 'C4'.
- *Depends-on-Work Unit*: In this connection, the completion of the current *Work Unit* does not depend on a *Work Unit Container,* but on the completion of another *Work Unit*. When the depending *Work Unit* is activated and the *Work Unit Container* containing the *Work Unit* on which the current *Work Unit* depends has not been running yet, it is started. This connection is illustrated by the *Work Units* 'A3' and 'B4' in Figure 6.
- *Initiates-Work Unit Container*: This connection is asynchronous. The *Work Unit* does not depend on anything, but when it becomes activated, the connected *Work Unit Container* is started.

C. Procedure

This section gives a short outline about the temporal coordination of different modules. The whole procedure can be decomposed into three processes partly dependent on each other:

- When a report from an analysis tool is received, the tool-specific format is first transformed into a

unified one. Thereafter, the report is processed by the *Rule Processing* module with triggered measures output to *Data Storage* whereby any subscribers are notified. They retrieve and use the data, in this case the reactive section of the *AGQM* module and the KPI calculation of the *Context Management* module.

- When a user finishes an activity, the Q-Slot detection is started within the *Context Management* module. If a Q-Slot is available, the *AGQM* module is triggered by the *Context Management* module to generate an ordered list of proposed measures. This list is used by the tailoring process in *Context Management* to select a measure that is then integrated into a workflow by the *Process Management* module.
- At certain configured time points, the *Context Management* module is triggered to do an assessment of the applied measures. This relies on the applied measures and the calculated KPIs.

D. Problem Detection

Problem detection relies on the receipt of information about CoSEEEK's environment. This is indicated by events in the CoSEEEK infrastructure. For reports of external tools, these events include the location of the reports. Other facts like the execution of load or functional tests may only be indicated by events and are continuously monitored. When reports are received, the *Rule Processing* module is triggered. To be able to automatically evaluate metrics and to assign appropriate measures if thresholds are exceeded, the module must be aware of the metrics, the measures, the thresholds, and the tool used to measure the metrics. Thus, we developed a GUI to more easily define the involved items as depicted in Figure 12.

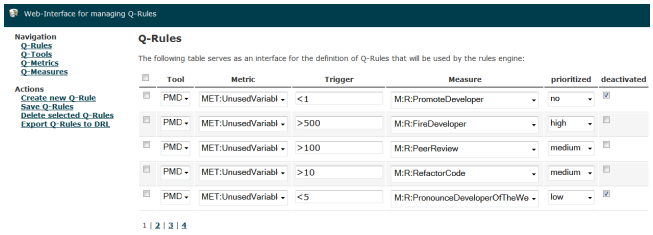


Figure 12. Rules GUI.

The rules produced by this GUI have the structure defined in Listing 1 and allow for the definition of rule priorities. The latter are used if, for example, two rules with different thresholds have been defined for the same metric and only one should be executed if both are triggered. The rules are then exported, transformed into the DRL format utilized by JBoss Drools, and loaded by the *Rules Processing* service. Any new reports then utilize the new rule set.

Listing 1: Rule example

```
<rule
 ID="12"
 Tool="PMD"
 Metric="MET:UnnecessaryConstructor"
 Trigger="&gt;=1"
 Measure="M:R:PeerReview"
 Prioritized="2"
/>
```

Rule processing produces unified XML reports containing all metrics whose thresholds have been violated and their associated triggered quality measures.

E. Measure Proposal

The output of a new unified report triggers the *Context Management* module to start the measure selection. For the prioritization of the measures, the *AGQM* module is triggered. This subsection gives some details about the agents utilized in the *AGQM* module.

The agent structure is defined as depicted in Figure 13. The AGQM agent is responsible for managing the agent module. It instantiates the other agents and determines whether a reactive or proactive measure will be proposed. For each defined goal, one goal agent is instantiated. In the proactive section, the goal agents communicate with the so-called session agent to realize the bidding process. The session agent takes the role of the "buyer" and thus selects the proactive measure from the goal agent with the highest bid. Each goal agent places bids according to its strategy. For the initial implementation, basic strategies were used. The three strategies 'offensive', 'balanced', and 'defensive' influence the starting bid of the agents as well as win-or-lose adaptation based on the last session. The strategy pattern allows these algorithms to vary. If insufficient points are left for the intended bid, the agent bids all points he has left. If an agent has no points left, it cannot place bids anymore until all agents have no points left, whereupon all points are reset to their initial value. Each agent has a list of proactive measures it could offer. Goals that are known to be at risk due to GKPI deviation are elevated to participation status in the bidding. If no report containing GKPI violations is received, all agents participate.

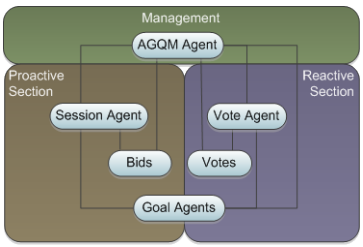


Figure 13. Agent Structure.

The reactive section is realized via the vote agent. Each time a report is received, the vote agent creates a weighted list of reactive measures using the report. To elicit the weight of each measure, the vote agent communicates with the goal agents. For each measure, a goal agent evaluates whether that measure is associated to its goal via the aforementioned

connection of measures, metrics, KPIs, and goals. In each voting process, a goal agent distributes all of its assigned points (initially allocated at the beginning of the iteration) uniformly across all measures in the current report that are associated to its goal. If multiple agents vote on one measure, the points are aggregated. If no report has been received yet, the voting process cannot be conducted. In that case, a proactive session is substituted.

That way the *AGQM* module creates a new ordered list of measures that mirror the predefined importance of the project's quality goals.

F. Measure Application

This part of the process was discussed in our initial approach [25]. However, due to technical issues, it has been extended and refined and this subsection presents how the integration of new quality measure activities into the user's workflow is accomplished. Therefore, new items have to be inserted both on the *Context Management* side and the *Process Management* side. This is done at first in the *Context Management* module. A quality measure is inserted as a new *Assignment* comprising certain *AssignmentActivities* and a separate *WorkUnitContainer* with *WorkUnits* and potential new *ExtensionPoints*. These are created from pre-specified template concepts that are connected to the *MeasureTemplate* (that is mentioned in Figure 17). To be able to insert the quality measure at the specified point, a new *WorkUnit* is created and inserted there, which is then connected to the newly created *WorkUnitContainer* belonging to the quality measure. This is illustrated in Figure 14.

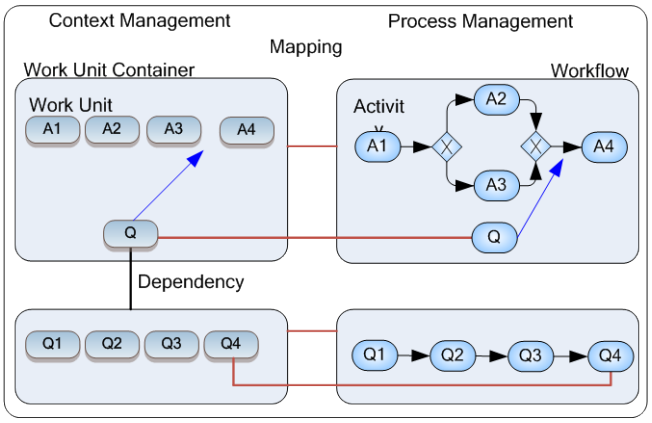


Figure 14. Measure Integration.

However, the insertion itself has to be also done within process management and therefore takes place later. In order to adapt a running workflow instance in AristaFlow, it has to be suspended from execution to apply the adaptations. This cannot be done if an activity in the instance is still active. However, at the time the quality measure integration is triggered, the activity that caused the *Q-Slot* generation is still active. Therefore, suspension is delayed until the activity is completed. This procedure is shown in Figure 15.

After the new individuals (*WorkUnits*, etc.) in the ontology have been created, a so-called *DeferredAction* is created and assigned to the *ExtensionPoint* where the measure should be integrated. That action will be automatically executed when the *WorkUnit* related to the *ExtensionPoint* is finished and will integrate the *Activity* containing the measure (named 'Q' in Figure 14) in the workflow instance. After that, a 'soft suspend' event is sent to *Process Management*, causing AristaFlow to do a soft suspend on the respective workflow instance. Thus, the instance will be automatically suspended right after the currently running activity is finished. This happens when the related *Work Unit* finishes via a 'signal Activity' event. This, in turn, causes the final integration of the quality measure activity in *Process Management* and is mirrored in *Context Management*.

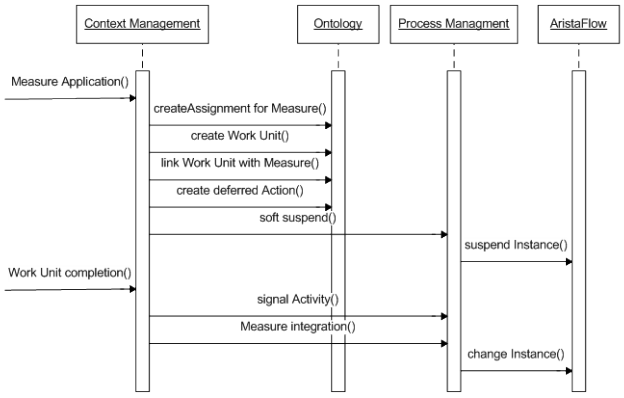


Figure 15. Measure Integration Procedure.

The order of the activities in the integration procedure was chosen in a way that the current activity is still running while the measure integration process starts. This was done to allow the insertion of a quality measure directly after every activity even if this is the activity the user currently processes that caused the creation of a *Q-Slot*. The soft suspend immediately suspends the workflow instance after activity completion and therefore no other activity is started. Then, the quality measure activity can be integrated and executed as the next activity if appropriate.

G. Quality Trend Analysis

The quality trend analysis is conducted in the *Context Management* module and the values are stored in the ontology. The concepts are illustrated in Figure 16.

Both *Metric* and *KPI* are united under the concept of the *QualityIndicator*. All concepts are separated into a template for the definition and a form containing a concrete value. When a *ViolationList* containing multiple *Metrics* is received, it is determined which *KPI* can be calculated via the *KPITemplate* and the *MetricTemplate*. For the computed values, new *KPIs* are then created.

To be able to do a uniform calculation, all received metric values are normalized to values between 0 and 1 where 1 is the best possible value and 0 is the worst possible one. Therefore, as part of the *MetricTemplate*, there is a defined maximum saved. The actual value is divided through this maximum to derive a value between 0 and 1. It also

defines a limit for the value, e.g., if a maximum of 15 has been defined as maximum for cyclomatic complexity, this would be the worst possible value. If the actual values had exceeded this limit, then 15 would be taken instead. There is also a property called *negative*, which indicates whether high values are negative (bad) or positive (good) indicators.

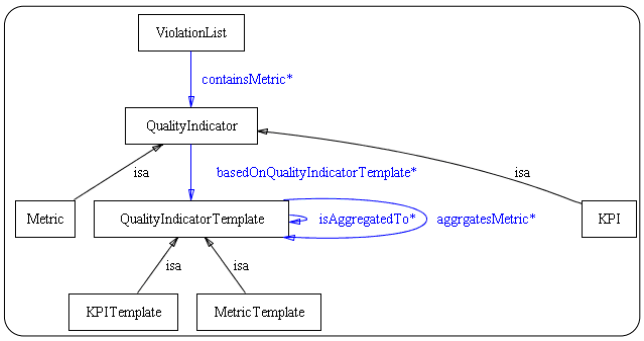


Figure 16. Quality Trend Analysis.

If a metric value is not available, the calculation will be done without that value. For some metrics, the absence of a value is also a negative indicator and thus a standard value can be defined in the *MetricTemplate*. If, for example, a metric had indicated the degree of functional testing compliance (a measure for the outcome of functional testing), its absence would indicate that no functional testing has been done yet. Since that fact should not be overlooked, a standard value can be defined.

It is also possible to integrate values from external tools as KPIs. If this is the case, a property 'external' can be used to indicate this. The *KPI* calculation is a weighted average and therefore each *KPITemplate* stores the weight used for that *KPI*.

H. Measure Assessment

In this part of the process, the calculated values of the KPIs are utilized for recalculating the measure utility factor. This is done using the changes (deltas) of the KPI values. For now, up to ten such deltas starting from the time of the measure application can be used. The concepts in the ontology realizing this are depicted in Figure 17.

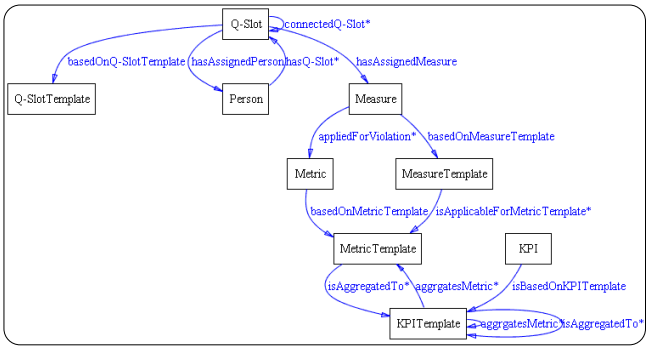


Figure 17. Measure Assessment.

More details on this calculation are provided in [25]. Compared to our initial approach, the ontology structure has been refined featuring the separation into template concepts and concrete ones for all involved concepts. Thus, they conform to the overall structure, implying a strict separation of the definition of certain items and their concrete values. That way, additional plausibility checks are also possible like the check whether a measure is permitted for a certain metric violation.

I. Modeling Effort

The presented approach implies modeling workflows and a myriad of extensions to them in the ontology. This leads to a relatively high modeling effort. That effort can nevertheless be limited: When the modeling is done utilizing the SE workflow language we developed in [24], both the concepts for the *Process Management* and the *Context Management* modules are automatically generated. If workflows are already in place in a workflow management system, the basic concepts in the ontology (*Work Units* and *Work Unit Co*ntainer) can be automatically generated and then be annotated manually. This could also limit the effort required for migrating to CoSEEEK in a company. If a supported workflow management system is in place there, only the annotations (e.g., *Extension Points*) have to be added manually. CoSEEEK will also include predefined metric sets and associated measures, standard SE processes, and GQM plans to facilitate the introduction of the system. That way small and medium sized companies could easily benefit from the higher level of automation CoSEEEK provides.

V. Evaluation

A scenario was constructed and technical measurements taken to evaluate the overall feasibility of the approach.

A. Scenario

Due to the large number of configuration factors involved and the breadth and depth of the approach we developed, a controlled scenario-based evaluation that combines real results with synthetic facts was chosen for initial feasibility testing. As input for code analysis, the org.eclipse.osee.framework.database package of the open source Eclipse Open System Engineering Environment was used.

1) Process

As a software development process, OpenUP [61] was chosen, a simplified free derivative of the Unified Process [62]. This process constitutes an iterative process featuring four project phases. In the *Inception phase* the scope of the project is defined, the use cases are outlined, risks are identified, and candidate architectures are selected. The *Elaboration phase* serves for capturing a healthy majority of system requirements and for addressing known risk factors. In addition, the system architecture is established and validated. In the *Construction phase* the system features are built based on the selected architecture. In the *Transition phase*, the system is deployed to the users. Each phase contains a number of iterations to complete its goals. Figure 18 shows how the OpenUP process can be used in the given scenario.

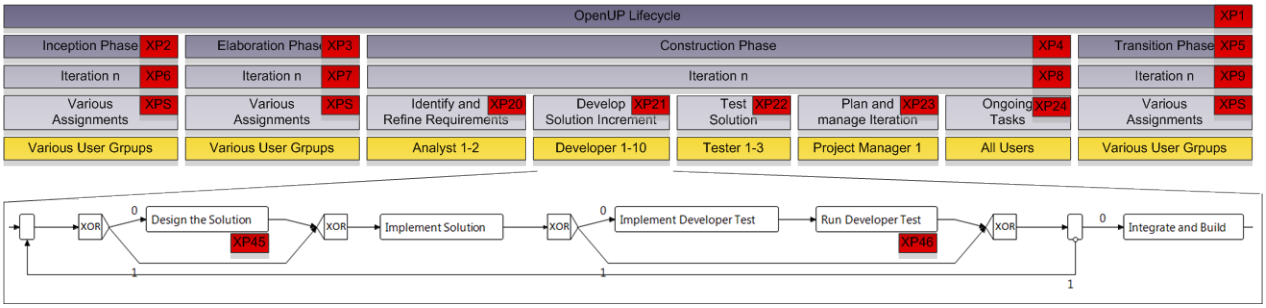


Figure 18. OpenUP Process with Configured Extension Points

Since the Construction phase is the largest phase in the project comprising most development activities, it was the focus of our evaluation. Figure 18 shows the other phases in a compressed way. The focus here is on the developers' activities, thus the 'Develop Solution Increment' workflow is shown in detail in the figure. Overall, 54 *ExtensionPoints* (XPs) have been defined for the workflows as depicted in the figure. The ones that are relevant for the developers' activities in a Construction iteration are XP8 at the end of the iteration, XP21 for measures to be applied between two assignments, and XP45 and XP46 for measures directly relating to coding or testing regarding certain artifacts.

2) GQM Plan

For the test scenario, a GQM plan was created to enable the AGQM agent processes shown in Table I. Four goals have been chosen: maintainability, reliability, performance, and functionality. This is just a simplified example of what is possible and can be incorporated and tailored by a quality manager.

The different metrics and KPIs that are part of the plan are illustrated in Appendix A. To measure the reliability of the code, different kinds of metrics have been chosen. On the one hand, well-known source code metrics like McCabe's cyclomatic complexity [63] or Nejmeh's npath complexity [64] have been used. On the other hand, metric suites were integrated, namely Chidamber and Kemerer's metrics suite [65] as well as the QMOOD metrics suite [66]. According to a study conducted in [67], these are good predictors for fault proneness and thus for reliability. Another factor that could affect the reliability of source code is whether it is covered by unit tests. This metric can be provided by tools like Cobertura [36] or EMMA [37] (see [35] for a comparison). Since, via sensors, it is possible to detect the execution of various tools for various activities, other factors can be used as metrics as well. An example for this is the degree of load testing that can also be an indicator of (the lack of) code reliability confidence.

For maintainability, a set of source code metrics have been selected and grouped to a question concerning the understandability of the code. To enhance the prediction quality of the goal, KPI external approaches have also been integrated: the maintainability index (MI) [68][69][70] is a formula proven to be a good predictor of maintainability and can be provided by the tool jhawk [71]. Maintainability can be also affected by certain problems in the source code called code smells. These can be detected via the DECOR approach [72], which is taken into account as well.

TABLE I. EXAMPLE GQM PLAN.

GKPI	QKPI	KPI	Metric
GKPI:REL	QKPI:CK		MET:WMC MET:DIT MET:NOC MET:CBO MET:RFC MET:LCOM
	QKPI:QMOOD		MET:ANA MET:CAM MET:CIS MET:DAM MET:DCC MET:MOA MET:MFA MET:NOM
	QKPI:COMP		MET:CYC MET:NPA
	QKPI:DD		MET:DD
	QKPI:CC		MET:CC
	QKPI:DLT		MET:DLT
GKPI:MAINT	QKPI:UND	KPI:CSV	MET:CR MET:TMM MET:UEM MET:UEC MET:ECB MET:TMF
	QKPI:CC		MET:CC
	QKPI:CSD		MET:DECOR
	QKPI:MI		MET:JHAWK
GKPI:FUNC	QKPI:UCC		MET:UCC
	QKPI:FTCF		MET:FTCF
GKPI:PERF	QKPI:CTAF		MET:CTAF
	QKPI:PRAF		MET:PRAF
	QKPI:PTCF		MET:PTCF

The implementation of all desired functionality is covered by the functionality goal. Thus, two metrics have been chosen to measure that. The use case coverage indicates how much of the desired functionality is implemented. The *functional testing compliance factor*, in turn, indicates how

many of the functional tests were passed. If no functional testing has been performed yet, the value of the functional testing compliance factor will be 0 in the worst case.

The performance goal comprises a metric called the *performance testing compliance factor*, which is similar to the *functional testing compliance factor*, but deals with performance tests. The other two metrics are related to the code optimization activities of code tuning and profiling.

3) Concrete situation

The scenario is targeted for a construction iteration of the OpenUP process that takes two weeks implying ten workdays. Ten developers are assumed to be part of the team and each developer has ten 'Develop Solution Increment' assignments, which are assumed to take one day each. Each night reports from code analysis tools are received as part of the nightly build process. As static analysis tool, PMD [33] is used; Appendix B shows the results of a report concerning the selected OSEE module. These results also include the threshold for each metric defined via the Rule module. For the concrete iteration, the focus is improving the quality of the source code, especially maintainability, since the functionality metrics are not violated, meaning the desired functionality is largely implemented. Therefore, the goal agents have been defined as depicted in Table II.

TABLE II. GOAL AGENT CONFIGURATION.

Agent	Points	Strategy
MAINT	100	Offensive
REL	80	Balanced
PERF	80	Balanced
FUNC	60	Defensive

For this scenario, the three strategies used for the agents have been defined as shown in Table III. As stated in Section III.F, they comprise three values: a start bid indicating how many of the distributed points an agent uses for its first bid and raise / reduce values indicating how the agent raises (reduces) its bid in case of loss (win).

TABLE III. AGENT STRATEGIES.

Strategy	Start bid	Raise	Reduce
Offensive	35%	20%	10%
Balanced	30%	15%	13%
Defensive	25%	10%	20%

Each time a *Q-Slot* occurs, the *AGQM* module is triggered to output an ordered list of proposed quality measures. For the current scenario, a 50:50 ratio between proactive and reactive measures was defined. Table IV shows the first ten proposed quality measures generated for a Q-Slot. Proactive measures are identified by the prefix "M:P:" and the assigned goal, reactive measures by "M:R:". The related metric whose threshold was violated for reactive measures is also shown.

TABLE IV. PROPOSED QUALITY MEASURES FROM AGQM

Slot	Quality Measure	Related Metric	ID
1	M:P:MAINT:Analyze Reuse Possibilities		m1
2	M:R::Increase Code Coverage	MET:CC	m2
3	M:R.Refactor Code	MET:ECB	m3
4	M:P:MAINT:Review Style Guidelines		m4
5	M:P:REL:Analyze Error Handling Implementation		m5
6	M:R:MAINT:Refactor Code	MET:TMM	m6
7	M:P:PERF:Do Profiling		m7
8	M:P:MAINT:Analyze Modularity		m8
9	M:R:PERF:Do Performance Testing	MET:PTCF	m9
10	M:R:Refactor Code	MET:CYC	m10

To determine the impact of the strategies in conjunction with the distribution of points in the proactive section, Table V shows the agents' bids for the slots, in which proactive measures were proposed. The numbers in parenthesis indicate the bid an agent would have placed according to its strategy when insufficient points were available.

TABLE V. AGENTS BIDS.

Slot	Winner	FUNC	REL	MAINT	PERF
1	MAINT	**35**	24	24	15
4	MAINT	**31**	28	28	17
5	REL	28	**32**	32	19
7	PERF	34	28	**37**	21
8	MAINT	**34(41)**	32	32	23

The results correlate with the expected arrangement of the proposed measures, where maintainability measures should be favored most, followed by reliability and performance measures.

For simplicity, in the current scenario, only early activity completion is assumed to have no defined quality overhead factor. Thus, the creation of Q-Slots only relies on execution time deviations of the assignments. These execution time deviations are shown in Table VI. Positive values indicate that an activity took less time than estimated, negative values indicate longer actual execution times, and grey boxes indicate *Assignments* after which the measure proposal process is started for the respective developer. For this scenario, it was assumed that a quality measure is possible if at least two hours are available.

With these values, five *Q-Slots* are possible in the iteration under consideration for the developers dev1, dev3, dev5, dev9, and dev10. For each Q-Slot, a measure from the list provided by the AGQM module has been selected, proposed, and assessed after application. The chosen measures, the applying developer, and the chosen *ExtensionPoints* are shown in Table VII. The measure utility has been initialized to '1' for all measures in the scenario. The table also shows the relating *KPI* used for assessment and the newly calculated 'measure utility' for the applied

measures. The calculations of the proactive measures have not been included here because in that limited scenario the GKPIs could not reflect an impact of the proactive measures. For a scenario with more details on measure tailoring and measure assessment, we refer to [25].

TABLE VI. EXECUTION TIME DEVIATIONS.

Developer	Assignment									
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
dev1	1	0	2	0	0	-1	1	1	1	1
dev2	0	1	-1	-1	0	1	0	1	1	1
dev3	1	0	-1	0	1	2	0	0	0	0
dev4	0	1	0	0	-2	2	0	-1	-1	-1
dev5	1	-1	1	0	2	0	0	0	0	0
dev6	-4	1	1	1	0	0	0	-1	-1	-1
dev7	1	0	0	0	-1	-2	1	1	1	1
dev8	0	1	0	1	0	1	0	1	0	0
dev9	0	1	0	0	0	0	2	0	0	0
dev10	1	0	-1	0	0	1	2	0	0	0

TABLE VII. APPLIED MEASURES.

Measure	Developer	Extension Point	KPI	Measure Utility
m1	dev1	21	GKPI:MAINT	1
m2	dev3	21	QKPI:CC	1.17
m3	dev5	21	KPI:CSV	1.17
m5	dev9	8	GKPI:REL	1
m9	dev10	21	QKPI:PTCF	1.29

While the scenario is not detailed and broad enough to ensure the applicability for the majority of SE real-world use cases, it shows the feasibility and potential of the approach towards addressing automated GQM and SQM. Future work will include trials of this approach with our industry project partners where empirical results can be evaluated.

B. Performance Measurements

For evaluating the technology and realization choices, performance measurements were conducted. Two different hardware configurations were utilized since the performance testing was performed by different developers on their own hardware (notebooks). Configuration A consisted of a computer with an AMD Turion II Dual-Core Mobile M500 2.2 GHz processor and 4 GB RAM. The software used was Windows 7 64-bit, Java Runtime Environment 1.6.0_16, Scala 2.7.7 final, Drools 5.1.0, Apache Ant 1.8.0, Apache CXF 2.2.4, and eXist 1.4.0. Configuration B consisted of one computer with an Intel Core i7 Q820 1.73 GHz processor and 6GB RAM. The software used was Windows 7 64-bit, the Java Runtime Environment 1.5.0_20, Apache CXF 2.2.4., eXist 1.2.6 (rev. 9165) and Jade 3.7. The tests were executed in a virtual machine (VMware Player 3.0.1 build-227600) assigned two processor cores and 4GB RAM.

All performance measurements were conducted five times consecutively, taking the average of the last three measurements. The first measurement series deals with the rule module and uses Configuration A and the second series covers the AGQM module and uses Configuration B. Other parts of the concept have been measured in [25].

1) Rules processing

Since the largest, most diverse, and regularly occurring amount of data to be analyzed by CoSEEEK are likely to be tool reports, and since the number of thresholds and quality measures needed to manage these can grow correspondingly, the scalability and performance of the *Rules Processing* was measured.

For the rule sets, both the loading latency and the execution time were measured for different numbers of rule sets as depicted in Table VIII. The XML report contained 4000 items generated to violate all of the rule sets that were defined for the test.

TABLE VIII. RULE PROCESSING PERFORMANCE.

Number of rules	Loading latency (sec)	Execution time (sec)
250	3.2	1.5
500	6.6	3.1
750	9.5	4.4
1000	13.0	5.8
1250	13.7	7.5
1500	16.3	8.6
1750	25.8	12.1

For scalability, the measurements show an almost linear increase of computation time. The loading performance is acceptable given that changes in rules, where reloading is necessary, should not be as frequent as rule execution. Since typical SE low-level activities are usually multiple minutes long, the execution time for the worst case measured (12 seconds) is still tolerable, making the approach suitable for the practical use in SE environments. Note that the rule engine would typically be run on a server and not on a notebook.

2) AGQM

For the AGQM module, two measurements were conducted to determine the impact of the number of goals (agents) and measures. First, the reactive measure list creation latency based on voting was measured. Second, the whole measure proposal process for a *Q-Slot* was measured.

The latency for vote list creation with varying numbers of measures and goals is depicted in Table IX. The results show that the number of measures has a greater impact on the latency than any increase in the number of goal agents voting (when measurement inaccuracies regarding the smaller values are disregarded).

TABLE IX. AVERAGE VOTE LIST CREATION LATENCY (MS) VS. GOALS AND MEASURES.

Measures	50	100	500	1000
5 Goals	111	194	273	924
10 Goals	113	160	815	1927
15 Goals	110	263	787	2090
50 Goals	92	317	842	2453
100 Goals	91	342	864	3003

A second measurement considered the measure proposal latency for a slot. It was assumed that for every goal exactly one proactive measure was defined, thus only the number of goals was of interest. All agents were given an offensive strategy and 100 points. For reactive measures, the measure list for voting was already prepared, from which only the first position was retrieved for simplification. The results are shown in Table X.

TABLE X. AVERAGE MEASURE PROPOSAL LATENCY (MS) VS. GOALS.

	5 Goals	10 Goals	15 Goals	50 Goals	100 Goals
Proactive	47	51	45	65	3211
Reactive	40	325	338	492	665

The reactive part shows the overhead of increasing agents for retrieving the top measure from the vote list. The proactive part remains constant for low goal numbers and then reaches an inflection point with a large number of goal agents. One possible explanation is extended bidding and thrashing with thread-based agents - this should be further investigated.

In summary, the performance of the current implementation appears to be sufficient for use in SEEs when the number of goals and measures used are within expected limitations. Performance could become an issue in large teams or projects or when large numbers of reactive measures are triggered. One way to address this would be to tune the *Rules Processing Module* to limit the number of reactive measures for which voting takes place. As to goal scalability, a large number of goals and goal agents would also imply a high degree of configuration overhead for a quality manager, thus likely naturally limiting the number of goals. Should nevertheless a large number of goals be desirable, distributing the agents could be considered.

VI. RELATED WORK

This section provides related work concerning our approach. It is structured into subsections covering the different topics of GQM support, contextual integration, and automated process adaptation.

A. GQM support

The combination of GQM with agents has been used for providing automated support for GQM plan creation [73][74][75] and for the computation of values for questions and goals [76][77]. In [75], a goal-driven use case method is utilized to elicit requirements. A set of agents assists the user in identifying goals and questions that are then used by another agent to obtain metrics. The collection of the measurement data and the creation of the measurement plan are then executed by two other agents. The ISMS (Intelligent Software Measurement System) [73][74] follows a similar approach using different groups of agents for user assistance and determination of different parts of the GQM plan. In [76][77], agents are used in the requirements process of the SW-CMM (Software Capability Maturity Model) model. The focus is the measurement and analysis of software processes using agents and fuzzy logic.

The approach presented in [78] aims at automated user assistance in GQM plan creation and execution but does not utilize agent technology. A tool was developed which allows creating GQM plans that use predefined forms as well as verifying the structural consistency of the plan and the reuse of its components. Furthermore, the tool supports data interpretation and analysis through aggregation of collected data. This approach is extended in [79], which integrates GQM more tightly with a development process to support GQM plan creation by an explicit process model.

For better integrating the GQM technique into the project flow via automation, different approaches were considered. [80] aims at integrating measurement programs as well as data collection into explicit process models, while [81] provides an object- oriented process model whose target is measurement. [82] proposes the usage of process models for creating GQM plans. Finally, the tool Prometheus [83] links executive plans with process models.

An approach extending the GQM technique is presented in [84]. It adds concepts such as entities, attributes, and units. cGQM [85] proposes the use of the Hackystat framework for GQM, applying continuous measurement with short feedback loops.

Other applications of agent technology include its utilization for automatic information retrieval [86], process monitoring [87], and collaboration support [88].

As opposed to the aforementioned approaches, CoSEEEK's AGQM process integrates its techniques into live software engineering environments, actively injecting SQM countermeasure proposals as guidance for developers. Agent technology is used differently in that the aim is neither user assistance in GQM plan creation nor assistance in interpreting measurement results. It is rather the fully automatic monitoring of goal fulfillment and the automatic assignment of quality measures for different types of quality deviations.

B. Contextual Integration of Process Management

Adapting application services to contextual changes is a major research area in areas like pervasive computing. A number of context-aware frameworks have been suggested to facilitate the implementation of application services that can somehow adapt their behavior to changing context. Frameworks like Context Management [89], CASS [90], SOCAM [91], and CORTEX [92] provide support for gathering and processing context data similar to our approach. However, they leave the reaction to context changes to the application or use hard-to-maintain rule-based approaches for dealing with respective changes.

Only few approaches like inContext [93] combine workflows with context-awareness as described in this paper. Regarding inContext, contextual information plays a central role similar to our approach; inContext strongly focuses on the teamwork domain, while our approach delivers a more generic technology enabling the development of context-aware, adaptive workflows.

The semantic annotation of process specifications to enable some method of contextual integration for the latter was addressed by various approaches. The focus of COBRA [94] is business process analysis. It presents a core ontology for business process analysis to provide better and easier analysis of processes to comply with standards or laws like the Sarbanes-Oxley act. A semantic business process repository is presented in [95]. It fosters automation of the business process lifecycle. It features capabilities for checking in and out as well as locking and options for simple querying and complex reasoning.

The approach presented in [96] aims at facilitating process models across various model representations and languages. This is achieved by multiple levels of semantic annotations: a meta-model annotation, a model content annotation, and a model profile annotation as well as a process template modeling language. [97] provides a concept for machine-readable process models to achieve better integration and automation. It utilizes a combination of Petri Nets and an ontology, whereas direct mappings of Petri Net concepts in the ontology are established. The approach described in [98] proposes an effective method for managing and evaluating business processes. This is realized via the combination of semantic and agent technology to monitor business processes. In contrast to the framework presented in this paper, these approaches do not consider the active intervention of a system in the execution of workflows. CoSEEEK exploits semantic annotation of the processes to a greater extent, using them to do context-based process adaptations.

C. Automated Process Adaptation

In the field of business process management, there exist several approaches supporting automated and dynamic adaptations of workflows during run-time [99]. As in our approach, their aim is to reduce error-prone and costly manual workflow adaptations during run-time and thus to relieve users from this task. As opposed to the presented work, the focus of these approaches is on automated exception handling. For this, the process-aware information system must be able to automatically detect exceptional situations, derive the dynamic change necessary to handle them, identify the workflows to be adapted, correctly apply the dynamic change to these workflows, and notify respective users. Existing approaches can be classified according to the basic method used for automatic exception detection and workflow adaptation:

Rule-based approaches. ECA-based (Event-Condition-Action) models are suggested for automatically detecting exceptional situations and determining the actions (i.e., workflow adaptations) required to handle them. In many ECA approaches, however, adaptations are restricted to currently enabled and running activities (e.g., to abort, redo, or skip activity execution) [100]. In contrast, AgentWork [101] further enables automated adaptations of the yet not entered regions of a running workflow (e.g., to add or delete activities). Basic to this is a temporal ECA rule model that allows specifying process adaptations at an abstract level and independent from a particular process model. When an ECA rule fires during run-time, temporal estimates are made to determine which parts of a running process instance are affected by the identified exception. These parts are then adapted immediately (predictive change) or, if this is not possible due to temporal uncertainty, at the time they are entered (reactive change).

Goal-based approaches formalize process goals (e.g., process outputs) and automatically derive the process model (i.e., the activities to be performed and their execution order) based on which these goals can be achieved. Further, if an exception (e.g., an activity failure) occurs during run-time that violates the formal goals, the process instance model is adapted accordingly. In ACT [102] for example, certain workflow adaptations (e.g., replacing a failed activity by an alternative one) are automatically performed if an activity failure leads to a goal violation. EPOS [103] rewrites software engineering workflows when process goals themselves change. Both approaches apply planning techniques to automatically derive and repair workflows in such cases. However, current planning methods do not cover all relevant process scenarios like our approach since important aspects (e.g., treatment of loops, appropriate handling of data flow) are not adequately considered.

Product-driven approaches interpret complex data structures representing a product in order to derive related workflow structures. Corepro, for example, allows product engineers to define complex data structures and to semi-automatically derive workflow structures from them [104]. The latter comprise the concrete workflows for engineering a particular product component (i.e., part) as well as the required synchronization between them. In particular, ad-hoc changes of a product structure are automatically compiled into respective adaptations of the workflow structure (on condition that certain correctness constraints are met). Corepro uses object life cycles and their dependencies in order to represent product components and their relations. DYNAMITE, in turn, uses graph grammars and graph reduction rules for defining the way in which a software engineering workflow may evolve over time [105]. Automatic adaptations are performed depending on the outcomes of previous activity executions (e.g., a design of a software module). Recently, more generic approaches aiming at a tighter integration or process and data have emerged (see [106][107] for an overview). These are particularly interesting for enabling artifact-based processes as in SE. For example, PHILharmonicFlows enables object-aware processes, which consider *object behavior* (i.e., the behavior of single objects and artifacts respectively) as well as *object interactions* (i.e., the coordinated processing of a collection of objects) [27]. Consequently, object-aware processes are based on two levels of granularity. In particular, data-driven process execution is enabled as well as integrated access to processes and data [108].

VII. CONCLUSION

SQA should be aligned to the SE process being used, and be relevant and applicable at the operation level. The manual combination of SQA with SEPM requires constant vigilance and associated labor in order to avoid missing quality

opportunities, to continuously monitor quality goal states, and to adapt measure and measure utility to new quality situations. The application of BPM in SE environments has been sparse due, among other factors, to a lack of contextual adaptability.

Automated quality guidance support could assist developers by providing SQA triggering that is based on current and factual data, continuously monitoring quality goal states and trends, and selecting and tailoring measure selection to that being most appropriate in the current situation. A set of requirements regarding context-awareness, process management, and quality measure selection was established in Section II.

Since the quality data and its analysis is not foreknown for reactive measures, and since there are limited time and resources for proactive measures, an automated selection of activity-based quality measures is beneficial. CoSEEEK's context-aware approach situationally adapts SE processes and ensures that quality opportunities are leveraged with the most appropriate measures for the current project quality risks. These are inserted into the appropriate point in the developer's workflow while taking developer properties such as competencies or available time into account. Quality risks can thus be mitigated and automation support can reduce inefficiencies.

Metric data from various tools can be integrated, thresholds can be continuously monitored, and appropriate measures can be triggered when the thresholds are exceeded. An automated awareness of schedule and early activity completion allows quality opportunities to be leveraged, and an overall quality overhead factor (could vary based on project phase) shall not to be exceeded. Process specification is extended to support flexible connections and dependencies between activities, enabling better context-based adaptations. GQM was extended for concrete metric-based automation support by agents. To deal with the expected plethora of reactive measures in projects, cooperative voting is used for reactive measure selection. For proactive measures, goals at risk bid against each other to allow importance and strategy to determine the point in time when proactive measures supporting their goals are proposed.

Measure selection is automatically tailored using a holistic project context comprising information about the project, tools, people, and their current situation. Measures are seamlessly integrated into running workflows using adaptive process management and semantic technology. Measure assessment adjusts the future use of measures based on their effectiveness, enabling the system to adjust and improve its SQA measure proposals.

A scenario-based evaluation exemplified the approach and showed its feasibility towards addressing automated GQM and SQM. Performance measurements indicated that the realization choices showed no significant scalability or performance issues.

Future work will assess the effectiveness of the approach via case studies in industrial settings. Concrete case studies at two companies have already been started and are expected to yield results soon. Work is also required to address the appropriate planning, determination, placement, and frequency of Q-slots in these industrial settings. More complex agent strategies, in addition to systematic detection of human expertise situations, will also be researched.

ACKNOWLEDGMENTS

The authors wish to acknowledge Andreas Kleiner, Stefan Lorenz, and Muhammer Tüfekci for their assistance with the implementation and evaluation. This work was sponsored by BMBF (Federal Ministry of Education and Research) of the Federal Republic of Germany under Contract No. 17N4809.

REFERENCES

[1] Grambow, G. and Oberhauser, R.: Towards Automated Context-Aware Selection of Software Quality Measures, Proc. of the 5th Int'l Conf. on Software Engineering Advances (ICSEA'10). IEEE Computer Society Press, 2010.

[2] Reijers, H.A. and van der Aalst, W.M.P.: The Effectiveness of Workflow Management Systems: Predictions and Lessons Learned. *Int'l Journal of Information Management*, 56(5), pp. 457-471, 2005.

[3] Heravizadeh, M.: Quality-aware Business Process Management. PhD thesis, Queensland University of Technology, Australia, 2009.

[4] Vollmer, K.: The EA View: BPM Has Become Mainstream, Forrester Research, 2008

[5] Mutschler, B., Reichert, M., and Bumiller, J.: Unleashing the Effectiveness of Process-oriented Information Systems: Problem Analysis, Critical Success Factors and Implications. IEEE Transactions on Systems, Man, and Cybernetics, 38(3), pp. 280-291, 2008.

[6] Brooks, F.P.: No Silver Bullet: Essence and Accidents of Software Engineering, Information Processing, 1986

[7] Glass, R.L.: Software Runaways: Monumental Software Disasters. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[8] Naur, P. and Randell, B.: Software engineering: Report of a conference sponsored by the NATO Science Committee. Garmisch, Germany, Scientific Affairs Division, NATO, 1968..

[9] Jones C.: Get Software Quality Right. In: Dr Dobb's Journal, June 28, 2010

[10] Eveleens, J.K. and Verhoef, C.: Quantifying IT forecast quality. Sci. Comput. Program. 74(11-12), pp. 934-88, 2009.

[11] Yourdon, E.: Death March, 2nd edition, Pearson Education, 2003

[12] Abdel-Hamid, T.: The economics of software quality assurance: a simulation-based case study, MIS Quarterly, 12(3), pp. 395-411, 1988.

[13] Kan, S.H.: Metrics and Models in Software Quality Engineering, Addison-Wesley, 2002

[14] Soini, J., Tenhunen, V., and Tukiainen, M.: Current Practices of Measuring Quality in Finnish Software Engineering Industry, In Richardson, I., Runeson, P., and Messnarz, R. (Eds.): Software Process Improvement, pp. 100-110, Springer, 2006.

[15] Gibson, D., Goldenson, D., and Kost, K.: Performance Results of CMMI-Based Process Improvement, Technical Report, CMU/SEI-2006-TR-004, Carnegie Mellon Software Engineering Institute, 2006

[16] McConnell, S.: Nine Deadly Sins of Project Planning, IEEE Software 18(5), pp. 5-7, 2001

[17] Slaughter, S.A., Harter, D.E., and Krishnan, M.S.: Evaluating the cost of software quality, Communications of the ACM, 41(8), pp. 67-73, 1998.

[18] Rausch, A., Bartelt, C., Ternité, T., and Kuhrmann, M.: The V-Modell XT Applied - Model-Driven and Document-Centric Development, Proc. 3rd World Congress for Software Quality, Vol. III, Online Supplement, pp. 131-138, 2005.

[19] WfMC. 1993. Workflow management coalition. http:// www. wfmc.org/

[20] Hill, J.B., Pezzini, M., and Natis, Y.V.: Findings: Confusion remains regarding BPM terminologies. Report No. G00155817, Gartner Research, 2008.

[21] Oberhauser, R.: Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments, In: "Semantic Web, In-Tech, 2010.

[22] Grambow, G., Oberhauser, R., and Reichert, M.: Semantic Workflow Adaption in Support of Workflow Diversity, Proc. 4th Int'l Conf. on Advances in Semantic Processing (SEMAPRO'10), Florence, 2010, pp. 158-165

[23] Grambow, G., Oberhauser, R., and Reichert, M.: Semantically-Driven Workflow Generation using Declarative Modeling for Processes in Software Engineering, Proc. of the 4th Int'l Workshop on Evolutionary Business Processes, IEEE Computer Society Press (accepted for publication).

[24] Grambow, G., Oberhauser, R., and Reichert, M.: Towards a Software Engineering Workflow Language, Proc. 10th IASTED Conference on Software Engineering, Innsbruck, Austria, 2011.

[25] Grambow, G., Oberhauser, R., and Reichert, M.: Employing Semantically Driven Adaptation for Amalgamating Software Quality Assurance with Process Management, Proc 2nd Int'l Conf. on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE'10), Lisbon, pp. 58-67, 2010.

[26] Müller, D., Herbst, J., Hammori, M., and Reichert, M.: IT Support for Release Management Processes in the Automotive Industry. Proc. 4th Int'l Conf. on Business Process Management (BPM'06), Vienna, Austria, pp. 368-377, 2006

[27] Künzle, V. and Reichert, M.: PHILharmonicFlows: towards a framework for object-aware process management. Journal of Software Maintenance and Evolution: Research and Practice, 23(4), pp. 205-244, Wiley, 2011

[28] Künzle, V. and Reichert, M.: Integrating Users in Object-aware Process Management Systems: Issues and Challenges. Proc. BPM'09 Workshops, 5th Int'l Workshop on Business Process Design (BPD'09), Ulm, Germany, pp. 29-41, LNBIP 43(1), 2009.

[29] Sadiq, S., Orlowska, M., Sadiq, W., and Schulz, K.: When workows will not deliver: The case of contradicting work practice. In: Proc. BIS'05. (2005)

[30] Basili, V., Caldiera, G., and Rombach, H.D.: Goal Question Metric Approach, Encycl. of Software Engineering, John Wiley & Sons, pp. 528-532, 1994

[31] Luckham, D.C.: 'The power of events: an introduction to complex event processing in distributed enterprise systems' Addison-Wesley, 2001)

[32] microTOOL in-Step: http://www.microtool.de/instep/en/index.asp [Jan .2011]

[33] Copeland, T.: PMD Applied, Centennial Books, 2005

[34] Ayewah, N,, Hovemeyer, D., Morgenthaler, J. D., Penix, J., and Pugh, W.: Experiences using static analysis to find bugs, IEEE Software, 25(5), pp. 22-29, 2008.

[35] Yang, Q., Li, J.J., and Weiss, D.: A survey of coverage based testing tools, Proc. Intl. Workshop on Automation of Software Testing (AST'06), pp. 99–103. ACM Press, 2006.

[36] Cobertura http://www.cobertura.sourceforge.net [Jan 2011]

[37] EMMA http://www.emma.sourceforge.net [Jan 2011]

[38] Johnson, P. M.: Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System, Proc. of the 1st Int'l Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society, pp. 81-90, 2007.

[39] Esper: http://esper.codehaus.org/ [Jan 2011]

[40] Gelernter, D.: Generative communication in Linda, ACM Transactions on Programming Languages and Systems, 7(1):80-112, 1985.

[41] Meier, W.: eXist: An Open Source Native XML Database, Web, Web-Services, and Database Systems, Springer, ,pp. 169-183, 2009.

[42] Browne, P.: JBoss Drools Business Rules. Packt P. Browne. JBoss Drools Business Rules. Packt Publishing, 2009.

[43] O'Brien, P.D. and Nicol, R.C.: FIPA — Towards a Standard for Software Agents, BT Technology Journal, 16 (3):51-59, 1998.

[44] Bellifemine, F., Poggi, A., and Rimassa, G.: JADE - A FIPA-compliant Agent Framework, Proc. 4th Int'l Conf. and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents. London, 1999.

[45] Gasevic, D., Djuric, D., and Devedzic, V.: Model driven Architecture and Ontology Development, Springer, 2006.

[46] World Wide Web Consortium: OWL Web Ontology Language Semantics and Abstract Syntax, 2004

[47] World Wide Web Consortium: Resource Description Framework (RDF) Concepts and Abstract Syntax, 2004

[48] Prud'hommeaux, E. and Seaborne, A.: 'SPARQL Query Language for RDF, W3C WD 4, 2006.

[49] Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., and Katz, Y.: Pellet: A Practical OWL-DL Reasoner. Journal of Web Semantics 5(2), pp. 51-53, 2006

[50] McBride, B.: Jena: a semantic web toolkit, Internet Computing, 2002

[51] Dadam, P. and Reichert, M.: The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support - Challenges and Achievements. Computer Science - Research and Development, Springer. 23(2), pp. 81-97, 2009.

[52] Reichert, M. et al: Enabling Poka-Yoke Workflows with the AristaFlow BPM Suite. Proc. BPM'09 Demonstration Track, Ulm, Germany, 2009.

[53] Reichert, M., Rinderle-Ma, S., and Dadam, P.: Flexibility in Process-aware Information Systems. LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), Special Issue on Concurrency in Process-aware Information Systems. LNCS 5460, pp. 115-135, 2009

[54] Weber, B., Reichert, M., and Rinderle-Ma, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. Data and Knowledge Engineering, Elsevier, 66(3), pp. 438-466, 2008

[55] Lanz, A., Kreher, U., Reichert, M., and Dadam, P.; Enabling Process Support for Advanced Applications with the AristaFlow BPM Suite. Proc. of the Business Process Management 2010 Demonstration Track, September 2010, Hoboken, New Jersey, USA.

[56] Reichert, M., Dadam, P., Rinderle-Ma, S., Jurisch, M., Kreher, U., and Goeser, K.: Architecural Principles and Components of Adaptive Process Management Technology. In: PRIMIUM - Process Innovation for Enterprise Software. Lecture Notes in Informatics , Vol. P-151, pp. 81-97, 2009.

[57] Reichert, M. and Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, 10(2), pp. 93-129, 1998

[58] Li, C. and Reichert, M. and Wombacher, A.: Mining Business Process Variants: Challenges, Scenarios, Algorithms. Data & Knowledge Engineering, 70(5), pp. 409-434, Elsevier, 2011.

[59] Günther, C.W. and Rinderle-Ma, S. and Reichert, M. and van der Aalst, W.M.P. and Recker, J.: Using Process Mining to Learn from Process Changes in Evolutionary Systems. Int'l Journal of Business Process Integration and Management, Special Issue on Business Process Flexibility, 3(1), pp. 61-78, 2008.

[60] Weber, B. and Reichert, M. and Wild, W. and Rinderle-Ma, S.: Providing Integrated Life Cycle Support in Process-Aware Information Systems. Int'l Journal of Cooperative Information Systems, 18(1), pp. 115-165, World Scientific Publ, 2009.

[61] OpenUp http://epf.eclipse.org/wikis/openup/ [November 2010]

[62] Scott, K.: The Unified Process Explained, Addison-Wesley Longman Publishing Co., Inc., 2002

[63] T. J. McCabe: A complexity measure, IEEE Trans. Software Eng. 2(4), pp. 308-320, 1976.

[64] Nejmeh, B.A.: NPATH: A measure of execution path complexity and its applications. Comm. of the ACM, 31(2):188-200, 1988.

[65] Chidamber, S.R. and Kemerer, C.F.: A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, 20 (6), pp. 476-493, 1994.

[66] Bansiya, J. and Davis, C.: A Hierarchical model for object-oriented design quality assessment, IEEE Transactions on Software Engineering, 28(1), pp. 4–17, 2002.

[67] Olague, H.M., Etzkorn, L.H., Gholston, S., and Quattlebaum, S.: Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes, IEEE Transactions Software Engineering., 33(6), pp. 402-419, 2007.

[68] Oman, P.W., Hagemeister, J., and Ash, D.: A Definition and Taxonomy for Software Maintainability, Technical Report #91-08-TR, Software Engineering Test Laboratory, University of Idaho, Moscow, ID, 1991.

[69] Coleman, D.: Assessing Maintainability, Proc. of the Software Engineering Productivity Conference 1992, Hewlett-Packard, Palo Alto, CA, pp. 525-532, 1992.

[70] Coleman, D., Ash, D., Lowther, B., and Oman, P.W.: Using Metrics to Evaluate Software System Maintainability, IEEE Computer, 27(8), pp. 44-49, 1994.

[71] JHawk http://www.virtualmachinery.com/jhawkprod.htm [November 2010]

[72] Moha, N., Gueheneuc, Y.G., Duchien, L., and Meur, A.F.: DECOR: A method for the specification and detection of code and design smells, IEEE Transactions on Software Engineering, 36(1), pp.:20-36, 2010.

[73] Chen, T., Homayoun Far, B., and Wang, Y.: Development of an Intelligent Agent-Based GQM Software Measurement System, Proc. 12th Asian Test Symposium (ATS), pp. 188-197, 2003.

[74] Junling Huang, Far, B.H.: Intelligent software measurement system (ISMS), Canadian Conf. on Electrical and Computer Engineering, pp. 1033–1036, 2005.

[75] FanJiang, Y.-Y and Wu, C.-H.: Towards a Multi-agents Architecture for GQM Measurement System, Proc. 9th Int'l Conf. on Hybrid Intelligent Systems, pp. 277-280, 2009.

[76] Seyyedi, M.A., Teshnehlab, M., and Shams, F.: Measuring software processes performance based on the fuzzy multi agent measurements, Proc. Intl Conf. on Information Technology: Coding and Computing - Volume II. ITCC. IEEE Computer Society, Washington, DC, pp. 410-415, 2005.

[77] Seyyedi, M.A., Shams, F., and Teshnehlab, M.: A New Method For Measuring Software Processes Within Software Capability Maturity Model Based On the Fuzzy Multi-Agent Measurements, Proc. World Academy Of Science, Engineering and Technology Vol. 4, pp. 257-262, 2005.

[78] Lavazza, L.: Providing automated support for the GQM measurement process, IEEE Software, 17(3), pp.:56-62, 2000.

[79] Lavazza, L. and Barresi, G.: Automated support for process-aware definition and execution of measurement plans, Proc. 27th Int'l Conf. on Software Engineering, pp. 234 – 243, 2005.

[80] Lott C.M. and Rombach H.D.: Measurement-based guidance of software projects using explicit project plans, Information and Software Technology 35(6-7), pp. 407-419, 1993.

[81] Morisio M.: Measurement Processes are Software Too, Journal of Systems and Software 49(1), pp. 17-31, 1999.

[82] Broeckers A., Differding C., and Threin G.: The Role of Software Process Modeling in Planning Industrial Measurement Programs, Proc. Int. Metrics Symposium, Berlin 1996.

[83] Visaggio, G.: Process Improvement Through Data Reuse, IEEE Software 11(4), pp. 76-85, 1994.

[84] De Panfilis, S., Kitchenham B., and Morfuni N.: Experiences introducing a measurement program, Information and Software Technology 39(11), pp. 745-754, 1997.

[85] Lofi C.: cGQM - Ein zielorientierter Ansatz für kontinuierliche, automatisierte Messzyklen, Proc. 4th National Conf. on Software Measurement and Metrics (DASMA MetriKon 2005), 2005.

[86] Pelletier, S.-J., Pierre, S., and Hoang, H.H.: Modeling a Multi-Agent System for Retrieving Information from Distributed Sources, Journal of Computing and Information Technology, 11(1), pp. 15-39, 2003.

[87] Wang, M., Wang, H., and Xu, D.: The design of intelligent workflow monitoring with agent technology, Knowledge-Based Systems, 18(6), pp. 257-266, 2005.

[88] Tan, W., Chen, R., Shen, W., Zhao, J., and Hao, Q.: An Agent-Based Collaborative Enterprise Modeling Environment Supporting Enterprise Process Evolution, Computer Supported Cooperative Work in Design III, pp. 217-226, 2007

[89] Korpipipää P. et al.: Managing context information in mobile devices. IEEE Pervasive Computing 2(3), pp.42-51, 2003

[90] Fahy, P. and Clarke, S.: CASS – a middleware for mobile context-aware applications. Proc. Workshop on Context-awareness (held in connection with MobiSys'04), 2004.

[91] Gu, T., Pung, H.K., and Zhang, D.Q.: A middleware for building context-aware mobile services. Proc. IEEE Vehicular Technology Conference (VTC), Milan, Italy, pp. 2656 – 2660, 2004.

[92] Biegel, G. and Cahill, V.: A framework for developing mobile, context-aware applications. Proc. 2nd IEEE Conference on Pervasive Computing and Communication, pp. 361 - 365 , 2004

[93] Dorn C., Dustdar S.: Sharing Hierarchical Context for Mobile Web services. Distributed and Parallel Databases 21(1), pp. 85-111, 2007.

[94] Pedrinaci, C., Domingue, J., and Alves de Medeiros, A.: A Core Ontology for Business Process Analysis, LNCS 5021, pp. 49-64, 2008.

[95] Ma, Z., Wetzstein, B., Anicic, D., Heymans, S., and Leymann, F.: Semantic Business Process Repository, Proc. Workshop on Semantic Business Process and Product Lifecycle Management, pp. 92–100, 2007

[96] Lin, Y. and Strasunskas, D.: Ontology-based Semantic Annotation of Process Templates for Reuse, Proc.10th Int'l Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'05), 2005.

[97] Koschmider, A. and Oberweis, A.: Ontology based Business Process Description, Proc. CAiSE´05 Workshops, pp. 321-333, 2005.

[98] Thomas, M., Redmond, R., Yoon, V., and Singh, R.: A Semantic Approach to Monitor Business Process Performance, Communications of the ACM 48(12), pp. 55-59, 2005

[99] Weber, B. and Sadiq, S. and Reichert, M:. Beyond Rigidity - Dynamic Process Lifecycle Support: A Survey on Dynamic Changes in Process-aware Information Systems. Computer Science - Research and Development, 23(2), pp. 47-65, Springer, 2009.

[100] Casati, F., Ceri, S., Paraboschi, S., and Pozzi, G.: Specification and implementation of exceptions in workflow management systems. ACM TODS, 24(3), pp. 405–451, 1999.

[101] Müller, R., Greiner, U., and Rahm, E.: AGENTWORK: A workflow system supporting rule–based workflow adaptation. Data & Knowledge Engineering, 51(2), pp. 223–256, 2004.

[102] Beckstein, C. and Klausner, J.: A planning framework for workflow management. Proc. Workshop Intelligent Workflow and Process Management. Stockholm, 1999.

[103] Liu, C. and Conradi, R.: Automatic replanning of task networks for process model evolution. Proc. European Software Engineering Conference, pp. 434–450. Garmisch, Germany, 1993.

[104] Müller, D., Reichert, M., and Herbst, J.: A new paradigm for the enactment and dynamic adaptation of data-driven process structures. Proc. CAiSE'08, pp. 48–63, 2008

[105] Heimann, P., Joeris, G., Krapp, C., and Westfechtel, B.: DYNAMITE: Dynamic task nets for software process management. Proc. Int'l Conf. Software Engineering (ICSE'06), pp. 331–341, 1996

[106] Künzle, V. and Weber, B. and Reichert, M.: Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches. Int'l Journal of Information System Modeling and Design (IJISMD), 2(2), pp. 19-46, IGI Global, 2011.

[107] Künzle, V. and Reichert, M.: Striving for Object-aware Process Support: How Existing Approaches Fit Together In: Proc. 1st Int'l Symposium on Data-driven Process Discovery and Analysis, Campione d'Italia, 2011 (accepted for publication).

[108] Künzle, V. and Reichert, M.: A Modeling Paradigm for Integrating Processes and Data at the Micro Level. In: Proc. 12th Int'l Working Conference on Business Process Modeling, Development and Support (BPMDS'11), London, June 2011, LNBIP 81, pp. 201-215, 2011

APPENDIX A: LIST OF UTILIZED METRICS

GKPI:REL:Reliability
GKPI:MAINT:Maintainablity
GKPI:FUNC:Functionality
GKPI:PERF:Performance
QKPI:CK:ChidamberAndKemerer
QKPI:QMOOD:QmoodMetricsSuite
QKPI:COMP:Complexity
QKPI:DD:DefectDensity
QKPI:CC:CodeCoverage
QKPI:DLT:DegreeOfLoadTesting
QKPI:UND:Understandability
QKPI:CSD:CodeSmellDensity
QKPI:MI:MaintainablitiyIndex
QKPI:UCC:UseCaseCovrage
QKPI:FTCF:FunctionalTestingComplienaceFactor
QKPI:CTAF:CodeTuningActivityFactor
QKPI:PAF:ProfilingActivityFactor
QKPI:PTCF:PerformanceTestComplianceFactor
KPI:CSV:CodingStyleViolations
MET:WMC: Weighted Methods per Class
MET:DIT: Depth of Inheritance Tree
MET:NOC: Number of Childeren
MET:CBO: Coupling between Objects
MET:RFC: Response for Class
MET:LCOM: Lack of Cohesion in Methods
MET:ANA:AvgNumberOfAncestors
MET:CAM:CohesionAmongMethods
MET:CIS:ClassInterfaceSize
MET:DAM:DataAccessMetric
MET:DCC:DirectClassCoupling
MET:MOA:MeasureOfAggregation
MET:MFA:MeasureOfFunctionalAbstraction
MET:NOM:NumberOfMethods
MET:CYC: CyclomaticComplexity
MET:NPC:NPathComplexity
MET:DD: DefectDensity
MET:CC: CodeCoverage
MET:DLT: DegreeOfLoadTesting
MET:CR:CommentRatio
MET:TMM:TooManyMethods
MET:UEM:UncommentedEmptyMethod
MET:UEC:UncommentedEmptyConstructor
MET:ECB:EmptyCatchBlock
MET:TMF:TooManyFields
MET:UCC:UCC:UseCaseCovrage
MET:FTCF:FunctionalTestingComplienaceFactor
MET:CTAF:CodeTuningActivityFactor
MET:PAF:ProfilingActivityFactor
MET:PTCF:PerformanceTestComplianceFactor

APPENDIX B: PMD RESULTS

Metric	Value	Violation Threshold
MET:AccClGen	1	5
MET:AvoidDeeplyNestedIfStmts	2	5
MET:AvoidInstanceof ChecksInCatchClause	1	5
MET:AvoidReassigningParameters	1	5
MET:AvoidSynchronized AtMethodLevel	2	5
MET:ClassWithOnlyPrivate ConstructorsShouldBeFinal	4	10
MET:CloseResource	1	5
MET:CollapsibleIfStatements	1	20
MET:CompareObjectsWithEquals	1	5
MET:ConfusingTernary	6	20
MET:CyclomaticComplexity	4	2
MET:EmptyCatchBlock	2	1
MET:EmptyMethodInAbstract ClassShouldBeAbstract	2	5
MET:ExcessiveImports	1	5
MET:ExcessivePublicCount	1	5
MET:LooseCoupling	4	20
MET:NPathComplexity	1	5
MET:OverrideBothEquals AndHashcode	1	5
MET:PositionLiterals FirstInComparisons	1	5
MET:SimplifyBooleanExpressions	2	5
MET:SingularField	1	5
MET:StaticMethods	1	5
MET:SwitchStmts ShouldHaveDefault	1	5
MET:TooManyFields	1	5
MET:TooManyMethods	4	3
MET:Uncommented EmptyConstructor	5	5
MET:UncommentedEmptyMethod	5	5
MET:UnconditionalIfStatement	1	5
MET:UseCollectionIsEmpty	2	5
MET:UseLocaleWith CaseConversions	2	5

Compact and Efficient Modeling of GUI, Events and Behavior Using UML and Extended OCL

Dong Liang, Bernd Steinbach
Institute of Computer Science
Freiberg University of Mining and Technology
Freiberg, Germany
email: liang@mailserver.tu-freiberg.de, steinb@informatik.tu-freiberg.de

***Abstract*—The model driven architecture (MDA) allows to move the software development form the time consuming and error-prone level of writing program code to the next higher level of modeling. The MDA requires tools for modeling, transformation of models, and code generation. In the past, we have developed such tools successfully. Using these tools we recognized serious problems preparing concise, uniform, and complete models using the unified modeling language (UML). In detail these problems concern first the specification and parameterization of GUI elements, second the event handling, and third the modeling of the required behavior. In this paper we show efficient solutions for these problems using the object constraint language (OCL) together with the UML for modeling. While the parameterization of GUI elements can be solved with the OCL directly, the last two problems were solved by an extension of the OCL into an executable OCL, which we call XOCL. We show the benefits of all three new approaches by means of an example of a complete platform independent model (PIM).**

***Keywords*-OCL extension, action language, event handling, platform independent model, class diagram**

I. INTRODUCTION

Traditionally, developing software means writing code in one of the programming languages. Recently, a novel approach called MDA (Model Driven Architecture) [6], which is proposed by OMG (Object Management Group), has evoked more and more attentions in software development. Instead of writing code directly, MDA suggests that software developers model their software products in a platform independent way. Such a software model is called Platform Independent Model (PIM). Then an MDA-tool is used to transform the PIM into one or more Platform Specific Models (PSM). The PSMs have involved detailed information for implementation. Hence, code generation from a PSM is straightforward. In order to realize model transformation, two different strategies are feasible. One of them defines the model transformation process in a high level specification language. The QVT (Query, View and Transformation) language [7][8] supported by OMG is the de facto standard for this strategy. The QVT allows to define a transformation in an imperative approach. Then a QVT compiler generates an implementation (e.g., in Java) of this transformation specified in the QVT source file as a model compiler, which is dedicated to this transformation. The other strategy is to develop the model compiler itself as an all-purpose model transformation framework as well as to provide all necessary information about the underlying target platform the PSM based on, in the form of Target Platform Models (TPM). To prove the second strategy, the model compiler MOCCA (Model Compiler for reConfigurable Architecture) [9][10] was developed in our institute. According to the actual state of MOCCA, a PIM can be transformed into C++ code, Java code for software design, as well as C++/VHDL code for software hardware co-design.

For the both strategies, one issue is still challenging. That is, how to create a PIM concisely, uniformly and completely. In this paper, we assume that all the PIMs are modeled using the Unified Modeling Language (UML) [5] and the Object Constraint Language (OCL) [4]. Based on our experiences with MOCCA, three sub-issues concerning creating PIMs are found.

1) Modeling GUI-layout of a GUI-based application and parameterize all the GUI elements in standard UML is a serious problem, because the model must contain the structural composition of GUI as well as all the geometrical and visual information of the GUI elements. There is no UML diagram type appropriate for both of these aspects.
2) Modeling event handling for GUI-based application in standard UML is time-consuming, because almost all programming languages have their own GUI libraries and the underlying event models and their details blow up the UML-model in an unnecessary manner.
3) Modeling behaviors in PIM concisely is very difficult, because on the one hand there is no standard universal action language based on the UML Action Semantics [5]; on the other hand, specifying behaviors using one of the UML behavioral diagrams can result in a model more complex than the target codes themselves.

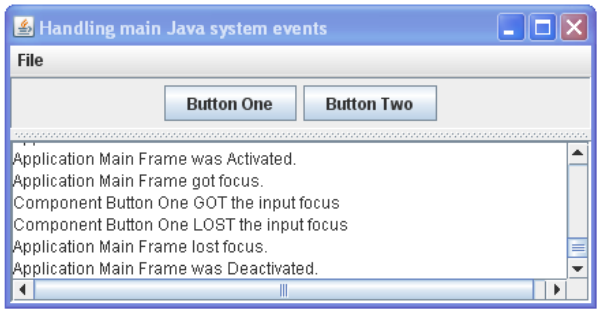
In order to solve these problems, developing new modeling approaches based on UML and OCL are the main aims of this paper. In Section II, a new approach will be introduced that uses UML Class Diagram and OCL-init-

expressions to model and to parameterize GUI elements. In Section III, the OCL is extended by the ability to register event handlers for an event source. This approach helps to create real PIMs for MDA-technology. In Section IV, the OCL is upgraded from a pure declarative language without side effects into an action language, which can be used to specify all kinds of operations in a concise and platform independent manner. In Section V we summarize these three new approaches in an example of a complete PIM.

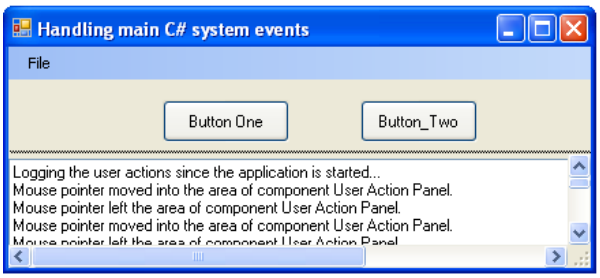
II. OCL-INIT-EXPRESSIONS – OUR NEW APPROACH TO PARAMETERIZE GUI ELEMENTS

In GUI-based application the user interacts usually with a window containing different kinds of GUI elements, which are typically menu items, icons, buttons, input fields etc. According to different platforms, these GUI elements may also be called GUI components or GUI controls. A GUI element usually represents a certain graphical entity that can be displayed on the screen. So they typically have parameters, which concern displaying them on the screen correctly. Such parameters are position, size, background- and foreground-color etc. Most modern object-oriented programming languages implement common GUI elements as classes and their important properties as attributes of the corresponding classes. They are deployed in libraries and can be used in certain languages. Hence, the programmers can use them directly. Figure 1 shows such a GUI-based application implemented in both Java and C#. Both implementations are similar in appearance, because for each GUI element involved in Java implementation, there is a C# counterpart. This analysis gives us a heuristic to model GUI in a platform independent manner. Since there are so many common points among GUI elements on different platforms, we can abstract a platform independent GUI tool-kit with most common GUI elements and their important properties for the general usage to develop a PIM for an application.

In fact, MOCCA supports this principle inherently. A Design Platform Model (DPM) [9][10] contains the most basic design types for primitive data types, IO facilities etc., has been used to establish PIMs. Such a GUI toolkit is just another extension of MOCCA DPM, which is still being developed. The class diagram in Figure 2 models the GUI elements of the application in Figure 1 in a completely platform independent manner. The types with prefix *DP*, which means Design Platform, are the common GUI elements involved in the GUI toolkit of MOCCA DPM. For example, the design type *DPWindow* can be considered as a common abstraction of both *JFrame* in Java and *Form* in C#. The properties defined in *DPWindow*, such as *length* and *height*, are used to model GUI elements exactly. The tool used to create this model is our own CASE Tool called *UML 2 Designer*. However, modeling in this way shows neither the composition structure nor the visualization of the GUI



(a) Java Implementation



(b) C# Implementation

Figure 1. A GUI-based application implemented in Java and C#

elements on the screen. The missing information has to be involved in the design model in a reasonable way.

Implementing GUI-layout as source code using a modern object-oriented programming languages is as complicated as modeling it in a design model. The powerful modern IDEs usually solve this problem by integrating an additional software component, which supports visual manipulation of GUI-elements. For C# the *Form Editor* of Visual Studio IDE can be used for this purpose, whereas the *Swing GUI Builder* integrated into the NetBeans IDE is the counterpart for Java. Both of them support programmers in a similar way. For a class implementing the GUI-layout of an application, e.g., the derived class of the *Form* base class in C#, there is a *design view* associated with it. The programmer chooses the required GUI elements from a *Toolbox*, which contains all the supported GUI elements in this context, positions them on the form, and edits them visually. Figure 3 shows how to use the Form Editor to edit the GUI elements of the application in Figure 1 (b) in a manner explained above.

After editing the GUI elements, all the geometrical and visual information are stored in the *Property Window*, which is visible on the right side of Figure 3. The Visual Studio generates C# codes automatically, which reflect the visual manipulation of the GUI elements done in the design view. These generated program statements are displayed in the C# source code view. Figure 4 shows the generated C# codes initializing the first button in Figure 1 (b).

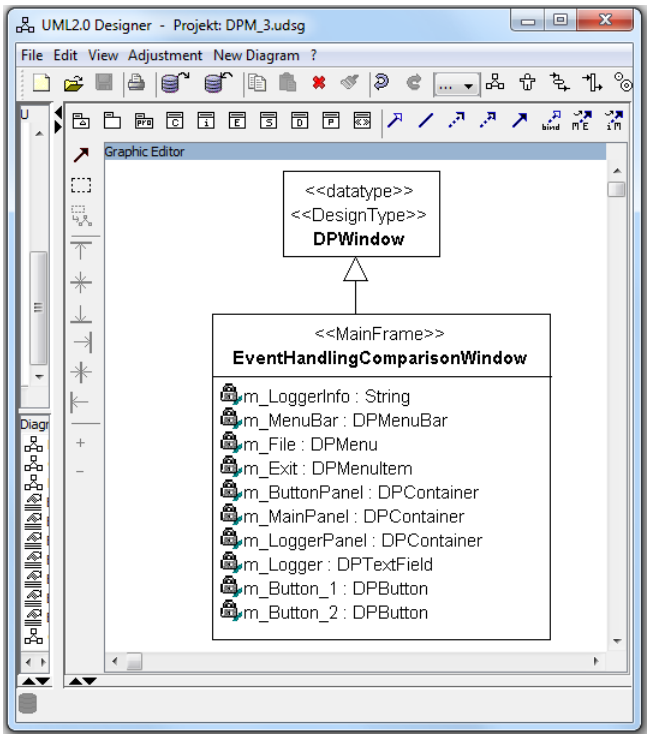


Figure 2. Platform independent model for the main window of the application in Figure 1

It is easy to understand that the *Swing GUI Builder* generates Java codes to reflect the visual manipulation of the Swing components. It is clear that both scenarios mentioned above are platform specific. Based on the considerations above, we suggest a solution to model the GUI-layout in the phase of establishing the PIM of a GUI-based application, which can be summarized as follows:

- A platform independent GUI toolkit is required that contains common GUI elements and their properties as the building blocks of a PIM. As explained, the prototype of this GUI toolkit has been created for our MOCCA DPM and can be used directly in our CASE Tool UML 2 Designer.
- The logical structure of an application window containing various GUI elements can be modeled in UML class diagram, as shown in Figure 2.
- The GUI-layout of an application window can be visually manipulated in a *view* associated to the UML class, which models the logical structure of an application window. This additional view is an add-on software component of the UML 2 Designer, whose prototype has been developed in a bachelor thesis and its upgrade version will be developed in another bachelor thesis. We call this software component the *Smart GUI Editor*. Figure 5 shows its usage. It is easy to understand that the Smart GUI Editor shares the same principles as the Form Editor and Swing GUI Builder.

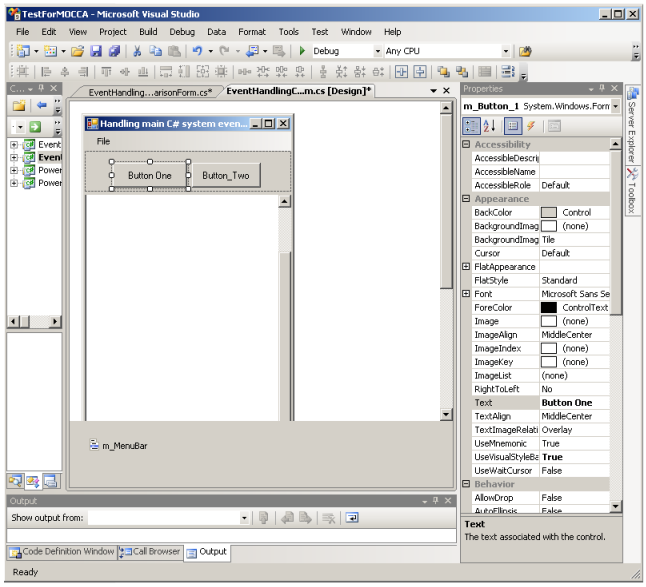


Figure 3. The Form Editor of Visual Studio 2008 used to edit the GUI elements of the application in Figure 1 (b)

```
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    // m_Button_1
    this.m_Button_1 = new System.Windows.Forms.Button();
    this.m_Button_1.Location = new System.Drawing.Point(35, 15);
    this.m_Button_1.Name = "m_Button_1";
    this.m_Button_1.Size = new System.Drawing.Size(93, 31);
    this.m_Button_1.TabIndex = 0;
    this.m_Button_1.Text = "Button One";
    this.m_Button_1.UseVisualStyleBackColor = true;
    this.m_Button_1.Click +=
                new System.EventHandler(this.m_Button_1_Click);
}
```

Figure 4. C# code generated by Visual Studio Form Editor

- In contrast to the Form Editor in Visual Studio, which can produce C# code to reflect the visual manipulation, a platform independent manner is required for our Smart GUI Editor, which can be transformed easily into target code in the later phase of model transformation. As solution, we suggest *using OCL-init-expressions to represent the information generated by our Smart GUI Editor*.

As explained in [3], the OCL-init-expressions can be used to give the initial values of attributes or association ends of a class at the moment that an instance of this class is created. Hence, the original OCL-init-expressions are usually attached to the properties of a class as their context. In this paper, the syntax of the original OCL-init-expressions have been slightly extended such that all the automatically generated OCL-init-expressions are attached to

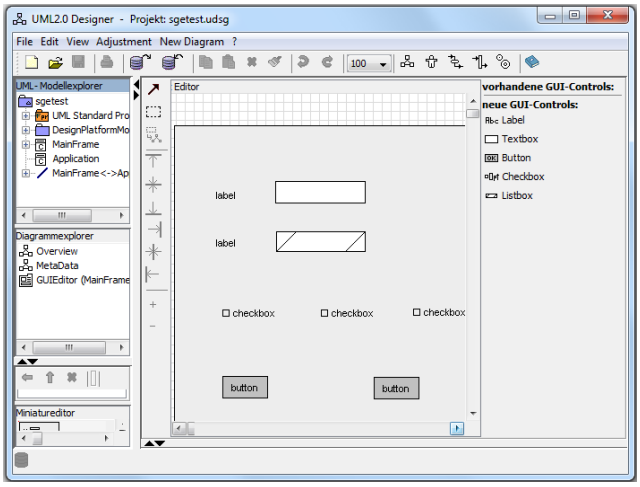


Figure 5. The Smart GUI Editor used to visually manipulate the GUI elements modeled in UML class diagram

the class directly, which is in our application a *GUI window*. According to our code-generation strategy, for initializing the properties of the GUI window itself, the Smart GUI Editor will generate an OCL-init-expression concatenating all the properties using OCL *and* operator. For each GUI-element contained in this window, the Smart GUI Editor will generate an additional OCL-init-expression specifying all its properties, again, connecting them by OCL *and* operator. The window instance can be retrieved by the OCL keyword *self*. Hence, the OCL-init-expressions in Listing 1 can be generated and attached to the class *EventHandlingComparisonWindow* as *constraints*. These OCL-init-expressions initialize the structural composition and geometrical information of the application window and its contained GUI elements *m_MainPanel*, which is a split panel, as well as the button *m_Button_1*. Other used GUI elements can be parameterized in the same way.

```
1  init: self.length = 480
2          and self.height = 629
3          and self.title = 'Main Window'
4
5  init: self.m_MainPanel.split = true
6          and self.m_MainPanel.horizontal = false
7          and self.m_MainPanel.owner = self
8
9  init: self.m_Button_1.posX = 35
10         and self.m_Button_1.poxY = 15
11         and self.m_Button_1.length = 93
12         and self.m_Button_1.height = 31
13         and self.m_Button_1.text = 'Button One'
14         and self.m_Button_1.owner = self.m_ButtonPanel
```

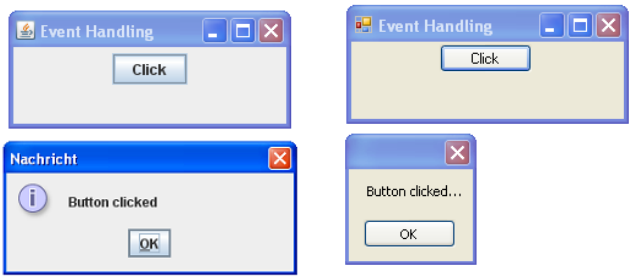
Listing 1. OCL–init–expressions to parameterize GUI elements

One more question must be answered. *How does the GUI-layout modeled in a PIM as suggested above make sense for the final GUI-layout mapped on a specific hardware platform?* In order to answer this question, we should make a difference between the GUI-layout and GUI-look-and-feel. For example, the both applications in Figure 1 do have the same GUI-layout but slightly different look-and-feels due to the underlying implementation platforms, say, Java-Swing and C#-FCL. Hence, modeling GUI-layout as suggested in this paper concentrates on the logical structure of an application window. That means which GUI-elements belong to which window, or to which panel etc. On the other hand, geometrical information can be modeled in a device independent coordinate system, which can be transformed onto concrete platform in the phase of model transformation by providing the model mapper with additional information about the underlying implementation platform.

III. OCL-EVENT-EXPRESSION – OUR NEW APPROACH TO MODEL EVENT HANDLING

Another important issue related to model GUI-based application in PIM is to model event handling [1]. It is difficult to deal with this modeling issue by using standard UML and OCL.

At first view, it seems to be possible to use the OCL *isSent* operator (denoted by ^) to model the coupling of events to their handling methods. However, the *isSent* operator can only be used in the post condition [3] of an event sending method, e.g., *fireActionPerformed()* operation of the class *JButton* in Java. It is needlessly for application modelers to specify post conditions of this type of operations, because it does not belong to the application model, but to the used GUI library. Hence, the *isSent* operator is not appropriate to connect events of GUI elements to their handling methods. To model the coupling of events to their handling methods using class diagrams in a traditional way is a serious problem, too. This will be illustrated by examples.



(a) Java Implementation (b) C# Implementation

Figure 6. A simple GUI-based application implemented in Java and C#

A very simple GUI-application, which has been implemented in both Java and C#, is shown in Figure 6. There is a single button in the main window of the application. When this button is pressed, a message box will be launched to confirm this operation. The same behavior occurs by pressing the closing symbol of the main window. Even for a simple application like this, the corresponding PSMs in Java and C# are not the same. Figure 7 shows both models.

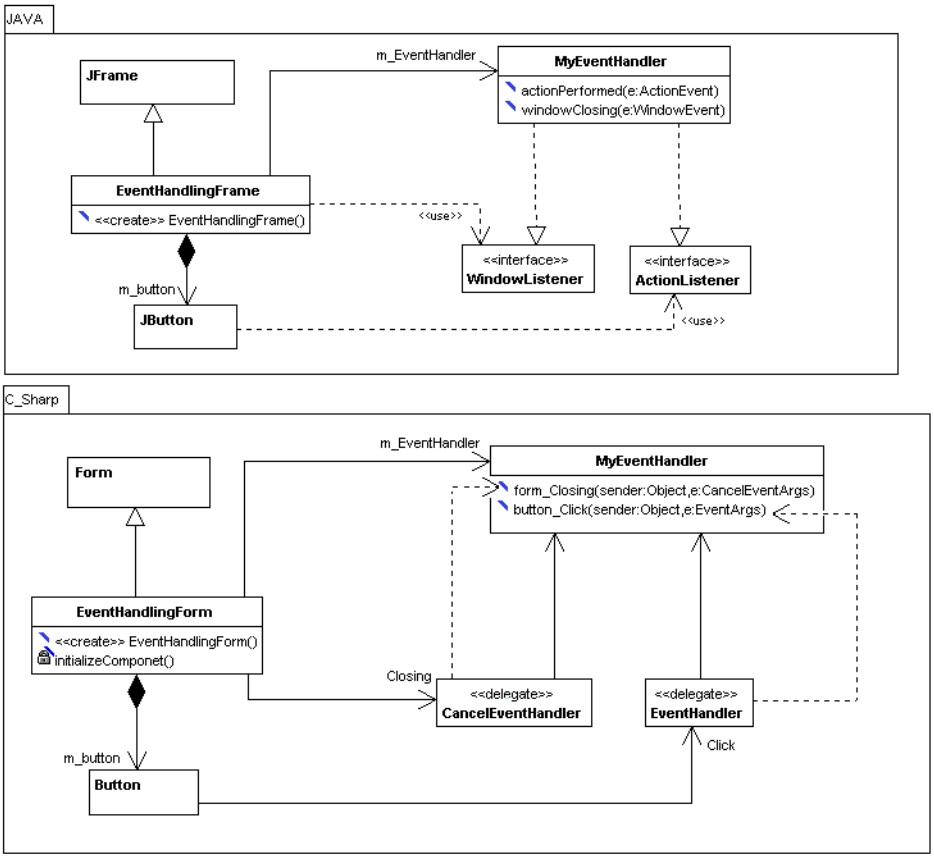


Figure 7. Java and C# models for the application of Figure 6

At first glance, these UML-models are similar to each other, because for each class and interface involved in the Java model, a counterpart can be found in the C# model. Only the number and types of the ”lines” between these elements are different. These differences are caused by the requirements of the languages. In Java, interfaces are used to connect events with their handler [12] while in C# delegates are used, which are in fact type safe callbacks based on function pointer [13]. The concept *delegate* is not supported by standard UML directly, so a stereotype must be defined in order to allow marking a class as delegate. In order to model the semantics of function pointers, two additional dependencies are used between the delegates and their pointed methods. The analysis of these UML models achieves an important conclusion: *the most complexities were brought into these models by modeling event handling in a too detailed manner.*

Taking into account the increased complexity of real GUI elements, modeling in such a way reduces the readability of class diagrams dramatically. The PSM for the Java implementation of the application in Figure 1 is shown in Figure 8. Due to poor readability, the corresponding C# PSM with increased complexity is not included in this paper. The Java PSM explores another remarkable drawback of traditional modeling of the event handling in class diagrams, which is: *only the three classes with colored background belong to the classes to develop; the other classes with white background are GUI-related library types*.

Due to these findings, a novel approach of modeling event handling in much simpler manner must be found. In this new approach, the tedious details explored in the PSMs must be hidden to the application modeler. The class diagram should contain as few library types as possible.

In order to find a unified and tight model for event-handling, a thorough understanding of the underlying event handling mechanisms is required. An event enables an object of a class (or a class itself) to publish changes of its state. Other objects and classes can then react to this change. This mechanism is usually called *Publishing – Subscription* model. Despite different implementations of this model in concrete programming languages, the entire event handling process can be divided into four parts [14]:

- *Static publishing* requires, that some kinds of events can be specified as members of their source. For example, events such as *Window Closing*, *Button Click* must be specified in GUI elements representing an application

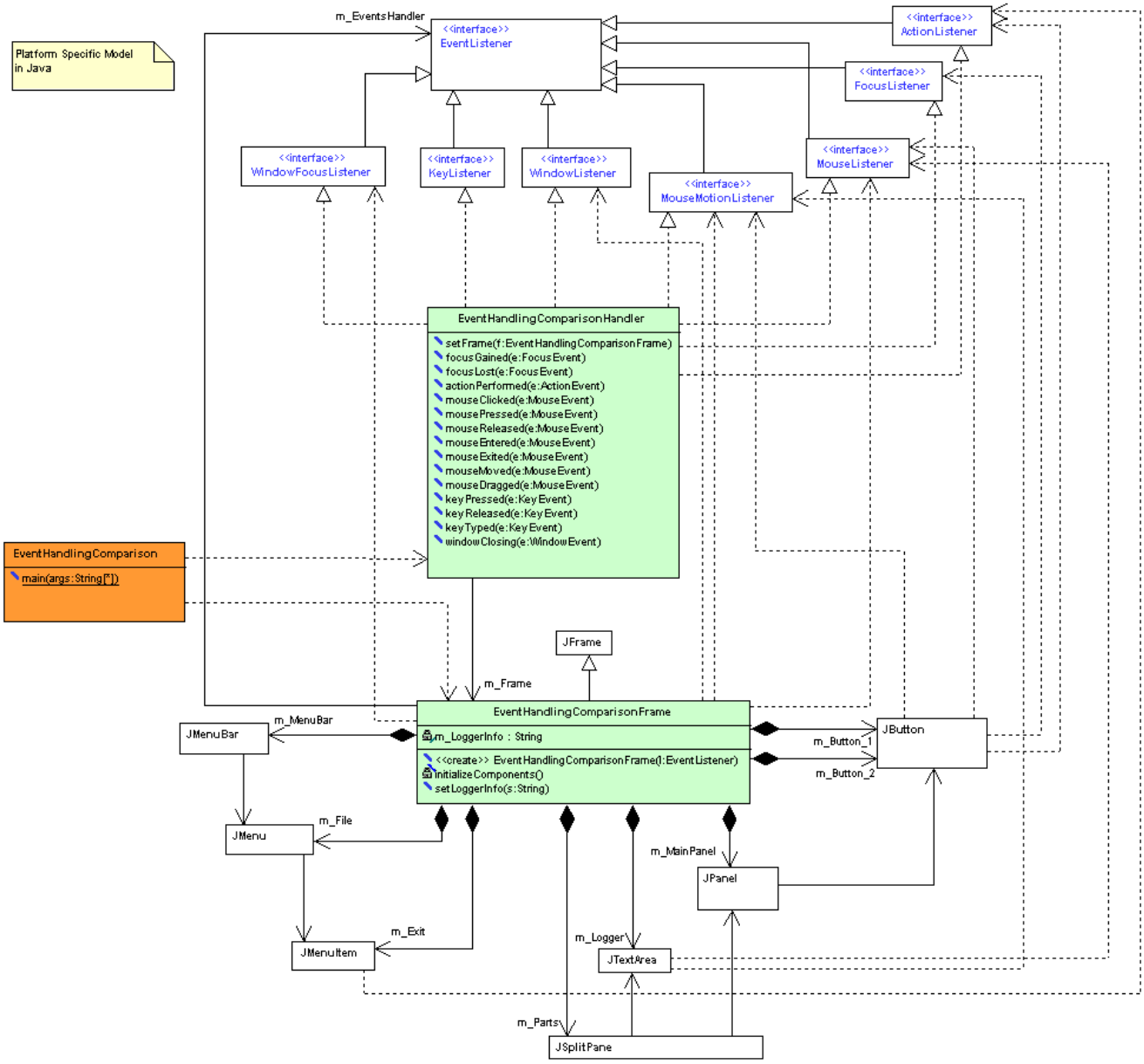


Figure 8. The PSM for the Java implementation of Figure 1

window or a button, respectively. This part is only important for customized events. The most significant GUI events have been defined by GUI developers.

- *Dynamic publishing* allows the transmission of the events. In both Java and C# this part is realized by a method, which triggers the execution of one or several dedicated event handling methods. Similar to static publishing, this part has been again implemented by GUI library designers.
- *Static subscription* requires the implementation of all event-handling methods. In fact, exactly this part specifies what must be performed when an event occurs. This part has to be modeled by application modeler. Along with dynamic publishing, this part belongs to behavioral specification of an application model and should be modeled compactly using some kind of high level action language, which will be addressed in next section.
- *Dynamic subscription* is done by establishing the connection between the event-source and the event-handling method. Such a process is often called registration of event handlers. Based on the analysis above,

modeling this part makes the class diagrams complex and heterogeneous, because various listener interfaces are used in Java to connect events with their handling methods loosely while C# delegates set up these connections directly. *The new approach discussed in this section is designed to simplify exactly this part of the entire event handling process*.

In order to develop a unified model for event-handler registration, a thorough comparison between Java and C# event models is completed to extract the similarities from them and to recognize the differences between them. The conclusion of this comparison can be summarized as follows:

1) In Java, the signature of an event-handling method is completely specified in one of the listener interfaces while C# *EventHandler* delegate (and its subtypes) determines only the parameter list and the type of the return value of possible event-handling methods.
2) In Java, the individual event cannot be referenced as a member of its source separately whereas C# supports it by using *event* key word to define each event as a separate member of its source object.
3) In both Java and C#, when an event occurs, certain additional information can be passed to the corresponding event-handling method. Subtypes of *EventObject* are used in Java to represent such information whereas there are EventArgs and its subtypes as counterparts in C#.
4) In Java, each invocation of one of the *addXXListener()* methods on an event-source can connect a group of related events with their corresponding handling methods implicitly, whereas the ”+=” operator of C# connects one single event with its handling method, explicitly.

Based on the comparisons above, the C# manner is more flexible and clearer in terms of expressiveness and it provides us a heuristic to develop a way, in which *dynamic subscription* in event-handling can be modeled uniformly for different platforms. All the essential elements involved in the *dynamic subscription* are:

- the *event-source object*, which are usually the GUI elements of a window or the window itself,
- different types of *event* of an event-source,
- *event-handling methods*, which are implemented in event-handler classes, and
- a *connection operator* that allows to set up the connection of an event to its event-handling method.

As solution to the problems mentioned above we suggest extending the OCL by a new expression, which is labeled by the keyword *event*. In such an *OCL-event-expression* the new registration operator ”∼” is used to establish the connection between an event on the left hand side and an event-handling method on the right hand side of this operator. We defined that the new OCL-event-expression in the above form has the type *OclVoid* and consequently no value.

Listing 2 shows the concrete syntax of the OCL-event-expression. Its abstract syntax is shown in Figure 9. An instance of *OCLEventExp* associates with two instances of the abstract syntax type *OCLFeatureCallExp* representing the event and its handling method respectively. The extended OCL abstract syntax will be discussed in next section more detailed.

```
1 <EventExpCS> ::= 'event' ':'
2 <OCLFeatureCallExpCS>'~'<OCLFeatureCallExpCS>
```

Listing 2. Grammar rule deriving OCL–event–expression

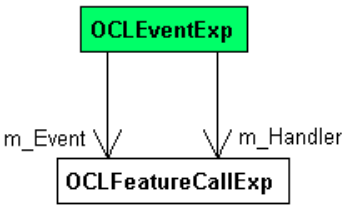


Figure 9. The abstract syntax of the OCL-event-expression

The class diagram in Figure 10 models the C# implementation in Figure 1. Compared with the Java PSM in Figure 8, this C# PSM is much simpler. Because the association ends to GUI elements have been modeled as normal properties, most the cumbersome ”lines” could be removed. The connections between the events and event-handling methods are modeled in a tight and well understandable manner using our new suggested OCL-event-expressions.

For example, to specify the method *control_Click()* as the handling method for the *Click* event of both the button *m_Button_1* and the button *m_Button_2* in the derived *Form* class, two expressions

event:
self.m_Button_1.Click∼m_EventHandler.control_Click
event:
self.m_Button_2.Click∼m_EventHandler.control_Click

can be written in the context *EventHandlingComparisonForm*.

As part of our new approach the OCL-event-expression allows to model *dynamic subscription* especially for GUI elements in class diagrams. Figure 10 shows that OCL-event-expressions lead to a very compact C# PSM.

The modified Java PSM is shown in Figure 11. Compared to the Java PSM in Figure 8, this model is both very compact and similar to the C# PSM in Figure 10. The two extended OCL expressions mean that the event-handling methods *button_1_actionPerformed()* and *button_1_mouseClicked()* are connected to their corresponding events of the button

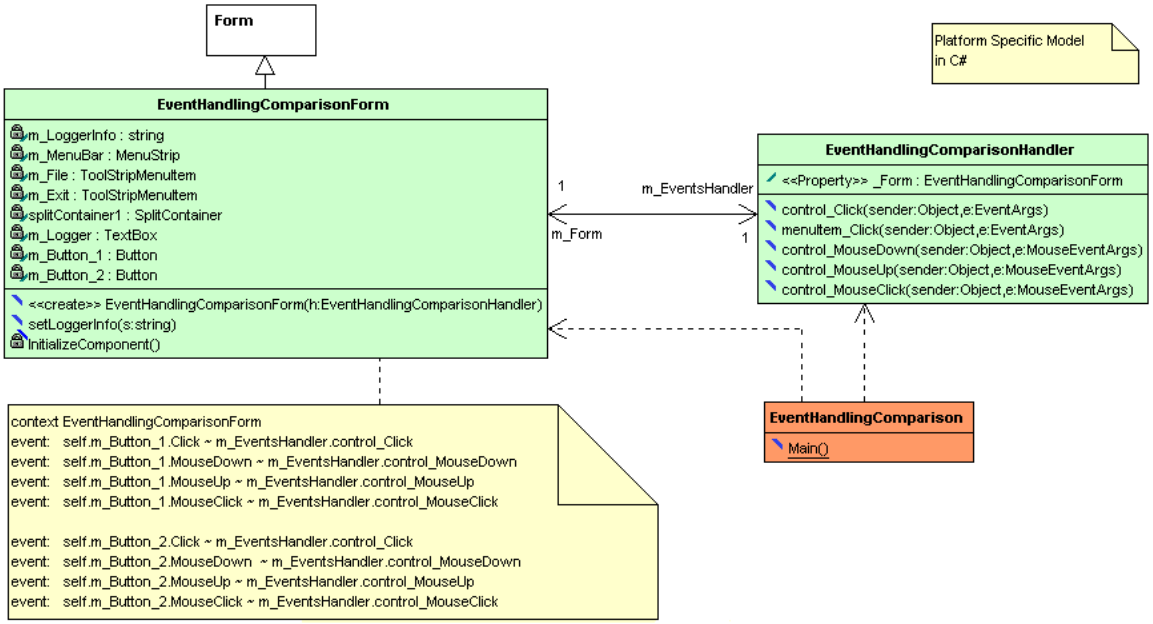


Figure 10. C# PSM using suggested OCL-event-expressions to model event-handler registration

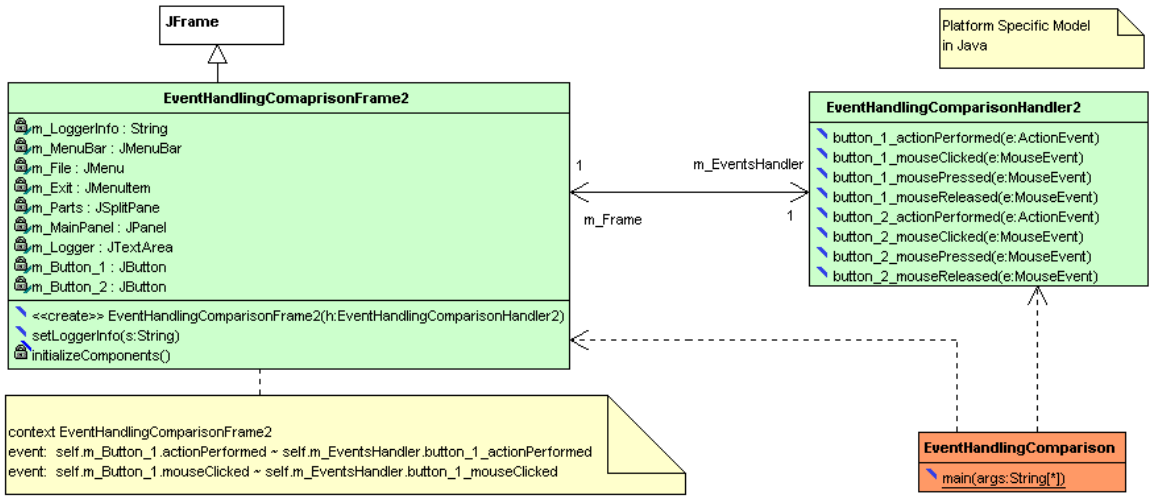


Figure 11. The modified Java PSM with our new approach

m_Button_1. Modeling in this way breaks the constraints in the Java event model in the following way.

- The event-handling methods can be declared as flexibly as in C#. Specifically, their names do not need to be pre-coded. Hence, it is not required any more that the event handler class implements the relevant listener interfaces. This is the solution for point one in the comparison given above.
- As solution of point two, an approach has to be found to identify a single event on an event source. An intuitive candidate may be a Java event object, e.g., *WindowEvent*, *MouseEvent* etc., but they are similar to their corresponding listener interfaces, which group several related events together. It requires an extra effort to select a single event of such a event-collection. As result of our detailed analysis, we found that it is possible to adopt the method name defined in the event listeners to identify a single event. If an event-source can register several event listeners, the method names in this set of listeners classify the events exactly.

Because the Java event-handling framework specifies that event-handling methods must connect to events via methods

named *addXXListener()*, only methods registered in this way can be used in the *dynamic publishing* phase of event handling. In order to overcome this restriction of connecting the event-handler with the event, a Java *anonymous class* [12] can be created as a bridge between the fixed method-name of a Java event-handling method and the free chosen name of the event-handler in the UML-model. Both expressions in Figure 11 can be transformed into the Java code as shown in Figure 12 either automatically by a model compiler or manually by Java programmer. For that, it is necessary to choose a correct listener interface or adapter class to declare the anonymous class. This is possible because the event-handling methods used to identify events have been declared or implemented in each listener interface or adapter class clearly. The matching between them is unambiguous.

```
this.m_Button_1.addActionListener(
    new ActionListener()
    {
     public void actionPerformed(ActionEvent e){
         m_EventsHandler.button_1_actionPeformed(e);
     }
    }
);
this.m_Button_1.addMouseListener(
    new MouseAdapter()
    {
        public void mouseClicked(MouseEvent e){
            m_EventsHandler.button_1_mouseClicked(e);
        }
    }
);
```

Figure 12. Java Code corresponding to the OCL-event-expressions of Figure 11

Although the names of GUI elements, event objects and even the concrete event-handling methods are different, the underlying PSMs are similar to each other both logically and visually. Even more, the reduced PSMs share the same logical structure with the PIM in Figure 2. Hence, we can extend the common GUI elements involved in the MOCCA DPM with common events. Then the OCL-event-expressions can be used the same way as OCL-init-expressions to parameterize GUI elements in order to model the registration of events to their handling methods platform independently.

IV. A Small Set of Additional OCL Expressions – our new Approach to Specify the Behavior of Platform Independent Models

Generally speaking, there are two possibilities to model behaviors platform independently. One of them is to use UML behavioral diagrams, e.g., state chart, activity diagram or sequence diagram. The other one means specifying behaviors in high level action language. Based on our experiences with MOCCA, in some circumstances, the behaviors modeled by, e.g., activity diagrams become more complex than the target code itself. So we suggest *modeling behaviors using an action language*. The actual situation seems a bit awkward, since the OMG has only specified the Action Semantics [5] without one standard surface language. Instead of defining a brand new OMG action-semantics-compliant action language, we find that the widely spread Object Constraint Language (OCL) seems closely to be a good action language due to the following reasons:

- OCL is a standard part of the UML 2 Specification. It has been widely used and proven its value.
- OCL covers large part of the entire Action Semantics of UML. Only the semantics involving changing the states in the model are missing. Such missing semantics can be easily added to the standard OCL with new syntax constructs. We introduce the required small set of additional OCL Expressions in this section.
- The OCL collection types and their predefined operations are powerful in terms of expressiveness and concise in terms of syntax. Together with OCL-body expression, very complex query operations in PIM can be specified both concisely and exactly.

Before we get into the technical issues, an example will be used to show the advantage of specifying complex query operations in OCL. Figure 13 shows a piece of class diagram modeling a payback management system. The query operation *selectPartnersHaveNoPointsInServ()* of the class *Payback* gathers all the program partners, which do not provide services awarding points to the customers. Listing 3 shows the OCL expression specifying the complex logic mentioned above, while Listing 4 shows its corresponding Java implementation. Evidently, the standard OCL expression can be used to specify complex querying operation both exactly and compactly. This is what needed in the modeling phase.

```
1 body: self.m_Partners->select(p:ProgramPartner |
2         p.m_DeliveredServices -> forAll(s:Service |
3         not s.m_PointsIn ))
```

Listing 3. Specifying complex querying operation in OCL

```
1  ArrayList<ProgramPartner> selectResult=
2        new ArrayList<ProgramPartner>();
3  Iterator<ProgramPartner> itr_0=
4        this.m_Partners.iterator();
5  while(itr_0.hasNext()){
6    ProgramPartner p= itr_0.next();
7    boolean forAllResult= true;
8    Iterator<Service> itr_1= p.getDeliveredServices().
         iterator();
9    while(itr_1.hasNext()){
10     Service s= itr_1.next();
11     forAllResult= forAllResult && !s.isPointsIn();
12   }
13   if(forAllResult)
14     selectResult.add(p);
15 }
16 return selectResult;
```

Listing 4. Java implementation of the OCL expression in Listing 3

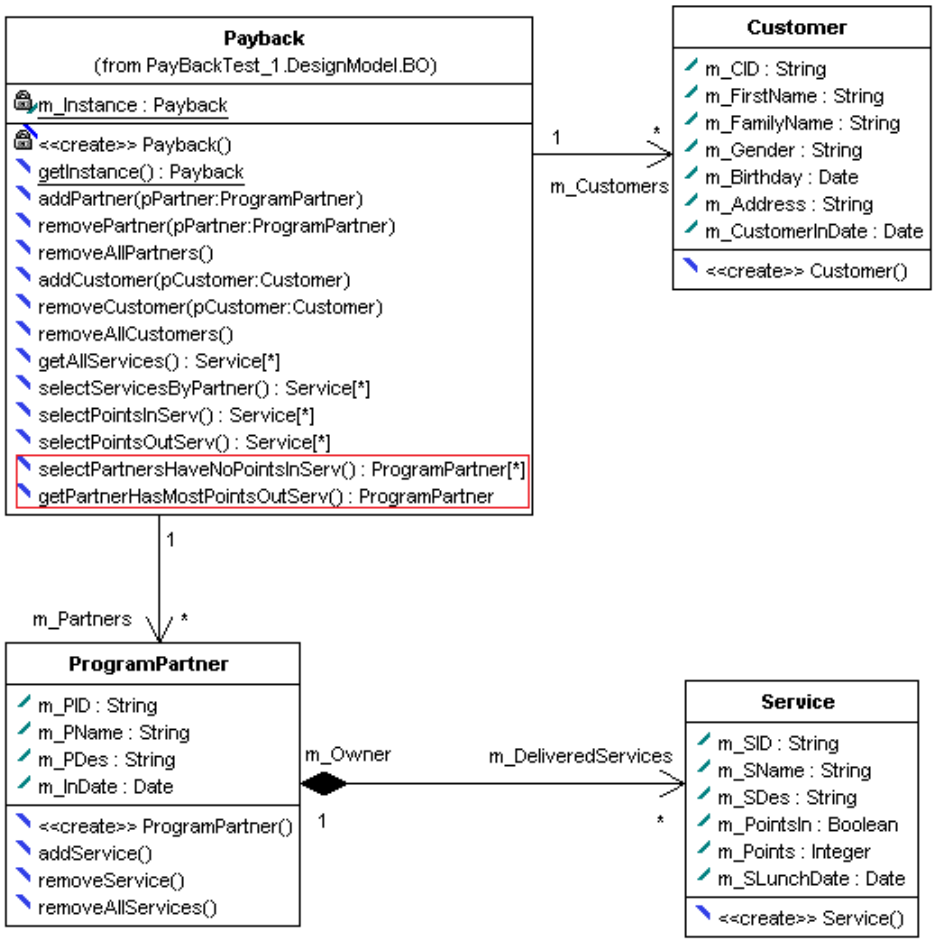


Figure 13. A part of the PIM modeling the business objects of a payback management system

UML supports modeling behaviors in arbitrary action languages inherently. Figure 14 (a) illustrates this support in UML meta-model. Every operation associates with its implementation represented by the abstract meta-class *Behavior*, which can be an activity, an interaction or an opaque behavior concretely. The opaque behavior is a behavior, whose semantics is determined by its body string, while the body can be specified in any kind of action language. Hence the opaque behavior can be used to carry (extended) OCL expressions at the time of modeling behaviors in PIM. This principle has been implemented in our UML 2 Designer. The opaque behavior containing OCL expression is created as a subordinate element of the operation, whose behavior has been specified by this expression.

As yet, the benefits of using standard OCL expression and its concrete usage in a tool have been illustrated. Towards upgrading OCL to a real action language, the next task is to select the important missing action semantics, add them to the standard OCL by defining new syntax constructs and extending the corresponding abstract syntax. We call the extended OCL XOCL, X means executable. Based on the achievement in [2] and the explanation in [5] as well as our research, the following actions, whose semantics are predefined in [5], are involved in the XOCL.

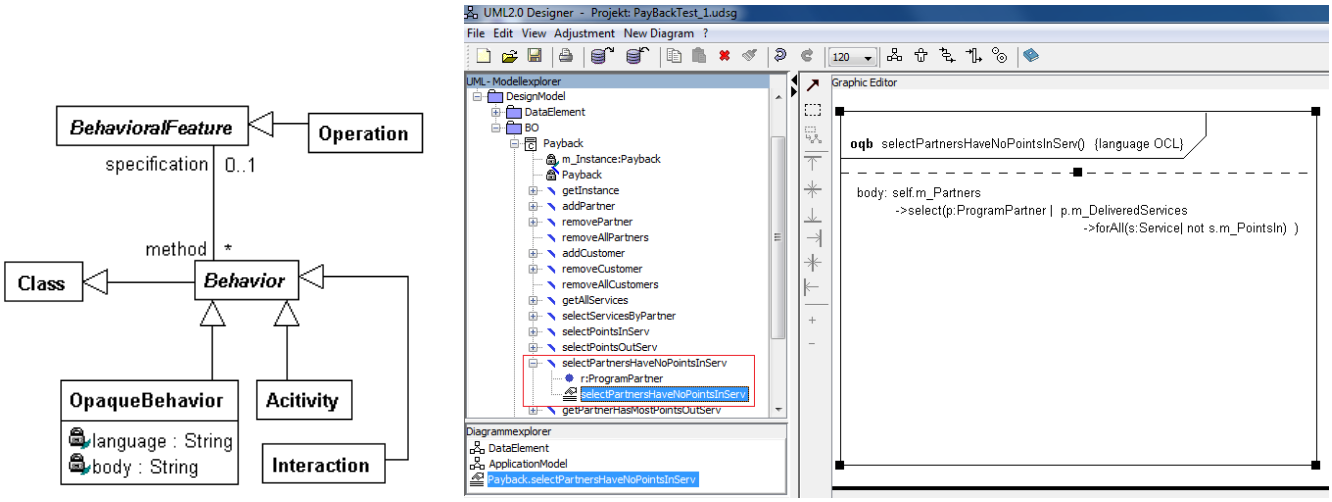
- *AddStructuralFeatureValueAction*,
- *RemoveStructuralFeatureValueAction*, and
- *ClearStructuralFeatureValueAction*:

These actions are important to update the state of an object, e.g., to assign the attributes of a class with new values. The syntax construct supporting them is specified by the grammar rule in Listing 5.

```
1 <PropertyAssignExpCS> ::= 'self' '.' ID ':=' <OCLExpCS>
```

Listing 5. Grammar rule deriving a property assignment expression

The notations of *GOLD Parsing System* [11] are used in this paper, because the GOLD Parsing System is used to implement an XOCL compiler for these extensions. According to GOLD, non-terminals are delimited by the angle



(a) UML meta-model of behavior (b) Supporting of opaque behavior in our own CASE tool - UML 2 Designer

Figure 14. Behavior as a meta-class in UML and its implementation in UML 2 Designer

brackets and terminals are delimited by single quotes or not delimited at all. As suggested in [4], each non-terminal has one synthesized attribute that holds the instance of the XOCL Abstract Syntax returned by the rule. For this rule, an instance of type *PropertyAssignExp* will be created with all information needed. Figure 15 shows all the types of XOCL Abstract Syntax. This inheritance hierarchy contains also the standard part of OCL Abstract Syntax, which is slightly different with the one specified in [4], in order to make the implementation efficiently.

- *AddVariableValueAction*,
- *RemoveVariableValueAction*, and
- *ClearVariableValueAction*:

These actions are used to assign and update local variables. The syntax construct supporting them is specified by the grammar rule in Listing 6. This rule returns an instance of the type *LocalVarAssignExp*. In fact, this rule can be involved in the rule of Listing 5. We make difference between them in order to keep type checking more efficiently, due to partition of symbol table into two parts.

```
1 <LocalVarAssignExpCS> ::= ID ':=' <OCLExpCS>
```

Listing 6. Grammar rule deriving a local variable assignment expression

- *CreateObjectAction*:

This action is used to create an instance of a class defined in UML model. Listing 7 shows its concrete syntax. An instance of the abstract syntax type *CreateObjectExp* will be created by this rule.

```
1 <CreateObjectExpCS> ::= 'new'
2       <OCLFullNameExpCS> '(' <OCLArgumentsCS> ')'
```

Listing 7. Grammar rule deriving a create object expression

- *DestroyObjectAcion*:

At the time of creating a PIM, the application logic cannot trust in the garbage collection system of the later target platform. So the feature of modeling destructing an object explicitly is provided by this action. Similar to other actions, Listing 8 shows its concrete syntax while *DestroyObjectExp* represents the abstract syntax.

```
1 <DestroyObjectExpCS> ::= 'delete' ID
```

Listing 8. Grammar rule deriving a destroy object expression

- *ReplyAction*:

The most imperative languages support this action with *return* keyword. We use the same keyword in XOCL. An instance of *ReplyExp* will be created to represent this action in abstract syntax.

```
1 <ReplyExpCS> ::= 'return' ID
```

Listing 9. Grammar rule deriving a reply expression

In order to make XOCL a complete action language, three elementary control flows, namely, sequential execution, conditional execution and iterative execution must be supported. In UML Action Semantics [5] they are represented by the meta-class *SequenceNode*, *ConditionalNode* and *LoopNode*. In OCL the conditional execution has been represented by OCL-if-expression; iterative execution has been hidden in the semantics of different *loop* operations, which are not appropriate for all the situations where the iteration semantics are needed; the explicit sequential execution are not supported at all. The extended syntax constructs supporting the elementary flow control semantics are added to the XOCL as follows:

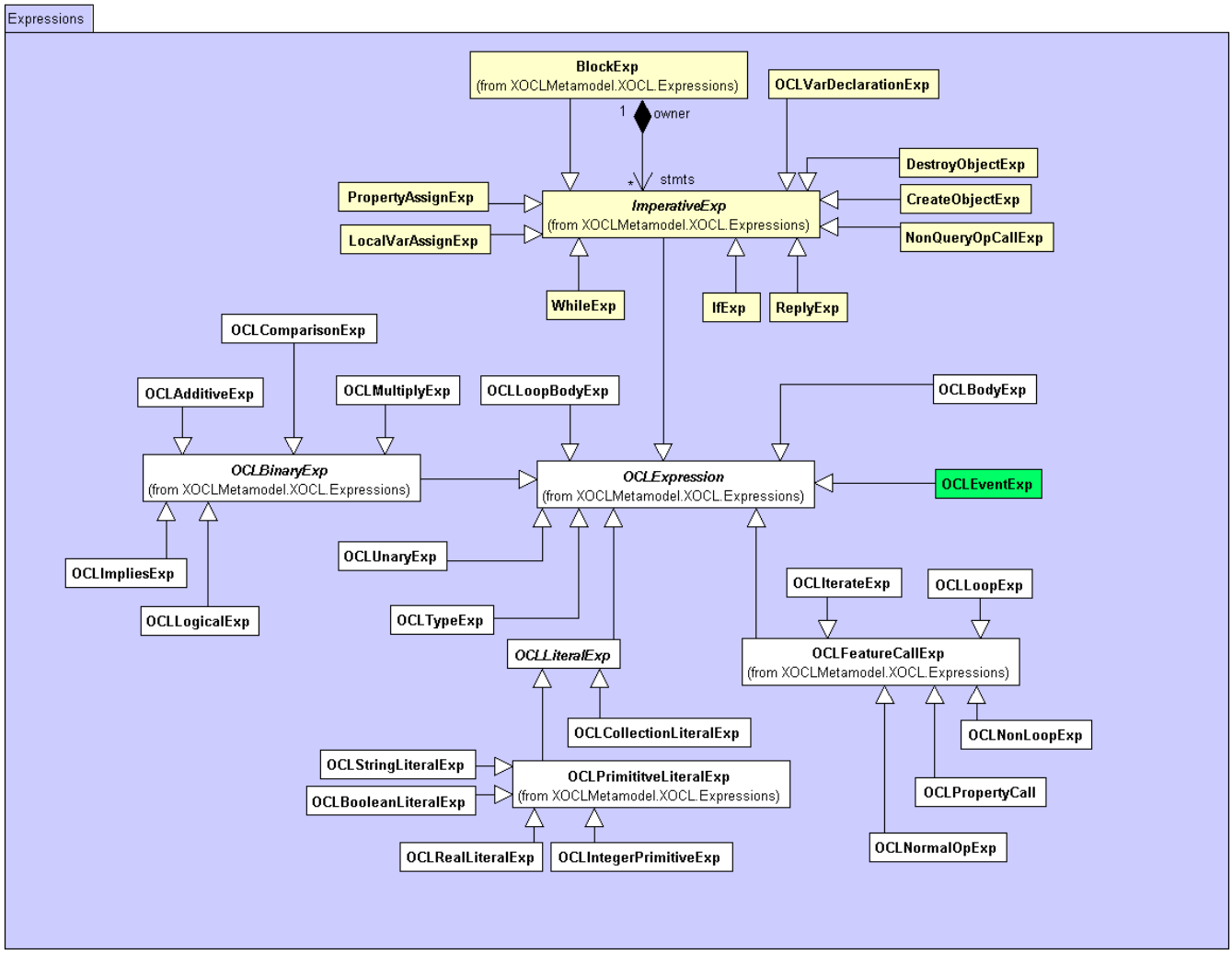


Figure 15. The inheritance relationships of XOCL abstract syntax

```
1  <BlockExpCS> ::= 'begin' <ImperativeExpList> 'end'
2                 | 'begin''end'
3
4  <ImperativeExpList> ::= <ImperativeExpList>
5                             <ImperativeExp>
6                         | <ImperativeExp>
7
8  <ImperativeExp> ::= <WhileExpCS>
9                    | <IfExpCS>
10                   | <OCLVarDeclarationExpCS>     ';'
11                   | <PropertyAssignExpCS>        ';'
12                   | <LocalVarAssignExpCS>        ';'
13                   | <DestroyObjectExpCS>         ';'
14                   | <ReplyExpCS>                 ';'
15                   | 'call' <OCLFeatureCallExpCS >';'
```

Listing 10. Grammar rules representing XOCL code block

- *Sequential Execution* is represented in XOCL by a block consisting other XOCL-expressions between the *begin* and *end* keywords. It can also be empty. Listing 10 shows all the grammar rules related to derive XOCL-block-expression. The alternatives in the body of the production for non-terminal *ImperativeExp* are the possible expressions, which can be used in an XOCL code block. The upper part of Figure 15 illustrates the corresponding abstract syntax of the concrete syntax in Listing 10. An instance of the abstract syntax type *BlockExp* serves as a container of other XOCL expressions. This can be recognized by the *composition pattern* in the class diagram of Figure 15.

```
1  <IfExpCS> ::= 'if' <OCLExpCS> 'then' <BockExpCS> 'endif'
2              | 'if' <OCLExpCS> 'then' <BockExpCS> 'else'
                   <BlockExpCS> 'endif'
3              | 'if' <OCLExpCS> 'then' <OCLExpCS>  'else'
                   <OCLExpCS>   'endif'
```

Listing 11. Grammar rules deriving XOCL conditional execution

- *Conditional Execution* can be specified in XOCL using *if*-expression. Listing 11 shows its syntax. The third

alternative represents the standard OCL-if-expression, the other two allow block expression to be used as body of an XOCL-if-expression. In abstract syntax, *IfExp* represents what is returned by these rules.

- *Iterative Execution* is represented by *while*-expression. Listing 12 shows its concrete syntax. The type *WhileExp* is used in the compilation to represent the abstract syntax of iterative execution.

```
1 <WhileExpCS> ::= 'while' <OCLExpCS> <BlockExpCS> '
      endwhile'
```

Listing 12. Grammar rule deriving XOCL iterative execution

The *CallOperationAction* has been involved in OCL. However, as explained in [3], only the query operations can be called in the normal OCL expressions. In XOCL non-query operations can also be called within an XOCL code block. The last alternative in the body of the production for the non-terminal *ImperativeExp* in Listing 10 makes difference between calling non-query operation in XOCL and calling query operations in original OCL by using a new keyword *call*. The non-query operation *call* is represented in the abstract syntax by the type *NonQueryOpCallExp* that functions as a wrapper of *OCLFeatureCallExp*.

Figure 15 illustrates the important inheritance relationships between the abstract syntax types. There are also important associations between these types as well as between them and the UML meta-classes. Instead of listing all the class diagrams modeling these relationships, an abstract syntax tree (AST) generated by our XOCL compiler after parsing the XOCL expression in Listing 3 will be used to illustrate both the associations between the types involved in the XOCL abstract syntax and the working principle of our XOCL compiler. Figure 16 shows this abstract syntax tree in the form of a *UML Object Diagram*. The both subtrees rooting in a *OCLLoopExp* represent the OCL *select()* and *forAll()* operation. At the runtime both instances of *OCLLoopExp* hold the information to identify the individual OCL loop operation. The *OCLFeatureCallExp* owns generally two branches; the one on the left-hand side is the caller while the other one is the callee. The callee usually uses the type information carried by the caller for type checking and code generation. Each object in yellow color represents a model element defined in the UML model. At the time of constructing an AST, each token, which may represent a model element, is type-checked against the symbol table, which is created based on the underlying UML model. If it is found, the corresponding model element will be associated with the AST node representing the actual token. For example, the AST node representing the token *m_Partners* has been linked with the association end *m_Partners* defined in the class diagram shown in Figure 13.

Since the abstract syntax tree stores all the type information that an XOCL expression involves, code generators for different platforms traverse each node in an AST, generate implementation codes on target platform by consulting the information modeled in the corresponding TPM. The construction of an AST for an XOCL expression happens in the phase of *Model Validation*, while code generation is done in *Model Transformation*. For example, after traversal the AST depicted in Figure 16, a *Java Code Generator* will emit the Java code shown in Listing 4 for the XOCL expression in Listing 3.

To conclude this section, we will use the extended language constructs of XOCL to specify another complex querying operation *getPartnerHasMostPointsOutServ()* of the class *Payback* in Figure 13. This operation gives back the program partner, who provides the most services that award points to their customers. Listing 13 shows the specification of this operation in XOCL. The new features of the XOCL allow software modelers to design their own implementation logics with the help of the existent OCL features on a higher abstraction level. The way of using XOCL is more or less similar to pseudo code used by mathematicians.

```
1  begin
2    resultPartner:ProgramPartner=self.m_Partners.first();
3    numberOfPointsOutServ:Integer=resultPartner.
         m_DeliveredServices->select(s:Service|s.
         m_IsPointsIn = false)->size();
4    index:Integer= 2;
5
6    while index<=self.m_Partners.size()
7    begin
8      num:Integer= self.m_Parnters.at(index).
           m_DeliveredServices->select(s:Service| s.
           m_IsPointsIn = false)->size();
9      if  num > numberOfPointsOutServ then
10     begin
11       numberOfPointsOutServ := num;
12       resultPartner :=  self.m_Parnters.at(index);
13     end
14     endif
15     index := index+1;
16   end
17   endwhile
18   return resultPartner;
19 end
```

Listing 13. XOCL expression specifying an operation

There is one more issue, which must be clarified before ending this section. Originally, the OCL was developed as a declarative language without side effect, which was primarily used in the UML models to specify constraints. After extending OCL into XOCL, the "purely declarative" characteristic is gone. Is this a problem? The answer is no. Firstly, the original OCL was completely contained in our XOCL as a subset. That means the OCL expressions can be used as usual to specify constraints as well as query operations. Secondly, the extended imperative language constructs are only required to specify non-query operations, which are excluded by the OCL. Finally, our language extensions are

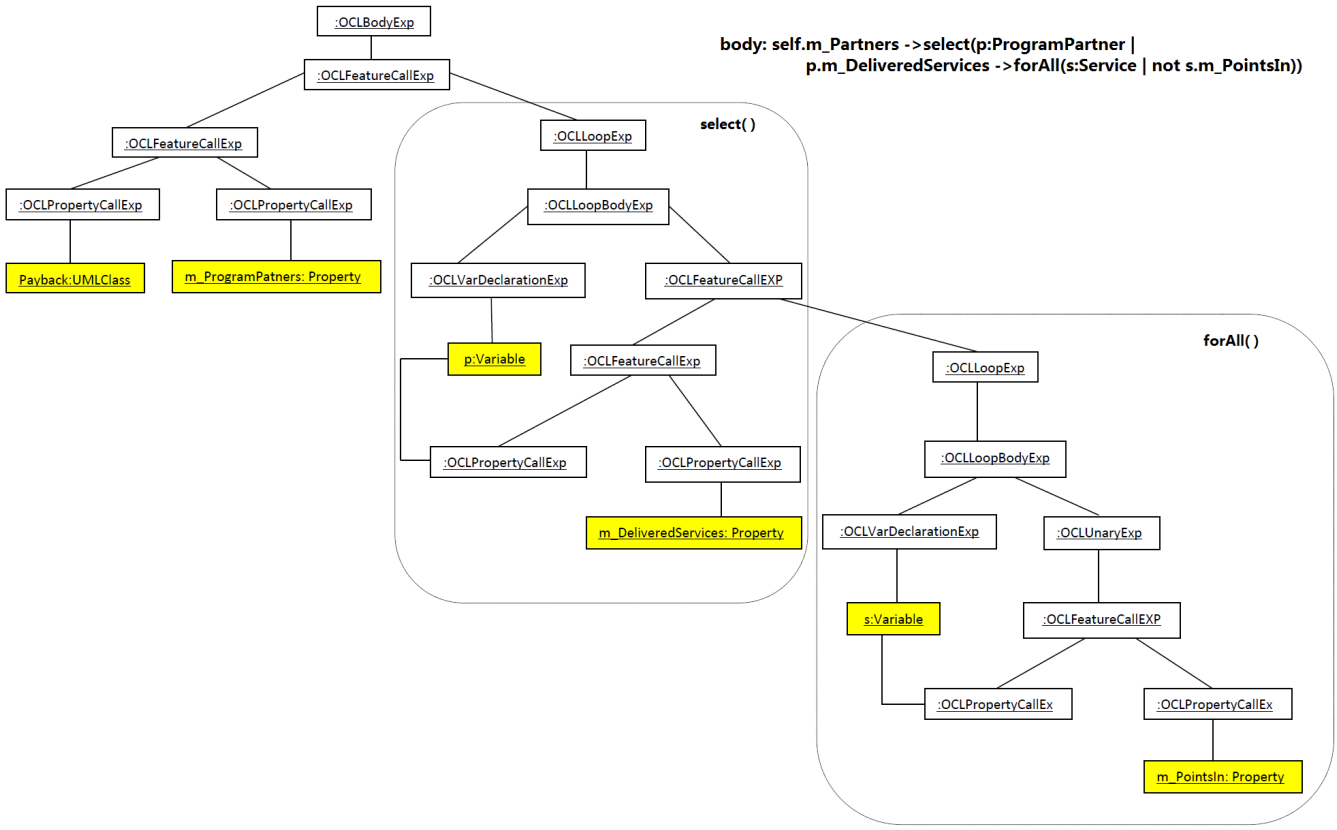


Figure 16. The abstract syntax tree of the XOCL expression in Listing 3

based on the OCL style, making its usage relatively easy.

An alternative to the slightly extended OCL into XOCL will be an extensive action language (AL), which must be defined from the scratch and must repeat many of the powerful data types and operations, which are already defined in OCL. In this unfavorable scenario three languages (UML + OCL + AL) are required to specify a complete model so that one main issue of models namely *simplicity* is not achieved.

V. Example of a Complete PIM

As an important experimental result to show the benefits, the platform independent model (PIM) of the GUI-based application of Figure 1 has been created using all three new approaches discussed in this paper. As Figure 1 illustrates, this application reacts to the events by means of logging them in the text area below the both buttons.

In the PIM created in this section, we concentrate on illustrating principles. Hence, only the *click*-events of the both buttons will be logged. The application will react to the *closing*-event of the main application window and the *click*-event of the menu item called *Exit* by means of disposing the held system resources explicitly. Figure 17 shows the class diagram of our example PIM. After manipulating the GUI elements in Smart GUI Editor, the OCL-init-expressions that parameterize them can be generated automatically and saved in a *constraint* attached to the class *EventHandlingComparisonWindow*. Listing 14 shows these generated OCL-init-expressions.

To register the important GUI-events to their corresponding handling methods, our OCL-event-expressions can be used. Listing 15 shows four examples. The other important events can be registered in the same manner. The abstract event-objects, such as *click*, *closing* have been modeled as properties of the corresponding GUI types in the MOCCA DPM.

The OCL-event-expressions must be used in the same *constraint*, which contains the OCL-init-expressions. After model transformation, all these (extended) OCL-expressions will be transformed into target language code and saved into the body of a special method called *initializeGUIComponents()*, which will be automatically generated and added to the class representing *EventHandlingComparisonWindow* on the target platform.

To model behaviors, or in other words, to specify the implementation of methods the XOCL-expressions are used as the body of an opaque behavior, which belongs to the method to be specified. To specify the implementation logic of the handling method for the event *Clos-*

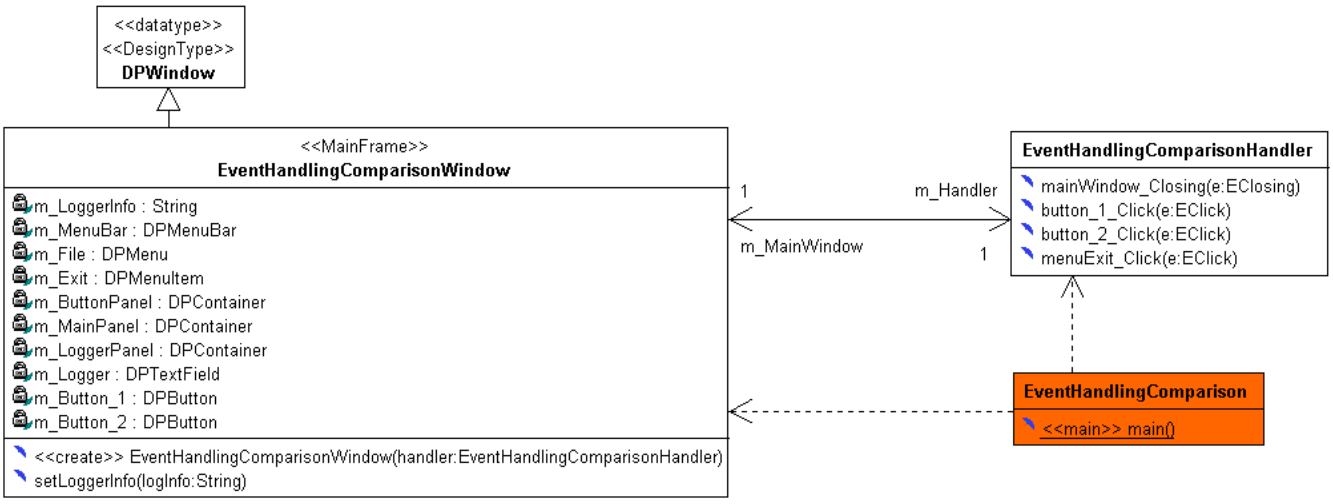


Figure 17. Platform independent model of the application of Figure 1

ing of the class *EventHandlingComparisonWindow*, XOCL-destroy-expression in Listing 16 is a good choice.

In model transformation, the semantics of destroying an application window will be translated into the target language constructs, e.g., *dispose()* together with *System.exit(0)* in Java. Listing 17 shows the same logic as Listing 16, because if the menu item *exit* is clicked, the entire application will be finished.

Listing 18 specifies the event handling method button_1_Click(). The platform independent semantics of the condition in *if*-expression can be understood as a test against null pointer in Java.

Listing 19 specifies the similar event handling method button_2_Click().

The XOCL-property-assignment-expressions in Listing 20 initialize the event handler as well as the logger string in the constructor of the class *EventHandlingComparisonWindow*. The identifier *handler* in Listing 20 is the formal parameter of the constructor.

In model transformation they are translated into target language constructs with the same semantics and additionally, the generated *initializeGUIComponents()* method will be appended in the transformed constructor to initialize the GUI elements as well as to register GUI events to their handling methods.

In Listing 21 the XOCL-create-expressions have been used to construct the necessary instances. Instead of using *main* method directly, we call such a unique method *start-up* method. Because no matter what it is called in PIM, the start-up method, which is identified by the stereotype *main* will be always transformed into the corresponding language construct on target platform.

As a review to the content discussed in Section IV, it should be understood that all the XOCL-expressions will be parsed into the abstract syntax trees like what is represented in Figure 16 in the phase *model validation*. These XOCL ASTs will be processed by consulting the corresponding TPM in the phase *model translation* to generate language construct on the target platform.

```
1  init: self.posX = 100 and self.posY=100 and
2        self.length = 500 and self.height = 800 and
3        self.title = 'Main Window'
4
5  init: self.m_MenuBar.owner = self
6
7  init: self.m_File.text = 'File' and
8        self.m_File.owner = self.m_MenuBar
9
10 init: self.m_Exit.text = 'Exit' and
11       self.m_Exit.owner = self.m_File
12
13 init: self.m_MainPanel.split = true and
14       self.m_MainPanel.horizontal = false and
15       self.m_MainPanel.owner = self
16
17 init: self.m_ButtonPanel.split = false and
18       self.m_ButtonPanel.owner = self.m_MainPanel
19
20 init: self.m_Button_1.posX = 35   and
21       self.m_Button_1.posY = 15   and
22       self.m_Button_1.length = 93 and
23       self.m_Button_1.height = 31 and
24       self.m_Button_1.text= 'Button One' and
25       self.m_Button_1.owner = self.m_ButtonPanel
26
27 init: self.m_Button_2.posX = 285  and
28       self.m_Button_2.posY = 15   and
29       self.m_Button_2.length = 93 and
30       self.m_Button_2.height = 31 and
31       self.m_Button_2.text= 'Button Two' and
32       self.m_Button_2.owner = self.m_ButtonPanel
33
34 init: self.m_LoggerPanel.split = false  and
35       self.m_LoggerPanel.owner = self.m_MainPanel
36
37 init: self.m_Logger.multiLines = true and
38       self.m_Logger.owner = self.m_LoggerPanel
```

Listing 14. OCL–init–expressions parameterizing GUI elements in Class EventHandlingComparisonWindow

```
event: self.closing ~ self.m_Handler.mainWindow_Closing
event: self.m_Exit.click ~ self.m_Handler.
    menuExit_Click
event: self.m_Button_1.click ~ self.m_Handler.
    button_1_Click
event: self.m_Button_2.click ~ self.m_Handler.
    button_2_Click
```

Listing 15. OCL–event–expressions registering events of the GUI elements in Class EventHandlingComparisonWindow to their handling methods

```
begin
  delete self.m_MainWindow;
end
```

Listing 16. XOCL–expression specifying the event handling method mainWindow_Closing()

```
begin
  delete self.m_MainWindow;
end
```

Listing 17. XOCL–expression specifying the event handling method menuExit_Click()

```
begin
  if not self.m_MainWindow.oclIsUndefined() then
  begin
    call self.m_MainWindow.setLoggerInfo('Button 1 was
        clicked!');
  end
  endif
end
```

Listing 18. XOCL–expression specifying the event handling method button_1_Click()

```
begin
  if not self.m_MainWindow.oclIsUndefined() then
  begin
    call self.m_MainWindow.setLoggerInfo('Button 2 was
        clicked!');
  end
  endif
end
```

Listing 19. XOCL–expression specifying the event handling method button_2_Click()

```
begin
  self.m_Handler := handler;
  self.m_LoggerInfo :='Logging the user actions...';
end
```

Listing 20. XOCL–expressions specifying the constructor of the class EventHandlingComparisonWindow

```
begin
  handler: EventHandlingComparisonHandler = new
      EventHandlingComparisonHandler();
  window: EventHandlingComparisonWindow = new
      EventHandlingComparisonWindow(handler);
  handler.m_MainWindow := window;
end
```

Listing 21. XOCL–expressions specifying the start–up method of the entire application

VI. Conclusion

The UML is a powerful language for preparing models of object oriented software. Such models can be used as documentation of existing software for their maintenance, and as source of new software to develop. In order to create precise models the OCL is used to specify the UML meta-model, and can be used for design models with the UML commonly.

The aim of the MDA technology is the generation of program code for a certain platform based on a complete UML/OCL model. In order to gain benefit from this innovative technology, it is necessary to create the basic platform independent model (PIM) concisely, uniformly, completely, and especially with low effort. Preparing such UML/OCL - PIMs we recognized three serious problems. In this paper we emphasize these problems and suggest efficient solutions, which base on UML and OCL.

1) How it is possible to describe both structural compositions and visual parameters of GUI elements?
   We suggest to use a normal UML-class of a window, to model the GUI elements as their attributes, and to specify the parameter values in OCL-init-expressions attached to their class-context. These OCL-init-expressions can be generated from a Smart GUI Editor based on the visual information.
2) How it is possible to model the connection between an event source and an event handler in a platform independent manner?
   We suggest a simple OCL extension. This single new OCL-event-expression allows to model the registration of handling methods to event sources in a more compact, well understandable, and uniform manner independent from the implementation platform. This new approach simplifies the class diagrams strongly without loss completeness. The application of our new OCL-event-expression leads to strong benefit in models of many GUI elements.
3) How it is possible to model the behavior in PIM both concisely, exactly, and compactly?
   We suggest a restricted extension of the OCL. The expressive, declarative OCL is upgraded into an imperative action language, which we called XOCL. With XOCL, complex query operations can be specified as usual while non-query operation with complex control flows can also be specified using the extended language constructs.

Using our three new approaches it is possible to create platform independent models efficiently, concisely, uniformly and completely.

References

[1] D. Liang and B. Steinbach, *A new General Approach to Model Event Handling*, ICSEA 2010, pp.14-19, 2010 Fifth

International Conference on Software Engineering Advances, 2010.

[2] K. Jiang, L. Zhang, and S. Miyake, *Using OCL in Executable UML*, MoDELS 2007.

[3] J. Warmer and A. Kleppe, *The Object Constraint Language, Getting Your Models Ready For MDA*, Addison-Wesley & Pearson Education, Boston, MA, USA, 2003.

[4] OMG: Object Constraint Language Specification 2.0, 2006. http://www.omg.org/spec/OCL/2.0/

[5] OMG: UML 2.4 Superstructure Specification, 2011. http://www.omg.org/spec/UML/2.4/Superstructure/Beta2/PDF/

[6] J. Warmer, A. Kleppe, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison-Wesley, 2003.

[7] OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, 2008. http://www.omg.org/spec/QVT/1.0/PDF/

[8] S. Nolte, *QVT Operational Mappings*, Springer, 2009.

[9] D. Fröhlich, *Object-Oriented Development for Reconfiguralbe Architectures*, Dissertation of Dr. Fröhlich, TU Freiberg, Germany, available July 2011. http://www.qucosa.de/fileadmin/data/qucosa/documents/2209 /InformatikFrXXhlichDominik80246.pdf

[10] B. Steinbach, D. Fröhlich, and T. Beierlein, *Hardware/Software Codesign of Reconfigurable Architectures Using UML*, UML for SOC Design, chapter 5, Springer, 2005.

[11] GOLD Parsing System Online Documentation, available July 2011. http://www.devincook.com/goldparser/index.htm

[12] C.S. Horstmann and G. Cornell, *Core Java*, 8th ed. Prentice Hall, 2008.

[13] A. Troelsen, *Pro C# 2008 and the .Net 3.5 Platform*, 4th ed. Apress, 2007.

[14] H. Keller and S. Krüger, *ABAP Objects*, 2nd ed. Galileo Press, 2007.

Analysis and Improvement of the Alignment between Business and Information System for Telecom Services

Jacques Simonin[1], Emmanuel Bertin[2], Yves Le Traon[3], Jean-Marc Jézéquel[4], Noël Crespi[5]

[1] Institut Télécom, Télécom Bretagne, Lab-STICC UMR CNRS 3192, UEB, Technopole Brest-Iroise, 29238 Brest, France

[2] Orange Labs, 42 rue des Coutures, 14066 Caen, France

[3] Université du Luxembourg, 6, rue Richard Coudenhove-Kalergi, 1359 Luxembourg, Luxembourg

[4] INRIA and Rennes University, Campus Universitaire de Beaulieu, 35042 Rennes, France

[5] Institut Télécom, Télécom SudParis, 9 Rue Charles Fourier, 91011 Evry, France

jacques.simonin@telecom-bretagne.eu, emmanuel.bertin@orange-ftgroup.com, yves.letraon@uni.lu, jezequel@irisa.fr, noel.crespi@it-sudparis.eu

Abstract **- The main aim of Enterprise Architecture (EA) is to master the development and the evolutions of Information Systems (IS). The EA process consists of designing the IS target architecture from several views, according to the company strategy. The business view represents the target organization of the particular company. The functional view focuses on the target functional architecture of that company's IS. In this paper, we propose a new formal solution to analyze and to improve the consistency between the target functional view and the target business view of telecom services. This solution is based on the definition of a strategic alignment of the target functional view with the target business view. Alignment is validated with a real case study implemented and deployed at Orange--France Telecom on their messaging service. An alignment measure completing this analysis provides an estimation of the gap between a target functional view and a target business view.**

Keywords - information system, enterprise architecture, business view, functional view, alignment, measure, function typing.

I. INTRODUCTION

A. Context and Motivation

Enterprise Architecture (EA) aims to simplify the Information Systems (IS) of a company, and to reduce the cost of IS development and evolution. This simplification of IS should be driven by the strategy of the company. For telecommunication service providers, the strategy mainly consists of providing new services (designed by marketing to fit user's needs) that rely as much as possible on existing infrastructures [1].

EA frameworks (such as that of Zachman [2]) define various points of view (business, system, technology, etc.) in order to take into account all the aspects of these strategic objectives. This paper relies on the four classic EA views (as defined in [3]): the business view defining 'why', the functional view defining 'what', the technical view defining 'with what', and the applicative view defining 'how'. The relationships between the functional view, the technical view, and the applicative view are deduced from the iterative development cycle, which relies on the Unified Process (UP) [4]. The business view should be an input for both the functional and the technical views.

This paper is focused on the strategic alignment of a company's functional view with its business view. A good alignment highlights the consistency within the organization of the company and its IS [5] and indicates that the business strategy and the IS strategy are synchronized.

The target business architecture and the target functional architecture must both fulfil the strategy of a company. However, the strategy guiding the business organization (business view) and that of the IS functions (functional view) are different and are not defined by the same people. Business and functional views evolve independently, following the business and the marketing evolutions. Moreover, the evolution of a company's organization is seldom synchronous with the evolution of its IS.

We therefore propose an innovative formal approach that allows a functional Enterprise Architecture to analyze the misalignment between the target functional architecture and the target business architecture. We also propose a metric for this alignment. The objective is to define an assessment in order to improve the alignment between the functional view and the business view.

B. Outline

This paper is organized as follows. Section II depicts the state of the art and Section III introduces the EA process and defines what is meant by the alignment of the functional view with the business view. Section IV describes the alignment measure of the functional view with that of the business view. The example in Section III and in Section IV is based on an Orange messaging service. Section V describes a solution to improve the alignment measure of the functional view in comparison with the business view, by typing functions according to business processes. Section VI depicts the first experimentation of the alignment measure at the Orange laboratories.

II. Related Work

In practice, most telecommunication companies directly map their business view with their IS applicative, aiming for an alignment between their core business and their IS. A company may decide that a given email platform (for example, Microsoft Exchange server) will manage the entire service business process of communicating by email. For telecom services, this method has one major shortcoming: it implies a tight coupling between the business view and the applicative view. The business analysis is then distorted by applicative considerations. For example, the messaging business may evolve so that it is driven by the evolutions of the selected platform, and no longer by the company strategy. The specification of a target business architecture that differs from the current applicative view may become virtually impossible.

In the literature, the alignment problem involving EA is mainly considered to be between the business view of a company and its IS [6]. Alignment may also be considered between the business view of a company and its objectives, as in the Business Motivation Model [7], or between an analysis model and a design model of the functional view of a telecom service [8]. The parameters related to the quality of the alignment are specific for each company [9]. For this type of alignment, heuristics may be defined to provide warnings in case of misalignment [5]. A measurement method would allow the evaluation of architectures in business terms (cost, benefit, risk). However, the measures relevant to business terms do not take IS concepts into account.

One significant contribution of this paper is a method to take into account the effects of a company's strategy on its functional view. The alignment perspective between the business and functional views is shown in Figure 1.

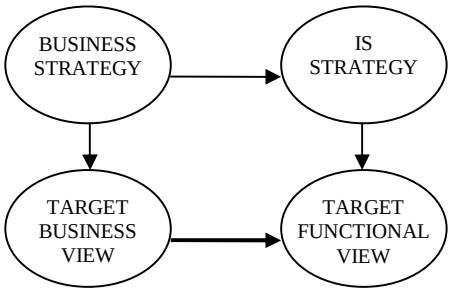


Figure 1. Alignment perspective between the target business view deduced from a business strategy and the IS target functional view deduced from an IS strategy [6].

The functional view choice is justified because an IS functional view is easier to align with than a business view. Functional, meaning comprehensible and practical, is indeed helpful to match business description. Moreover, the IS applicative view is largely the facet that implements the functional view. Alignment of the applicative view with the business view is thus dependent on the alignment of the functional view with the business view.

Moreover, many object-oriented measures exist outside the scope of EA. To estimate model alignment, coupling measures [10] are the most appropriate means, since relationships between models are the main characteristics of the solution proposed in this paper.

III. Alignment of the Functional View with the Business View Definition

We focus here on the alignment--or the misalignment of the functional view with the business view. As demonstrated in the previous section, this topic has not been studied in much detail.

A. Alignment Definition Home Domain Analogy

Let us introduce the alignment definition with a "home domain" analogy. In a "home domain", the house customer is responsible for the "home processes" of a home domain business view. One is the *Have a meal at home* process and its activities *Cooking at home* and *Eating at home*. The house occupant defines that *Eating at home* comes after *Cooking at home*. "Home IS" is designed by house architects, who define models of homes based on the home domain functional view. House architects design models of homes from "home functions". Three components composed of "home functions" and their dependencies support the *Have a meal at home* process. The so-restricted "Home IS" is represented by a home model in a Unified Modelling Language (UML) [11] component diagram given in Figure 2. Figure 2 illustrates three dependencies between "home functions": *Have breakfast* on *Cook at home*, *Have diner* on *Cook at home* and *Have lunch* on *Cook at home*.

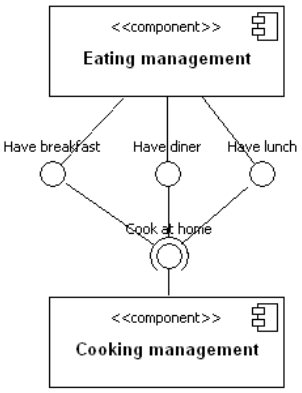


Figure 2. "Home domain" analogy with "Home IS", which owns two functional components: *Eating management* and *Cooking management.*

Alignment of the functional view with the business view is easily illustrated with this "home domain":

- *Have breakfast*, *Have diner*, *Have lunch* are "home functions" aligned with an *Eating at home* "home activity";
- *Cook at home* and *Get food* are "home functions" aligned with a *Cooking at home* "home activity",

- *Have breakfast* on *Cook at home*, *Have diner* on *Cook at home* and *Have lunch* on *Cook at home* are dependencies between "home functions" aligned with *Cook at home*, and come before any *Eating at home* activity succession.

With this home analogy, we can highlight that there is a reversal between the dependency and the succession relationships. The eating-on-cooking dependency means that cooking must come before eating.

B. EA and Target Architecture

The EA process has two main goals:

- to depict existing IT architecture, in order to describe what functions are implemented on each IT system, how each IT system is deployed and which process is supported by each IT system; and
- to design several target architecture views to separate the concerns of the various stakeholders in the enterprise.

Even if the company strategy is constant during the design of all the target architectures of these views, the required skills are different: on one side, the company's core business experts elaborate the business view; on the other side, enterprising functional architects design the functional view. This independency is particularly significant for the evolution of each view because their lifecycles are different. A complete synchronization of a company's organization evolutions and IS evolutions is very difficult to achieve for a large company. This is especially true for telecom service operators whose markets and technologies are very dynamic.

A company usually elaborates its target business architecture following a process analysis, which provides for descriptions of the business processes that belong to the core business of a company. The business view has the **activity** as its main concept**,** which is a part of a business process and which is under the responsibility of an organizational role. Concepts are modelled with UML [11]. A UML activity diagram can be used to capture a procedure designed in the target business architecture. Within Orange--France Telecom, the usage of telecom services is specified with approximately 10 roles and several tens of activities.

For illustration, let us consider a messaging service, limited to the message receipt. When a new requirement appears in the telecom operator strategy, such as the need to protect children from inappropriate electronic messages, the access control must evolve. In this example, the operator chooses to implement its strategy by creating a new *Child protection provider* role. Furthermore, the *Messaging service provider* role will depend on the new *Child protection provide*r role in the new target organizational infrastructure. So, to achieve the messaging receipt activity, the *Messaging service provider* role needs the intervention of the *Child protection provider* role.

The procedure deduced from the messaging service process is therefore easily captured using an activity diagram such as the one in Figure 3.

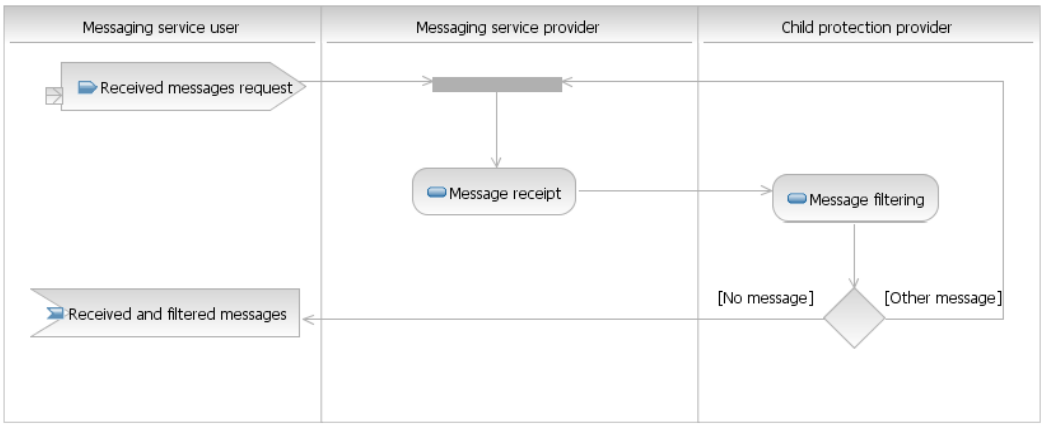


Figure 3. Sample activity diagram of the messaging service..

The IS target functional architecture contains functional elements implemented by IS systems. Functional architectures design the target functional architecture according to the company strategy. The main concept of the functional view meta-model holds that the **function** defines the functional component. Functional view concepts such as "Functional component" and "Dependency between functional components" are also close to UML concepts. The target functional view may be represented by a component diagram. The target functional view of our messaging illustration is represented by the component diagram in Figure 4.

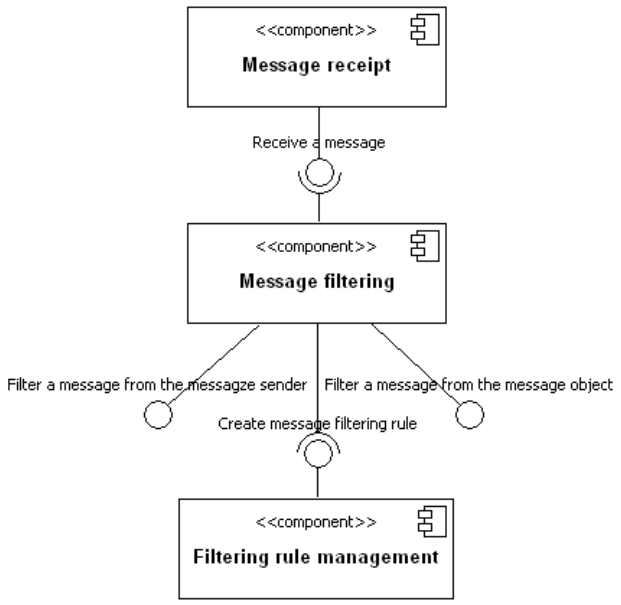


Figure 4. Sample target functional architecture of the messaging service.

C. Alignment of the Functional View with the Business View Definition

Alignment criteria are required to define the alignment between models. Our innovative criterion is based on the associations between concepts of the business view and of the functional view. Enterprise Architecture chooses business view concepts consistent with functional view concepts, while taking the alignment definition into account. The consistency between these concepts is known as the alignment value between a business model and a functional model.

We have considered two relevant types of possible associations:

- associations between business data manipulated by business activities and functional data manipulated by functions; and
- associations between business activities and functions.

An approach based on the first type (business data and functional data) can be viewed as static because it does not take into account the evolution of the states of data.

We have therefore chosen an innovative approach by considering the second type (business activities and functions). We have qualified this approach as dynamic, because it relies on the UML dynamic diagrams (activity diagram, sequence diagram) that show the live comportment of a system.

The idea of a dynamic approach (as opposed to an approach based on data) is to base the alignment on service usage scenarios instead of basing it on data models. For development methods in relation to the entity relationship model [12], the methodological complexity is a consequence of the simultaneous modelling of data and treatments. To resolve this complexity, our approach is based on the dynamic perspective because it allows functional reusability, a useful improvement. This reusability involves service components called enablers, as defined by the OMA (Open Mobile Alliance) [13]. Moreover, the alignment between business data and functional data can be deduced from the alignment between business activities and functions, as business data are produced by business activities and functional data by functions.

The business view, as illustrated in Figure 3, instantiates dynamic concepts. A procedure is indeed described by an activity sequence instead of a business data model. Concerning the functional view, the design of an interaction sequence carrying out a telecom service usage scenario does precede the data modelling. This chaining is feasible because each data is produced or used by a function during a scenario. With this dynamic approach, a dependency between functional components corresponds to an interaction between two functional component instances. The equivalence between an interaction sequence and a telecom service usage scenario denotes the dynamic aspect of this approach (see Figure 5).

A "request" type dependency of the functional view is therefore an information request. A functional dependency has a "resource" type if it represents an answer to an information request.

The association completing the alignment criterion is between a succession relationship of two business activities and a dependency between functions. We define the following links:

- Succession relationships are between two business activities if the end of one precedes the beginning of the other in a UML activity diagram capturing a business process (for example, in Figure 3, the succession relationship is the one from the business activity Message receipt to the business activity Message filtering);
- Dependency between two functions occurs
    - if they are associated to two interactions between functional components, which have either the "request" type or the "resource" type,
    - and if the end of one of these interactions precedes the beginning of the other interaction in a UML sequence diagram

  (an example in Figure 5 is the dependency between the function *Filter a message from the message sender* on the function *Create message filtering rule*).

The alignment of the functional view with the business view can thus be defined from these alignment criteria:

- a function is aligned with the business view
    - if the function has a common meaning with at least one activity of the business view,
    - and if each business activity aligned with the function has at least one succession relationship with another business activity aligned with the function ; and

- a dependency between two functions F1 and F2, such as F1 depends on F2, is aligned with the business view
    - if there is at least one business activity A1 aligned with F1,
    - if there is at least one business activity A2 aligned with F2, and
    - if A1 follows A2 in an activity diagram (succession relationship).

The alignment definitions are illustrated in Figure 3 for the business activity and the activity succession relationship concepts, in Figure 4 for the function concept, and in Figure 5 for the function dependency concept:

- *Receive a message* function is aligned with
    - *Message receipt* business activity;
- *Filter a message from the message sender* and *Filter a message from the message object* functions are aligned with
    - *Message filtering* business activity; and
- the dependency relationship from *Filter a message from the message sender* function on *Receive a message* function is aligned with
    - the succession relationship from the *Message receipt* business activity to the *Message filtering* business activity.

In Figure 5, the *Create message filtering rule* function and the dependency relationship of the *Filter a message from the message sender* function on the *Create message filtering rule* function are not aligned with the business view. No business activity in Figure 3 has a meaning in common with *Create message filtering rule* function.

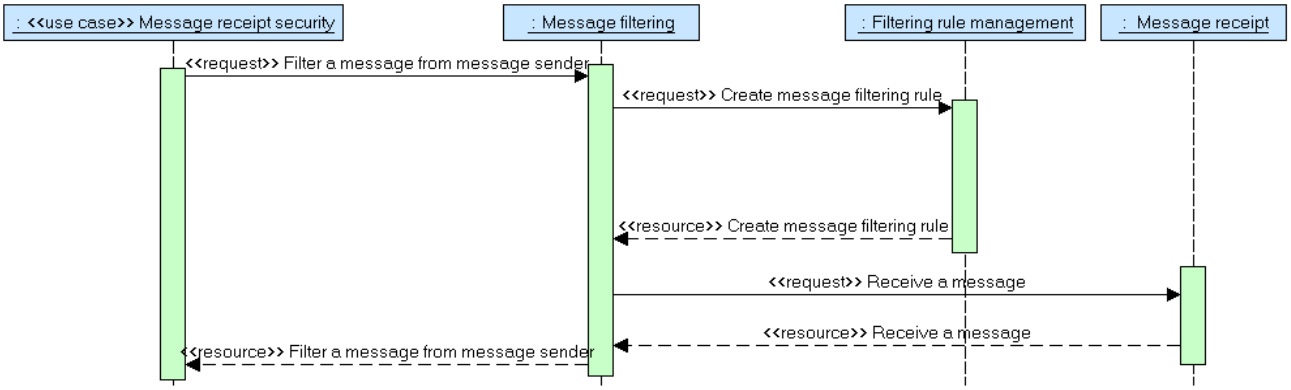


Figure 5. Functional view sequence of messaging service example.

IV. Alignment Measure of the Functional View with the Business View

Axiomatization allows the intuitive properties of the alignment of a functional view to be specified in comparison to a business view description [14]. We propose an alignment measure according to these axioms.

A. Alignment Measure Home Domain Analogy

Alignment measure could be illustrated with a "home domain" analogy. We assume in this section that "home function" *Get food* is offered by the *Cooking management* component. This "home function" is added to our "Home IS" designed in Section III. Figure 6 represents our new "Home IS" with a UML component diagram.

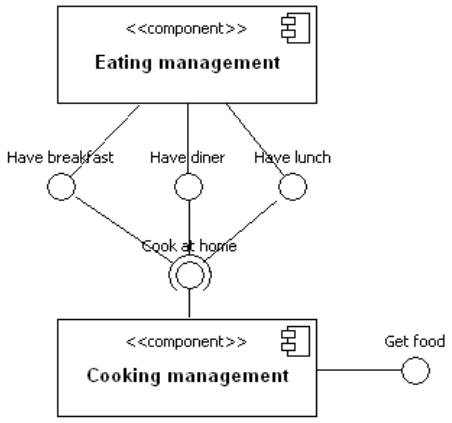


Figure 6. New "Home IS", which owns a new "home function": *Get food*.

This new "home function" *Get food* is not aligned with the "home activity" of *Eating at home* process because no activity of "Home processes" restricted to *Have a meal at home* shares a sense with food management, which is the closest repository management activity. We can say that the alignment of the new "Home IS" with "Home process" is worse than the alignment of the "Home IS" designed in Section III with "Home process".

B. Alignment of the Functional View with the Business View Axiomatization

An axiom is an expected and understandable property of an alignment measurement that has also a meaning in the mathematical model. The following BFA axioms define this intuitive behaviour. Axioms are parameterized by functional view concepts affected by the alignment.

BFA1 – Function addition. The alignment resulting from the addition of a function in the functional view is either:

- worse than or identical to the previous alignment if the function has no business meaning in common with at least one activity of the business view,
- better than the previous alignment if the function has a business meaning in common with at least one activity of the business view.

BFA2 – Function dependency addition. The alignment resulting from the addition of a dependency between functions in the functional view is either:

- worse than or identical to the previous alignment if there is no business activity time succession of the business view that is aligned with the function dependency,
- better than the previous alignment if there is at least one business activity time succession of the business view that is aligned with the function dependency.

BFA3 – Function deletion. The alignment resulting from the deletion of a function in the functional view is either:

- worse than the previous alignment if the function shares a common business meaning with at least one activity of the business view,
- better than or identical to the previous alignment if the function has no common business meaning with even one activity of the business view.

BFA4 – Function dependency deletion. The alignment resulting from the deletion of a dependency between functions in the functional view is either:

- worse than the previous alignment if there is at least one business activity time succession of the business view that is aligned with the function dependency,
- better than or identical to the previous alignment if there is no business activity time succession of the business view aligned with the function dependency.

C. Alignment of the Functional View with the Business View Measure

An alignment measure depends on the alignment concepts defined in Section III. The number of relationships captured in a diagram is a well-known parameter for data model estimation [15]. The dependencies from the target functional view are the parameters of a proposed alignment measure, named **BFAM**, of the functional view with the business view. These dependencies, which are or are not aligned with the business view, enable estimation of the alignment of the functional view with the business view.

$$BFAM(FV)=\left(\frac{N\_f(FV)-N\_\{naf\}(FV)}{N\_f(FV)}\right)*\left(\frac{N\_d(FV)-N\_\{nad\}(FV)}{N\_d(FV)}\right) \quad (1)$$

where, for a functional view FV:

- N_f(FV) is the number of functions,
- N_{naf}(FV) is the number of functions that are not aligned with business activities,
- N_d(FV) is the number of dependencies between functions,
- N_{nad}(FV) is the number of dependencies between functions that are not aligned with a business activity time succession of the business view.

The BFAM value is a real number which value is between 0 (no function and no dependency relationship between functions are aligned with the business view) and 1 for a perfect alignment (all functions and all dependency relationships between functions are aligned with the business view).

BFAM measures complies with axioms **BFA1**, **BFA2**, **BFA3**, **BFA4** (see Section IV) of the alignment of the functional view with the business view.

The compliance for the $\mathbf{BFA_1}$ axiom is detailed as follows:

- Let F a function added to the functional view FV,

$$BFAM(FV\cup\{F\})=\left(\frac{N\_f(FV\cup\{F\})-N\_\{naf\}(FV\cup\{F\})}{N\_f(FV\cup\{F\})}\right)*\left(\frac{N\_d(FV\cup\{F\})-N\_\{nad\}(FV\cup\{F\})}{N\_d(FV\cup\{F\})}\right)=\left(\frac{N\_f(FV)+1}{N\_f(FV)+1}-\frac{N\_\{naf\}(FV)+N\_\{naf\}(\{F\})}{N\_f(FV)+1}\right)*\left(\frac{N\_d(FV)-N\_\{nad\}(FV)}{N\_d(FV)}\right)$$

because the number of dependency relationships between functions is unchanged. And then

$$\Rightarrow BFAM(FV \cup \{F\}) - BFAM(FV) = \left(\left(1 - \frac{N\_\{naf\}(FV) + N\_\{naf\}(\{F\})}{N\_f(FV)+1}\right) - \left(1 - \frac{N\_\{naf\}(FV)}{N\_f(FV)}\right)\right) * \left(\frac{N\_d(FV) - N\_\{nad\}(FV)}{N\_d(FV)}\right) = \left(\frac{N\_\{naf\}(FV)}{N\_f(FV)} - \frac{N\_\{naf\}(FV) + N\_\{naf\}(\{F\})}{N\_f(FV)+1}\right) * \left(\frac{N\_d(FV) - N\_\{nad\}(FV)}{N\_d(FV)}\right)$$

- If F has no common meaning with at least one business activity, then $N\_\{naf\}(\{F\}) = 1$

$$\Rightarrow BFAM(FV \cup \{F\}) - BFAM(FV) = \left(\frac{N\_\{naf\}(FV)}{N\_f(FV)} - \frac{N\_\{naf\}(FV)+1}{N\_f(FV)+1}\right) * \left(\frac{N\_d(FV) - N\_\{nad\}(FV)}{N\_d(FV)}\right)$$

and so,

$$\frac{N\_\{naf\}(FV)}{N\_f(FV)} - \frac{N\_\{naf\}(FV)+1}{N\_f(FV)+1} = \frac{N\_\{naf\}(FV) - N\_f(FV)}{N\_f(FV) * (N\_f(FV)+1)} \leq 0$$

$$\Rightarrow BFAM(FV \cup \{F\}) \leq BFAM(FV)$$

because the number of misaligned functions is lower than or identical to the complete number of functions.

The BFAM measure is identical to the previous alignment measure when $N\_f(FV) = N\_\{naf\}(FV)$ i.e., when no function of the functional view FV is aligned with the business view.

- If F has a common meaning with at least one activity of the business view, then $N\_\{naf\}(\{F\}) = 0$

$$\Rightarrow BFAM(FV \cup \{F\}) - BFAM(FV) = \left(\frac{N\_\{naf\}(FV)}{N\_f(FV)} - \frac{N\_\{naf\}(FV)}{N\_f(FV)+1}\right) * \left(\frac{N\_d(FV) - N\_\{nad\}(FV)}{N\_d(FV)}\right)$$

and so,

$$\frac{N\_\{naf\}(FV)}{N\_f(FV)} - \frac{N\_\{naf\}(FV)}{N\_f(FV)+1} = \frac{N\_\{naf\}(FV)}{N\_f(FV) * (N\_f(FV)+1)} \geq 0$$

$$\Rightarrow BFAM(FV \cup \{F\}) \geq BFAM(FV)$$

The BFAM measure is identical to the previous alignment measure when $N\_\{naf\}(FV) = 0$, i.e., when all functions of the functional view are aligned with the business view.

- The proof is similar for the **BFA_2**, **BFA_3** and **BFA_4** axioms.

The alignment measure **BFAM** may be the stop criterion of an iterative development process. A higher estimation of the alignment implies a better consistency between the target business and target functional views.

V. IMPROVING THE ALIGNMENT OF THE FUNCTIONAL VIEW WITH THE BUSINESS VIEW

Aligning the functional view with the business view is one of the most complex activities of EA, since it consists of integrating in an abstract view the very concrete strategy of the enterprise. To help perform this alignment, we propose to type functions with respect to duration of its instance compared to business process of the enterprise.

A. Alignment Improvement Home Domain Analogy

A first alignment improvement of "Home IS", designed in Section IV with "Home processes" could be deduced from updating the "Home processes". For example, a new *Taking out food from the freezer* activity could be added to *Have a meal at home* "Home process". *Taking out food from the freezer* should come before *Cooking at home* because the second activity needs the result of the first. Alignment between the *Get food* "Home function" and the *Taking out food from the freezer* activity indeed improves the alignment of "Home IS" compared to "Home processes". A second improvement of this alignment takes into consideration the functional dependency of *Cook at home* on *Get food*, which could be aligned with an activity succession of the *Taking out food from the freezer* activity, which in turn comes before the *Cooking at home* activity. Function dependency aligned with this succession is represented inside a UML component diagram in Figure 7.

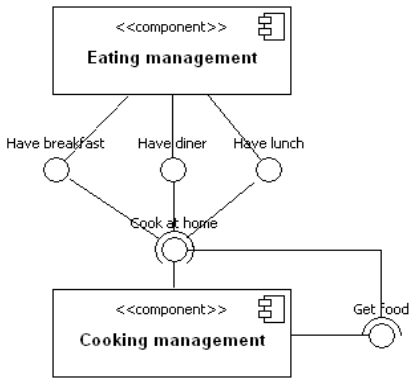


Figure 7. New "Home IS", which owns a new "home function dependency: *Cook at home* on *Get food*.

A new dependency of *Cook at home* on *Get food* allows an alignment with the activity succession (*Taking out food from the freezer* activity comes before *Cooking at home* activity).

Alignment improvement is the result of functional dependency design, which is consistent with activity succession. In the following section, we propose rules, named **BFRs**, that provide for alignment improvement from functional views in comparison with business views.

B. Typing of functions in relation to the Business View

The principle of typing functions comes from the architectural layer concept from the OSI model (Open Systems Interconnect) [16]. Typing each element allows to locate it on a specific layer. This principle is applied to the design of functions and of functions' dependencies.

A function owns types, which distinguishes it from the business activities aligned with it. The following types are consistent with a networking system [17]:

- *Command*, if this function is aligned with a business activity, then it has a life duration depending on the instances of business processes containing it (for example, *Give a telecommunication service order* function is aligned with a business activity, which has a life duration depending on the *Order capture* process (defined in [18])),
- *Data*, if this function is aligned with a business activity, then it has life duration that is independent of the instances of business processes containing it (for example, *Have access to telecommunication service catalogue* function is aligned with business activity, which has a life duration independent of the *Order capture* or *Billing* processes containing catalogue management activity).

Let F be the set of functions and T be the set of function types, then a type *t* of function *f* is the power set of T:

$$\boxed{\begin{array}{l} t : F \rightarrow P(T) \\ f \mapsto t; t \subset P(\{Command, Data\}) \end{array}} \quad (2)$$

So, a function can be of type {command, data} if it is aligned with two activities, such as life duration depending on the processes containing it and life duration independent of the processes containing it.

The functional architecture represented in Figure 4 shows four functions:

- *Create message filtering rule* function, which is of both *Command* and *Data* types. An alignment improvement would identify business activities that share a sense in common with this rule creation function. Two activities must be inserted in the messaging telecom service use process: *Filtering rule creation* and *Filtering rule consultation*. On the one hand, *Filtering rule creation* activity has a life duration, which depends on the use process, while *Filtering rule consultation* does not depend on it. We may indeed consult a filtering rule, while other messaging telecom service use process instances do not;
- *Filter a message from the message sender* and *Filter a message from the message object* functions own *Command* type. Their life duration depends on messaging telecom service process instances; and
- *Receive a message* function owns *Command* type for the same reason.

From definition (2), the typing can be expressed as:

$$t(\text{Create message filtering rule}) = \{Command, Data\}$$
$$t(\text{Filter a message from the message sender}) = \{Command\}$$
$$t(\text{Filter a message from the message object}) = \{Command\}$$
$$t(\text{Receive a message}) = \{Command\}$$

In our messaging telecom service, the *Create message filtering rule* function has both *Command* and *Data* types. To improve alignment with the business view, function typing should associate each function to only one type (*Command* or *Data*). These functions, *Create message filtering rule* and *Consult message filtering rule*, are illustrated in a UML sequence diagram in Figure 8.

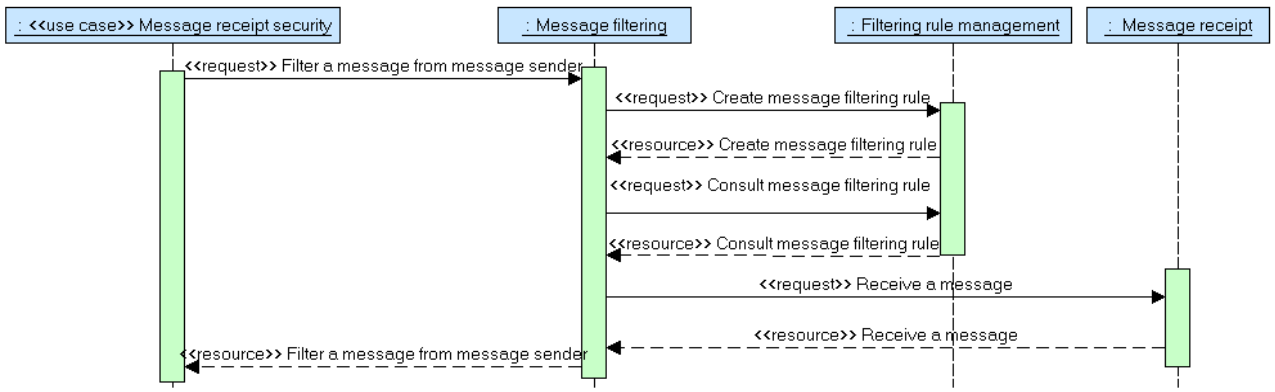


Figure 8. Functional view sequence of messaging service with a multi-typed function splitting example.

We propose then the first two rules, **BFR1 and BFR2,** for improved alignment with the business view:

BFR1 – Multi-typed function splitting. Multi-type functions must be split in sub-functions so that each sub-function is only of one type.

BFR1 improves alignment because activities are under the responsibility of specific roles. Activities with life durations dependent on processes are indeed under the responsibility of the enterprise's front-office, and activities with life durations independent of processes are mainly under the responsibility of an enterprise's back-office.

On one hand, the *Filtering rule creation* activity is under the responsibility of the *Child protection provider*, which is a telecom service back-office role. On the other hand, *Filtering rule consultation*, *Filter a message from the message sender* and *Filter a message from the message object* are activities under the responsibility of the *Messaging service provider* role, which is a telecom service front-office role.

Type pattern [19] may complete function typing, defining relationships between mono-typed functions. This pattern must be consistent with type definitions in relation to the business view. Pattern motivation enables alignment improvement of the functional view with the business view. A pattern may, for example, be associated with types as defined in (2). The UML class diagram in Figure 9 illustrates that a *Command* function depends on the *Data* function. An activity, which has life duration dependent on the instance of a business process, comes before an activity that has life duration independent of the instances of business processes.

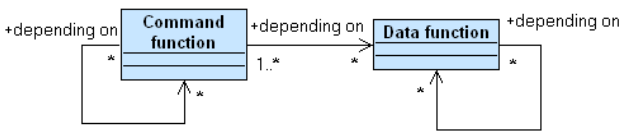


Figure 9. Type pattern applied to {Command, Data} function typing.

The motivation for this pattern is that each process provides one end-result for its "process customer" mapping with at least one activity, whose life duration depends on the instance of this process [20]. This activity may then need the end results of activities that have life durations independent of the instance of this business process. For example, in order process, order activity comes after a product reference consultation activity.

We can complete this type pattern with the following rules:

- *Command* function could depend on a *Data* function;
- *Command* function could depend on another *Command* function;
- *Data* function could depend on another *Data* function; and
- *Data* function never depends on *Command* function.

BFR2 – Multi-typed function sub-functions model. Sub-functions deduced from one multi-type function must satisfy a type pattern.

Considering the *t* function defined in (2), the type pattern motivated in Figure 9 is applied to sub-functions in Figure 10.

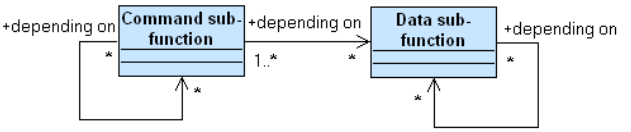


Figure 10. Type pattern applied to sub-functions typed by {Command, Data}

Alignment of the functional view with the business view may be improved because of type pattern consistency in relation to business view design.

C. *Typing of Functional Component Dependency in relation to the Business View*

Type pattern consistency with the business view may also be applied to function dependency.

BFR3 – Mono-typed functions model. A dependency between two mono-typed functions must respect type pattern for dependency existence and dependency orientation.

To decide the orientation of this dependency, let us note that each process has to be supported first by a command-typed function.

Messaging dynamic illustrations in Figures 5, 8 and 11 all show that message receipt process is supported by *Filter a message from message sender*, which is a *Command* typed function. A dependency of the *Consult message filtering rule* on the *Create message filtering rule* is designed in the messaging sequence diagram. This dependency does not satisfy BFR3 because the *Consult message filtering rule* is a *Data* typed function and the *Create message filtering rule* is a *Command* typed function. The business interpretation of this dependency is that one activity of message rule consulting is not useful in message rule creation.

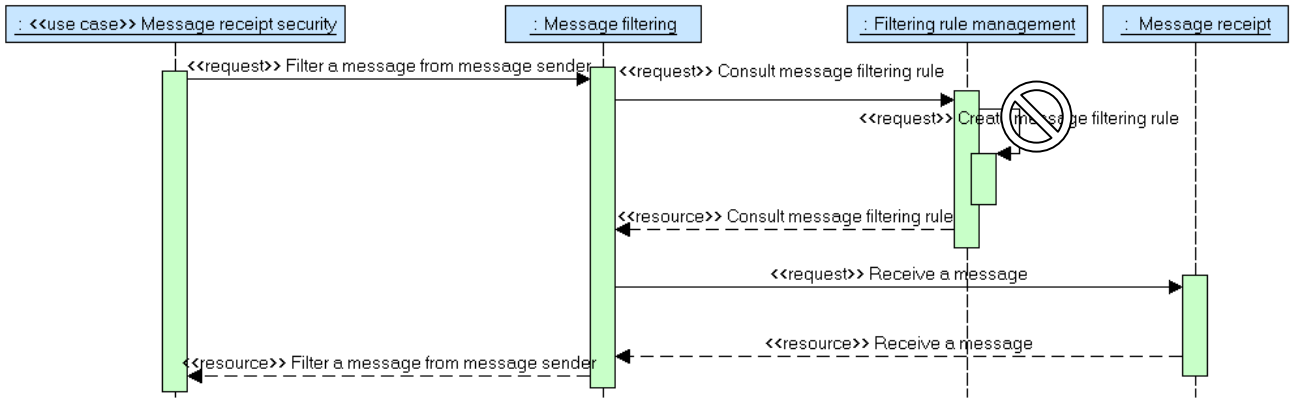


Figure 11. Functional view sequence of messaging service with mono-typed functions dependency example.

This dependency can, moreover, be analyzed at the data production level and with a data access security point of view. A command-typed function's need for a data-typed function to enable a process to be achieved is illustrated by the dependency of data characterizing the reception of an email, such as its reception date, toward data characterizing a filtering rule. This data dependency provides the opportunity to secure data produced by data-typed function. It is then possible to realize a data-typed function with high security requirements (in the present example: access control depending on the data and on the role of the user: administrator or developer of the mail service), but the realization of the command-typed block is associated with weaker security requirements (in the example: user authentication in the mail service).

VI. CASE STUDY

The alignment measure of the functional view with the business view is a tool for functional architects to compare the business alignment of various functional domains (messaging, IPTV, telephony, etc.) and so to prioritize their actions and improve the alignment. Such action can be guided by an assessment of the alignment. A case study with telecom messaging functions was conducted within the Orange Labs. This domain contains 8 functional components, 12 functions, and 16 function dependencies between functional components for six scenarios. The associated business view contains three activities and two activity time successions.

The alignment measure, **BFAM,** of the messaging domain functional view with the business view of telecom service usage is estimated (see formula (1)):

$$BFAM(\text{Messaging}) = \frac{7}{8}$$

Let us illustrate with a simple case how to improve the alignment. The assessment for messaging is the following: the alignment of the functional view of the Messaging domain with the business process of message sending would be perfect (i.e., with a measure estimated as 1) if the dependency relationship

- from the *Transmit a message* function defining *Message exchange* functional component
- on the *Store a message* function defining *Message storage* functional component

could be reversed (see Figure 12).

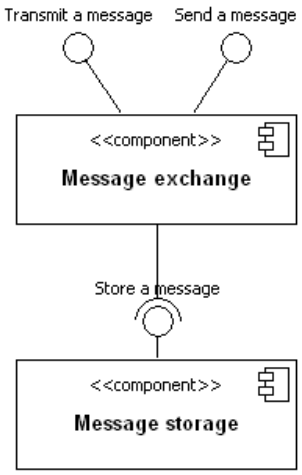


Figure 12. Messaging domain problem.

To maintain the aligned dependency from *Send a message* on *Store a message*, this reversal should require the *Message exchange* functional component to be split as follows into a:

- *Message sending* functional component defined by the *Send a message* function, and a
- *Message transmission* functional component defined by the *Transmit a message* function.

From definition (2), typing can be expressed as:

$$t(\text{Send a message}) = \{Command\}$$

$$t(\text{Transmit a message}) = \{Data\}$$

$$t(\text{Store a message}) = \{Command\}$$

Message transmission is indeed a network function within the telecommunication services IS. This function has a life duration independent of the instances of telecommunication service processes. Moreover, *Send a message* and *Store a message* are dependent on the telecommunication service process because the telecommunication service user has an access to data provided by these two functions (message sending date and message storage date, for example).

A dependency relationship of the *Message transmission* functional component on the *Message storage* results from the previous target functional architecture.

One dependency relationship represented in Figure 13 is consistent with the type pattern in Figure 9:

- a dependency relationship of the *Message storage* functional component on *Message transmission*, which improves the alignment.

Another dependency relationship represented in Figure 13 is consistent with the type pattern in Figure 10:

- a dependency relationship of the *Message sending* functional component on *Message transmission*, deduced from the *Message exchange* splitting.

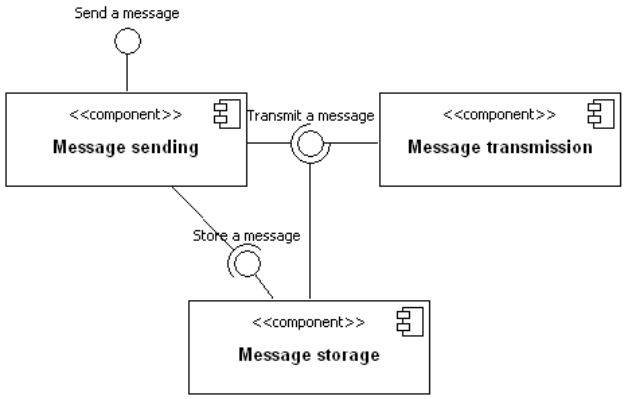


Figure 13. Messaging domain solution.

Using type patterns thus provides a perfect alignment: $BFAM(\text{Messaging}) = 1$ (see (1)).

Functional architects may use this assessment as a tool to improve the business alignment, by checking if the suggested modifications conform to the enterprise strategy.

VII. CONCLUSION

The modelling process described in this paper enables a representation of the alignment of an IS functional view with the business view of the IS company. The alignment definition is consistent with the meta-modelling used to instantiated both the business and the functional models. An alignment measure is proposed, which provides estimation of the synchronization of a company's strategic integration of business and functional views.

Finally, a good alignment of the target functional view with the business target view induces a good alignment of the applicative view, which in turn implements the target functional view with the target business view. We illustrate this applicative alignment in Figure 14 with our "home domain". An applicative view that implements the functional view is represented in Figure 7.

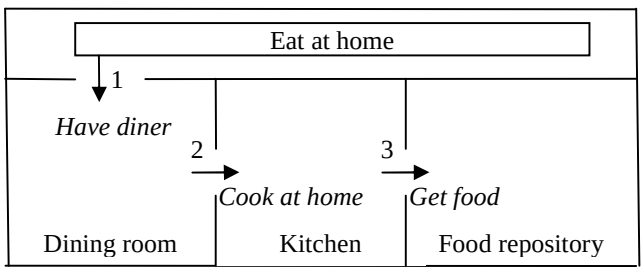


Figure 14. "Home IS" applicative view implementing the *Eat at home* "Home process".

Functional dependency is implemented in the applicative view by only one call from the "Home bus". Function dependency implementation is deployed on the doors between rooms. "Home bus" traffic is thereby lowered. This

property, deduced from the alignment improvement of the functional view in compared to the business view, may be significant in Service Oriented Architecture (SOA) [21] design.

The applicative view may contain several hundred applications, which can hardly be directly mapped with the company business processes. In our approach, the target functional view is used as a link between the business and applicative views. This indirect mapping allows an efficient tool to govern IT evolution according to company strategy.

REFERENCES

[1] J. Simonin, E. Bertin, Y. Le Traon, J.-M Jézéquel, and N. Crespi, "Business and Information System Alignment: a Formal Solution for Telecom Services," International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, 2010.

[2] J.A. Zachman, "A Framework for Information Systems Architecture," IBM Systems Journal 26, no. 3, 1987, pp. 276–292.

[3] C. Longépé, "The Enterprise Architecture IT project – The Urbanisation paradigm," Kogan Page, 2003.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, "The Unified Software Development Process," Addison-Wesley, 1999.

[5] C.M. Pereira and P. Sousa, "Getting into the Misalignment between Businesss and Information Systems," 10th European Conference on Information Technology Evaluation, Madrid, Spain, 2003.

[6] J.C. Henderson and N. Venkatraman, "Strategic Alignment: Leveraging Information Technology for Transforming Organizations," IBM Systems Journal, vol. 32, no. 1, 1993, pp. 4–16.

[7] The Business Rules Group, "The Business Motivation Model," http://www.businessrulesgroup.org/bmm.shtml, 2007.

[8] J. Simonin, Y. Le Traon, and J.-M.Jézéquel, "An Enterprise Architecture Alignment measure for Telecom Service development," 11th IEEE International Enterprise Distributed Object Computing Conference, Annapolis, Maryland U.S.A., 2007.

[9] J.N. Luftman, R. Papp, and T. Brier, "Enablers and Inhibitors of Buisness-IT Alignment," Communications of the Association for Information Systems, vol. 1, article 11, 1999.

[10] S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no. 6, 1994, pp. 476–493.

[11] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language – User Guide," Addison – Wesley, 1999.

[12] P.P.S. Chen, "The entity relationship model - Towards a unified view of data," ACM Transactions on Database Systems, vol. 1, no. 1, 1976, pp. 9–36.

[13] Open Mobile Alliance, "OMA Service Environment," OMA-Service-Environment-V1_0_2-20050803-A, 2005.

[14] M.J. Shepperd and D. Ince, "Derivation and Validation of Software Metrics," Oxford University Press, 1993.

[15] S.G. MacDonell, M.J. Shepperd, and P.J. Sallis, "Metrics for Database Systems: An Empirical Study," Software Metrics Symposium, 1997, pp. 99–107.

[16] H. Zimmermann, "OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection," IEEE Transactions on Communications COM-28, no. 4, 1980.

[17] M. Farsi, K. Ratcliff, and M. Barbosa, "An overview of controller area network," Computing & Control Engineering Journal, vol. 10, n°3, 1999, pp. 113–120.

[18] TM Forum, "enhanced Telecom Operations Map," http://www.tmforum.org/browse.aspx?catID=1648

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns – Elements of Reusable Objetc-Oriented Software," Addison Wesley, 1995.

[20] M. Hammer, J. Champy, "Reengineering the Corporation: a Manifesto for Business Revolution," Harper – Collins, 1993.

[21] R.T.Burlton, "Business Process Management – Profiting from Process," Sams Publishing, 2001.

Model Validation in a Tool–Based Methodology for System Testing of Service–Oriented Systems

Michael Felderer*, Joanna Chimiak–Opoka*, Philipp Zech*, Cornelia Haisjackl*, Frank Fiedler†, Ruth Breu*
*Institute of Computer Science
University of Innsbruck
Innsbruck, Austria
Email: {michael.felderer, joanna.opoka, philipp.zech, cornelia.haisjackl, ruth.breu}@uibk.ac.at
†Softmethod GmbH
Munich, Germany
Email: frank.fiedler@softmethod.de

***Abstract*—In this article we present a novel model–driven system testing methodology for service–centric systems called Telling TestStories, its tool implementation and the underlying model validation mechanism. Telling TestStories is based on tightly integrated but separated platform–independent requirements, system and test models. The test models integrate test data tables and encourage domain experts to design tests. This process is supported by consistency, completeness, and coverage checks in and between the requirements, system and test models which guarantees a high quality of the models. Telling TestStories is capable of test–driven development on the model level and provides full traceability between all system and testing artifacts. The testing process of the Telling TestStories methodology comprises model development, model validation and system validation. The model development and the system validation are managed by the Telling TestStories tool and the model validation is managed by the SQUAM tool. All process steps, the underlying artifacts and the tools for implementing the process steps are presented by an industrial case study.**

***Keywords*-Model–Driven Testing; System Testing; Model Validation; Testing Methodology; Testing Tools; Service–Oriented Architecture;**

I. Introduction

The number and complexity of service–oriented systems for implementing flexible inter–organizational IT based business processes is steadily increasing. Basically, a *service–oriented system* consists of a set of independent peers offering services that provide and require operations [1]. Orchestration and choreography technologies allow the flexible composition of services to workflows [2], [3]. Arising application scenarios have demonstrated the power of service–oriented systems. These range from the exchange of health related data among stakeholders in health care, over new business models like SAAS (Software as a Service) to the cross-linking of traffic participants. Elaborated standards, technologies and frameworks for realizing service–oriented systems have been developed, but system testing tools and methodologies have been neglected so far.

System testing of service–oriented systems, i.e., validating the system's compliance with the specified requirements, has to consider specific issues that limit the testability of such systems including the integration of various component and communication technologies, the dynamic adaptation and integration of services, the lack of service control, the lack of observability of service code and structure, the cost of testing, and the importance of service level agreements (SLA) [4].

Model–driven testing approaches [5], i.e., the derivation of executable test code from test models by analogy to Model Driven Architecture (MDA) [6], are particularly suitable for system testing of service–oriented systems because they can be adapted easily to changing requirements, they support static model validation to improve the quality of the tests, they provide an abstract technology and implementation independent view on tests, and they allow the modeling and testing of service level agreements. The latter allows for defining test models in a very early phase of system development even before or simultaneous with system modeling supporting test–driven development on the model level.

In this article, we present a tool–based methodology to model–driven system testing of service–oriented systems called *Telling TestStories* (TTS) and its integrated model validation mechanism. The methodology is explained by an industrial case study from the telecommunication domain.

TTS is based on separated but interrelated requirements, system, and test models. All requirements in the requirements model are traceable to system and test model artifacts. The test model invokes operations provided by system services.

The quality of manually designed models and therefore the quality of the overall test results in our methodology can significantly be improved by model validation which is therefore a core component of TTS.

Model validation is an activity where the model is statically analyzed against a set of consistency and completeness criteria and metrics. *Consistency* criteria assure that a model is non–contradictory, and *completeness* criteria assure that a model element contains all essential information. We define intra–

model validation checks, e.g., that each test contains at least one assertion, and inter–model validation checks, e.g., that each requirement is traceable to at least one test. Additionally, we consider *coverage* checks, i.e., inter–model completeness checks where one model is the test model, e.g., that each system service is invoked in at least one test. Although our validation rules check the conformance of models and model elements, we do not use the term verification in our respect to avoid confusion with formal verification based on a proof system.

Besides the advantages of model–driven testing and model validation, our approach supports test–driven development on the model level, the definition and execution of tests in a tabular form as in the Framework for Integrated Testing (FIT) [7], guarantees traceability between all types of modeling and system artifacts, and is suitable for testing SLA which we consider as non–functional properties. We also show how our testing approach supports traceability between requirements, system and test models, and the system under test (SUT).

This article substantially extends the tool–based methodology for model–driven system testing of service–oriented systems presented in [8] where the model development and the system validation in TTS have been considered. Herein, we complete the work of [8] by also considering the model validation in TTS. We define consistency and completeness/coverage metrics and criteria in and between the requirements, system and test model. We also explain how the model validation has been implemented in the framework for Systematic Quality Assessment of Models (SQUAM).

The article is structured as follows. We first present the basic concepts of the TTS methodology by defining the underlying artifacts, the testing process and the metamodel of the model artifacts (see Section II). We then explain the model development, the model validation, and the system validation by a case study and its tool integration (see Section III) and describe the architecture of the TTS and SQUAM tool–implementations (see Section IV). Afterwards we provide related work (see Section V), and finally we draw conclusions and discuss future work (see Section VI).

II. Basic concepts of the methodology

A *testing methodology* defines a testing process and the underlying artifacts such as the defined models, the generated code, and the running systems.

In this section, we provide an overview of the TTS artifacts, the testing process and the metamodel of the model artifacts which is the basis for model validation.

A. Artifacts of TTS

Fig. 1 shows the artifacts and dependencies within the TTS framework. In the TTS framework we can distinguish three formalization levels with informal artifacts (at the top), model artifacts (in the middle), and implementation artifacts (at the bottom). Informal artifacts are depicted by clouds, formal models by graphs, code by transparent blocks and running systems by filled blocks. Due to formalization at the two lower levels we can conduct automatic transformations and validations. Formalized dependencies between artifacts are depicted by solid lines, whereas informal dependencies are depicted by dashed lines. In the following paragraphs, we explain the artifacts and dependencies of the TTS framework.

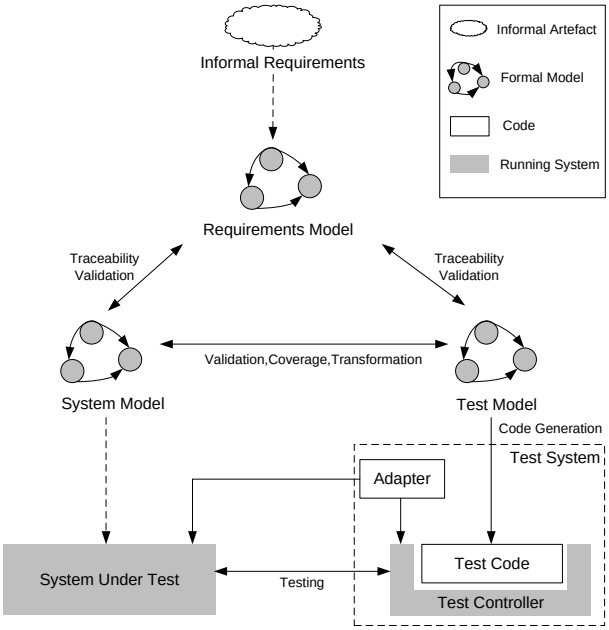


Fig. 1. Overview of the TTS Artifacts

1) Informal level: There is only one type of artifact on this level, namely the **Informal Requirements**, i.e., written or non–written capabilities and properties of the system. This artifact is not discussed in detail because it is not in the main focus of our testing methodology and as an informal one can not be automatically validated.

2) Model level: There are three models on this level: requirements, system and test model. The models are formally related through different dependencies related to: traceability, validation, coverage, and transformation.

Requirements Model. The requirements model contains the requirements for system development and testing. Its structured part consists of a requirements hierarchy. The requirements are based on informal requirements depicted as cloud. The requirements model provides a way to integrate textual descriptions of requirements which are needed for communication with non–technicians into a modeling tool.

System Model. The system model describes the system structure and system behavior in a platform independent way. Its static structure is based on the notions of services,

components and types. Each service operation call is assigned to use cases, actors correspond to components providing and requiring services, and domain types correspond to types. We assume that each service in the system model has a one–to–one correspondence to an executable service in the running system to guarantee traceability. Therefore the requirements, the service operations and the executable services are traceable.

Test Model. The test model defines the test data and the test scenarios as so called test stories. *Test stories* are controlled sequences of service operation invocations exemplifying the interaction of components. Test stories may be generic in the sense that they do not contain concrete objects but variables which refer to test objects provided in tables. Test stories can also contain setup procedures, tear down procedures and contain assertions for test result evaluation. The notion of a test story is principally independent of its representation. We have used UML sequence diagrams in previous case studies [9] and use activity diagrams in this article. Test stories include references to a table of test data including values for all free parameters of the test story. Each line in this table defines test data for one test case. We use the terms 'test story' and 'test' interchangeably in this article depending on the context. If we address the more abstract view, we use the term 'test'. If we address the more application–oriented and process–oriented view, we use the term 'test story'.

Traceability. For model maintenance, transformations and validations traceability between different model elements is required. In the TTS framework traceability between elements on the model level is guaranteed by links between model elements, and between the model level and the implementation level by adapters for each service. The adapters link service calls in the model to executable services. Therefore every service invocation is traceable to a requirement.

Transformation. In the TTS framework we consider model–to–model transformations to obtain a (partial) test model from the system model. In such a system–driven development approach, test behavior and test data can be extracted from the graph of a global or local workflow.

Validation. Models designed manually require tool supported validation. Our approach is suitable for test–driven modeling because the test model is used to validate the system. In the context of TTS, we consider two properties: consistency and completeness/coverage.

Consistency checks assure that there is no conflicting information in models. Consistency of a model enables error–free transformation from the model to another model or to the source code. For manually designed (parts of) models consistency within and between them should be automatically checked. In TTS we have implemented consistency criteria for all three models and between pairs of them.

Completeness checks assure that one artifact is complete, i.e., contains all essential information. Similarly like for consistency, we can consider completeness within one model (for elements and their properties) and between models. Completeness of the system model is crucial for the TTS framework and determines whether transformations from the system model to the test model can be applied. If the system model is complete, then behavioral parts of the test model can be generated by model transformations.

Coverage can be considered as a variant of inter–model completeness where one model is the test model. This aspect is very important in context of testing and is used to check to what extend the test model covers the requirements and system model and implicitly the system. We adopted a series of coverage criteria from testing [10] and model–driven testing [11] to fit into the TTS framework.

3) Implementation level: At this level the test code generated from the test model is executed by the test controller against the system under test. The executable services of the system under test are invoked by adapters.

Code Generation. The test code is generated automatically by a model–to–text transformation from the test model as explained in [12]. For each test in the test model, a test code file is generated.

Test Code. The test code language is Java. Adapters which bind abstract service calls in the test code to running services of the system under test make the test code executable.

Adapters. The adapters are needed to access service operations provided and required by components of the system under test. For a service implemented as web service, an adapter can be generated from its WSDL description. Adapters for each service are the link for traceability between the executable system, the test model and the requirements. Adapters make it possible to derive executable tests even before the system implementation has been finished which supports test–driven development.

Test Controller. The test controller executes the test code and accesses the system services via adapters. Our implementation of the test controller executes test code in Java but other JVM–based programming or scripting languages are also executable without much implementation effort.

Test System. The test controller, the adapter and the test code constitute the test system.

Testing. The evaluation of the service–centric system by observing its execution [10] is called testing. Services are invoked in the test code executed by the test controller via adapters.

System Under Test. The system under test is a service–oriented system, i.e., offering services that provide and require interfaces. It may contain special interfaces for testing purposes.

The informal requirements can be considered as external input to the TTS framework, and the system under test is the target of the application of TTS. This is shown by two dashed arrows in Fig. 1. The first dashed arrow goes from the informal requirements to the TTS requirements model, and the second dashed arrow goes from the test controller and adapters to the system under test. Both, the informal requirements and the system under test, are out of the scope of this article.

B. Testing Process

The process consists of a *design*, *validation*, *execution*, and *evaluation* phase and is processed in an iterative way. Initially, the process is triggered by requirements for which services and tests have to be defined. The process is depicted in Fig. 2.

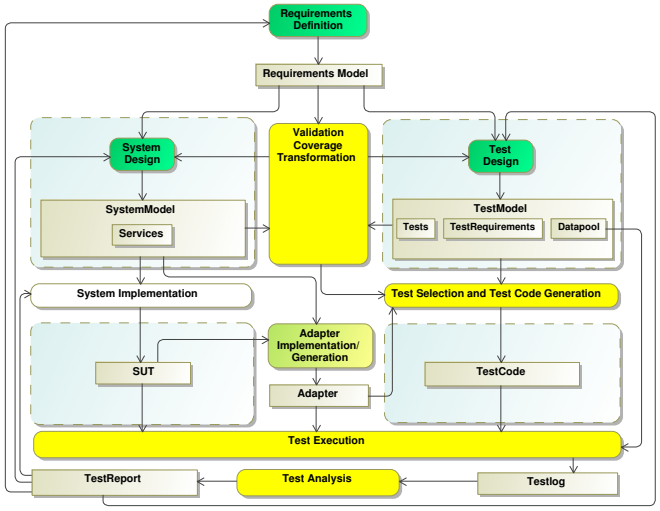


Fig. 2. Model–driven Testing Process

The first step is the definition of requirements. Based on the requirements, the system model containing services and the test model containing tests are designed. The test design additionally includes the data pool definition and the definition of test requirements. The system model and the test model, including the tests, the data pool and the test requirements, can be validated for consistency and completeness and checked for coverage. In a system–driven approach tests can be generated from the system model by model–to–model transformations, and in a test–driven approach tests can be integrated in the system model. This validity checks allow for an iterative improvement of the system and test quality. In principle, the testing process can also be considered as test model–driven development process. Our methodology does not consider the system development itself but is based on traceable services offered by the system under test. As soon as adapters which may be – depending on the technology – generated (semi–)automatically or implemented manually are available for the system services, the process of test selection and test code generation, i.e., model–to–text transformation can take place. In the tool implementation, adapters for web services can be generated automatically based on a WSDL description, adapters for RMI access can only be generated semi–automatically. The generated test code is then automatically compiled and executed by a test controller which logs all occurring events into a test log. The test evaluation is done offline by a test analysis tool which generates test reports and annotations to those elements of the system and test model influencing the test result. Test reports and test logs are implementation artifacts that are not important for the overall process but for the practical evaluation. Therefore test reports and test logs have not been considered in the previous section.

In [13] we have introduced the term *test story* for our way how to define tests by analogy to the agile term user story defining a manageable requirement together with acceptance tests.

The separation of the test behavior and the test data has been influenced by the column fixture of the Framework for Integrated Test (FIT) [7] which allows the test designer to focus directly on the domain because tests are expressed in a very easy–to–write and easy–to–understand tabular form also suited for data–driven testing.

The manual activities in the TTS testing process and the test execution are conducted by specific roles. In Fig. 3 these roles and their activities are shown.

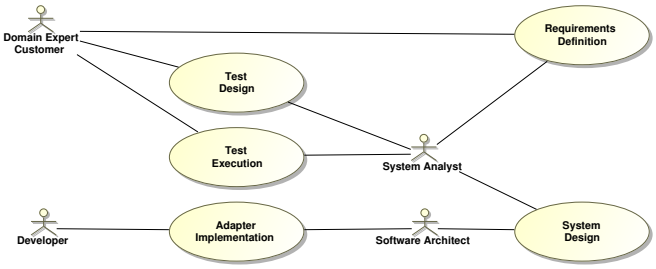


Fig. 3. Roles in the TTS Process

A *Domain Expert* or *Customer*, which are represented by one role in Fig. 3, are responsible for the test design and the requirements definition. Additionally, domain experts or customers may initiate the test execution and all related automatic activities (validation, test code generation, test analysis). Domain experts and customers are responsible for the same activities but have different views on testing, i.e., domain experts represent the internal view conducting system tests and customers represent the external view conducting acceptance tests.

A *System Analyst* is partially responsible for the test design, the requirements definition, and the system design. Additionally, the system analyst may also initiate the test execution and all related activities. The system analyst is especially responsible for the definition of non–functional requirements, e.g., for security or performance and corresponding tests.

A *Software Architect* is partially responsible for the system design and the adapter implementation. The software architect defines interfaces and component technologies in coordination with the system analyst and the developer.

A *Developer* implements adapters if they have to be developed manually.

C. Metamodel

In this section we define a metamodel for the requirements model, the system model, and the test model of TTS. The consistency, completeness, and coverage checks defined in Section III-B are based on this metamodel. The metamodel is shown in Fig. 4.

The package `Requirements Model` defines the element `Requirement` which is supertype of the types `FunctionalRequirement` and `NonFunctionalRequirement`. The type `NonFunctionalRequirement` may have further subtypes for specific types of non–functional requirements, e.g., security or performance. The requirements itself can be of an arbitrary level of granularity ranging from abstract goals to concrete performance requirements. Requirements are modeled explicitly on the metamodel level to define traceability between requirements and other model elements such as tests. The requirements model is very generic but it can easily be extended for specific purposes.

The package `System Model` defines `Service` elements which provide and require `Interface` elements and are composed of basic services. Each interface consists of `Operation` elements which refer to `Type` elements for input and output parameters. Types may be primitive types, enumeration types or reference types. Operations may also have a precondition (`pre` constraint) and a postcondition (`post` constraint). Each service has a reference to `Actor` elements. It is also possible to identify services with a service operation (if there is only one) and to identify them with service calls. Services can be therefore be considered as executable use cases. Services may have `LocalProcess` elements that have a central control implemented by a workflow management system defining its internal behavior. Different services may be integrated into a `GlobalProcess` without central control. Orchestrations can be modeled as local processes, and choreographies as global processes.

The package `Test Model` defines all elements needed for system testing of service–centric systems. A `TestSequence` consists of `SequenceElement` blocks, that integrate a `Teststory`, its `DataList`, and an `Arbitration`. A `Teststory` consists of the following elements:

- `Assertion` elements for defining expressions for computing verdicts,
- `Call` elements, i.e. `Servicecall` elements for invoking operations on services, or `Trigger` elements for operations that are called by a service,
- `ParallelTask` elements for the parallel execution of behavior, or
- `Decision` elements for defining alternatives.

`Testsstory` elements are completely recursive and, in principle, there is no limit to the number of levels to which tests can be nested. However, in practice nesting depths greater than three are not applied and it is even not clear what use nesting depths greater than two or three would be.

A `DataList` contains `Data` elements that may be generated by a `DataSelection` function. A `Testsequence` has several `TestRun` elements assigning `Verdict` values to assertions. The verdict can have the values *pass*, *fail*, *inconclusive*, or *error*. Pass indicates that the SUT behaves correctly for the specific test case. Fail indicates that the test case has violated. Inconclusive is used where the test neither passes nor fails. An error verdict indicates exceptions within the test system itself. In the model itself only a pass or a fail can be specified. Inconclusive or error are assigned automatically. This definition of verdicts originates from the OSI Conformance Testing Methodology and Framework [14].

Assertions are boolean expressions that define criteria for computing pass, fail or inconclusive verdicts. Assertions can access all variables in the actual evaluation context.

The system model and the test model are created manually or are partially generated from each other. If the system model and the test model are created manually, their quality is validated by consistency, completeness and coverage rules. Alternatively, if the system model is complete then test scenarios, test data and oracles can be generated. If the test model is complete, then behavioral fragments of the system model can be generated.

An `Arbitration` element defines a criterion on the set of the verdicts of a test run to determine whether a sequence of tests assigned to a `SequenceElement` has been executed successfully or not.

In the case of a test, a data list defines a test table, i.e., a list of lists. Data selection functions for example randomly generate integers within a specific range. This function is then denoted by `genInt(a,b)` and generates a random integer between the integers a and b.

The TTS metamodel has been implemented as a UML profile. For all metamodel elements despite data–specific elements that are implemented in tables, stereotypes of the same name have been introduced and assigned to UML metaclasses, e.g., a `Service` in our metamodel is assigned to the UML metaclass `Class`, a `Requirement` is assigned to `Class`, a `Testsequence` is assigned to an `Activity`, a `SequenceElement` is assigned to an `Action`, a `Teststory` is assigned to an `Activity`, and a `Servicecall` is assigned to an `Action`.

III. Tool–Based Case Study

In this section we present a tool–based case study for our testing methodology. We consider the the model development

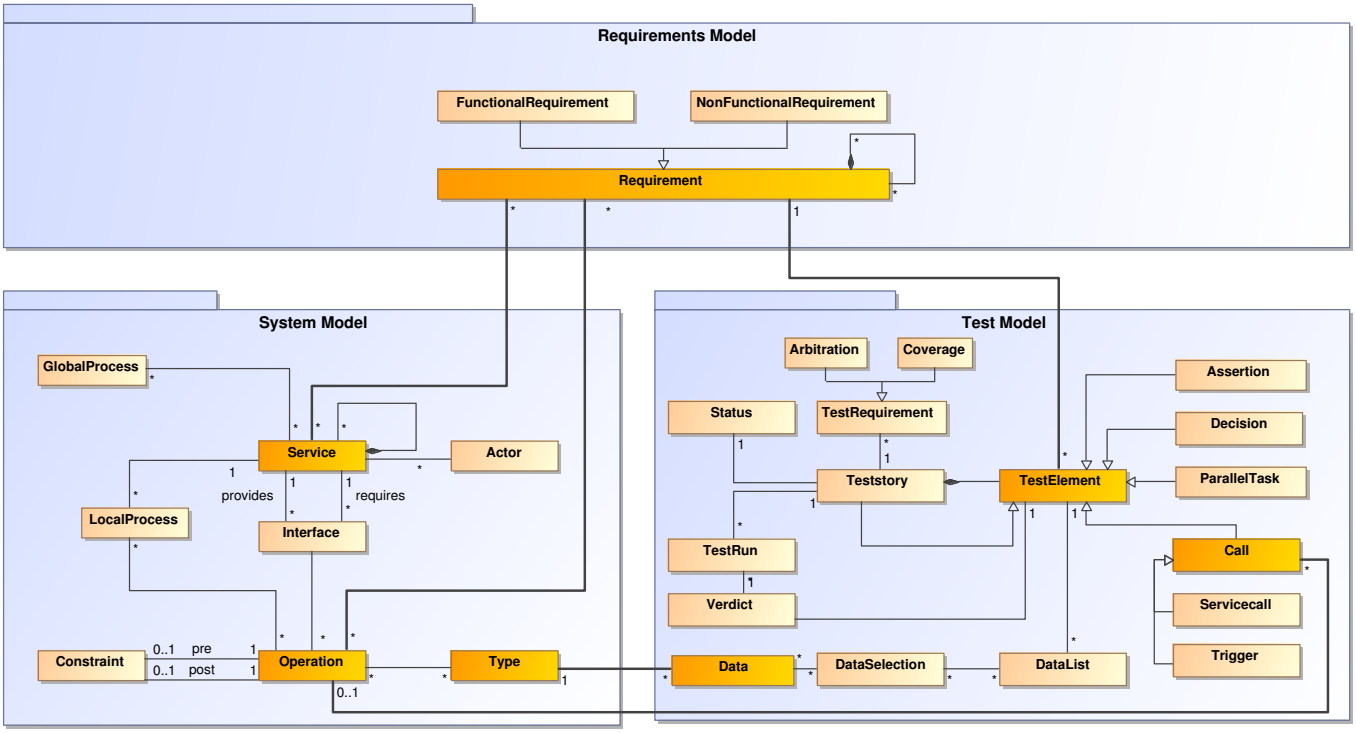


Fig. 4. Requirements, System and Test Metamodel of TTS

phase (see Section III-A), the model validation phase (see Section III-B), and finally the system validation phase (see Section III-C) of the TTS testing process.

A. Model Development

We have applied our testing tool on several case studies, including an industrial one. In this section, we explain our methodology and its tool implementation by a *Telephony Connector* case study.

The Telephony Connector is an application in the area of Computer Telephony Integration (CTI) and parts of it have already been tested with unit tests. But the whole application can currently only be tested by manual tests. TTS provides model–driven testing support for the telephony connector which is more efficient concerning the testing time and the error detection rate.

In this section, we explain how we have tested the telephony connector case study with our framework and which conclusions can be drawn. As first step, we have developed the requirements, the static parts of the system model and the test model with our tool.

The requirements are modeled as class diagram where high level requirements are aggregated by low level functional and non–functional requirements. This representation is analogous to requirements diagrams of SysML [15] and guarantees that requirements are integrated into the model which simplifies the implementation of traceability.

The requirements for routing a call are depicted in Fig. 5.

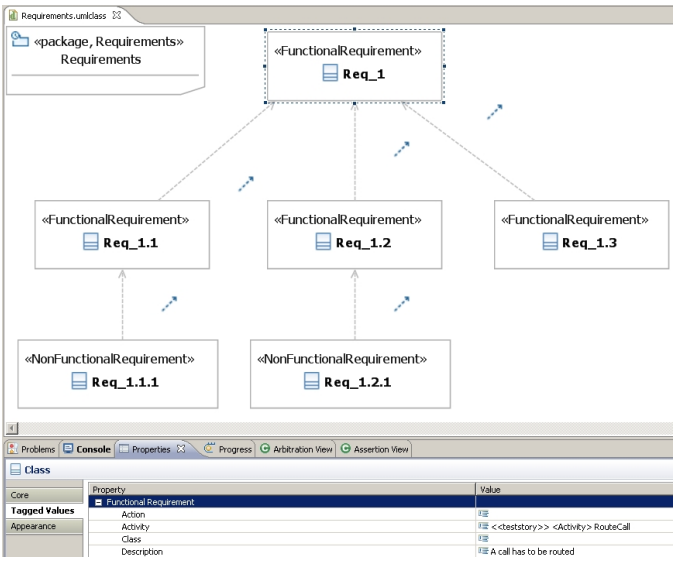


Fig. 5. Requirements to the Callmanager Application

We have modeled a requirement for routing a call (`Req_1`) and its parts, including non–functional performance requirements to hangup a call within 1000ms (`Req_1.1.1`) and to send the route signal within 1000ms (`Req_1.2.1`).

In the system model types are represented as class diagrams, and services as classes with their providing and requiring interfaces. In Fig. 6 the interfaces provided by the service

VehicleService and *TelephonyConnectorService* are depicted. Required contracts for the operations depicted in the interfaces, are modeled in terms of pre– and postconditions. Yet these contracts may only be defined over the scope of input parameters to service invocations, as currently SUT specific program variables reside in another runtime and hence are outside the accessibility of TTS.

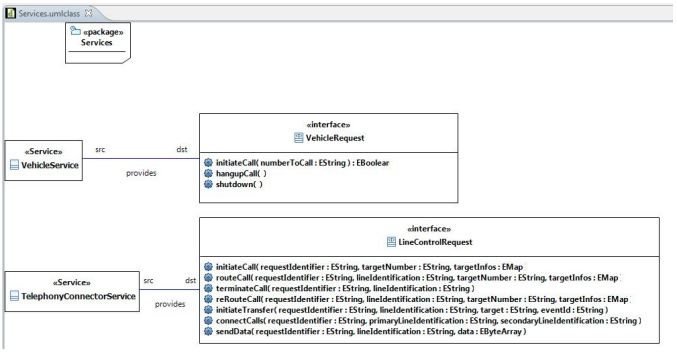


Fig. 6. Extract of the Services of the Callmanager Application

Local and global processes are modeled as behavior diagrams, i.e., state machines, activity diagrams or sequence diagrams. In our experiments we have mainly modeled global workflows by activity diagrams, and local workflows by activity diagrams and state machines.

Tests are modeled as activity diagrams or sequence diagrams. Test sequences or high level test suites are modeled as activity diagrams. In Fig. 7 a test is depicted.

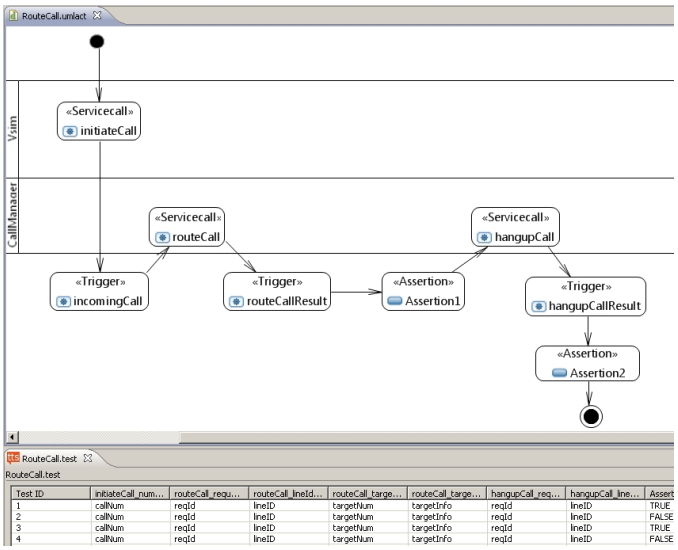


Fig. 7. Test story for the Callmanager Application

In the upper part of the test depicted in Fig. 7, the test *RouteCall* for routing a call is depicted. After the car service (*Vsim*) initiated a call, the Callmanager routes the call and terminates the call. The results of these calls are triggered on the test controller. Intermediate assertions check whether the result provided by the trigger equals to the expected one. Each test may have test data which is defined for the test story *RouteCall* in the table *RouteCall.test* depicted in the lower part of Fig. 7. The test stories and their corresponding data files are executed as test sequence elements depicted in Fig. 10 which defines an additional arbitration to define a global verdict specifying when all tests of a story have passed.

B. Model Validation

As mentioned in Section II-A model validation comprises checks for consistency and completeness or coverage. The validation is static, i.e., it is based on rules defined on metamodel elements without program execution. Due to the dynamic evaluation of test models provided by early test execution, static validation provides the most efficient and effective validation result: dynamic validation is provided by early test execution, and the modeling effort and computational complexity for static analysis is not as high as for verification techniques like model checking and constraint solving. The focus on static analysis is also supported by empirical research because in [16] it is shown that incompleteness and inconsistencies in UML models are already detected with OCL–based static analysis and that formal methods such as model checking or constraint solving are not required.

First each model is validated as a separate artifact (intra–model aspects) and then in relation to other models (inter–model aspects). In total we have 3 single models (requirements, system and test model) and 3 pairs of models (Table I). The inter–model relationship between the requirements, system and test model is implemented by tagged values or associations. For instance, the relationship between a requirement (in the requirements model) and a test (in the test model) is implemented by a tagged value of the requirement and the test model element referencing each other. In Fig. 5 the requirement `Req_1` is associated to the test `RouteCall`.

As mentioned before in TTS we investigate three types of validation rules: consistency, completeness, and coverage rules.

Consistency checks assure that there is no conflicting information in models. Consistency of a model enables error–free transformation from the model to another model or to the source code.

Completeness checks assure that one artifact is complete, i.e., contains all essential information. Completeness of the system model is crucial for the TTS framework and determines whether transformations from the system model to the test model can be applied.

Coverage can be considered as a variant of inter–model completeness where one model is the test model. This aspect is very important in context of testing and is used to check to what extend the test model covers the system model and implicitly the system. We adopted a series of coverage criteria from testing [10] and model–driven testing [11] to fit into TTS.

Additionally, we distinguish between criteria and metrics. *Criteria* provide only a boolean result: true if a model fulfills a given criterion, false otherwise. They provide a warning mechanism implemented, i.e., the severity levels information, warning, and error, to inform modelers about an incorrectness in a model. *Metrics* have a numeric result. Typically they provide information to what extend a corresponding criterion is fulfilled, thus they are fractions ranging from `0` to `100%`. They can be used for the evaluation of models in a similar way as for the evaluation of source code in [17]. Metrics are well suited for summarizing particular aspects of models and for detecting outliers in large models. They scale up and aggregate many details of models.

To address particular quality criteria we start from informal descriptions to obtain metrics at the final stage. First we define a criteria in natural language. We adapt it to the context of the TTS metamodel. Next, we express it either as a constraint over a model or as a boolean query. Finally, we construct a numeric metric for it. As a formalization language we apply the Object Constraint Language (OCL) [18]. Table I provides and overview of specified criteria and metrics as a proof of concept. Before we give more details about example criteria and metrics, we will describe their development process.

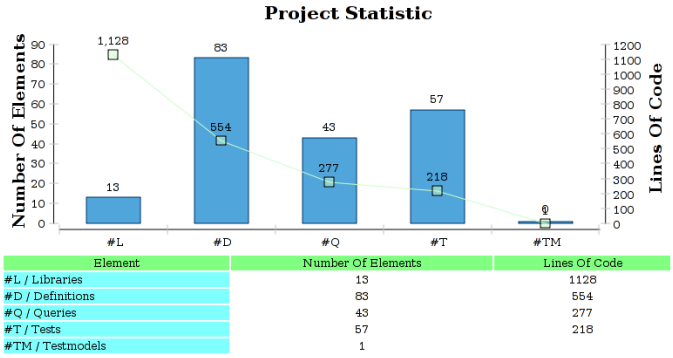
TABLE I
OVERVIEW OF VALIDATION ASPECTS

type	models	consistency metrics	consistency criteria	completeness/coverage metrics	completeness/coverage criteria	
intra	Requirements	×	√	√	√	complet.
intra	System	×	√	√	√	complet.
intra	Test	×	√	√	√	complet.
inter	Req–Syst	×	√	√	√	cov.
inter	Req–Test	×	√	√	√	cov.
inter	Syst–Test	×	√	√	√	cov.

To specify criteria and metrics we follow the model analysis and OCL library development process (Fig. 8). The upper swimlane corresponds to the manual model analysis, the lower swimlane to the library development process. First, a common requirement for model analysis and library development is specified. A quality aspect is selected, e.g., a completeness criterion defining that each requirement should have a unique name. For this aspect OCL definitions and queries are specified in the development step. The next step is quality assessment, where the results of the manual and automatic analysis are cross–checked. For the selected aspect, manual inspection is used to determine the result of this aspect for the model. Simultaneously, appropriate queries are evaluated on the model. If the results of the model inspection and the query evaluation differ, the reason has to be determined and either the OCL definition specification or manual inspection needs to be repeated. The manual inspection of the model is conducted as long as correctness of a query achieves a defined confidence level. Afterwards the query can be used for automatic model analysis.

If the results are equal, the last step, i.e., quality assurance, can be executed. The aim of this step is to assure semantic correctness of OCL expressions in the future development of the library. For this purpose OCL unit tests [19] are specified and evaluated regularly. In the test evaluation step, OCL unit tests with corresponding OCL test models are required. The OCL test model is an instance of the requirements, system and test metamodel. Note that the OCL test model is not the same as a test model in TTS but a model instance of the considered metamodel for the OCL expression under test. This instance is used as test data for OCL unit tests to assess the desired semantics of definitions. OCL unit tests are similar to JUnit [20] tests applied to assess the semantic correctness of source code.

In Fig. 9 we show the size of the OCL project developed for the TTS approach. We specified `83` definitions used in `43` queries and split into `13` libraries. The number of OCL expressions is higher than the total number of the criteria (26) and metrics (10) as it was necessary to define helper methods. The helper expressions can be used in the specification of further criteria/metrics and make their development less time consuming. To assure correctness of OCL expressions we wrote `57` OCL unit tests [19] and evaluated them over one test model.



Element	Number Of Elements	Lines Of Code
#L / Libraries	13	1128
#D / Definitions	83	554
#Q / Queries	43	277
#T / Tests	57	218
#TM / Testmodels	1	

Fig. 9. OCL Project Statistics

In the next paragraphs we show the OCL formalization of selected consistency, completeness and coverage rules.

a) Consistency: We have defined several consistency criteria to assure that there is no conflicting information in the models.

The criterion `isServiceUnique` in Listing 1 guarantees the uniqueness of service definitions in a system model.

The criterion checks for a specific service whether its name identifier is unique. An additional query `allServicesUnique` checks whether all services in the system model are unique.

The criterion `isAssertionConsistent` in Listing 2 guarantees the consistency of an assertion.

If the definition of pass equals the definition of fail then the definition is inconsistent because it is not possible to compute a meaningful verdict. The definition

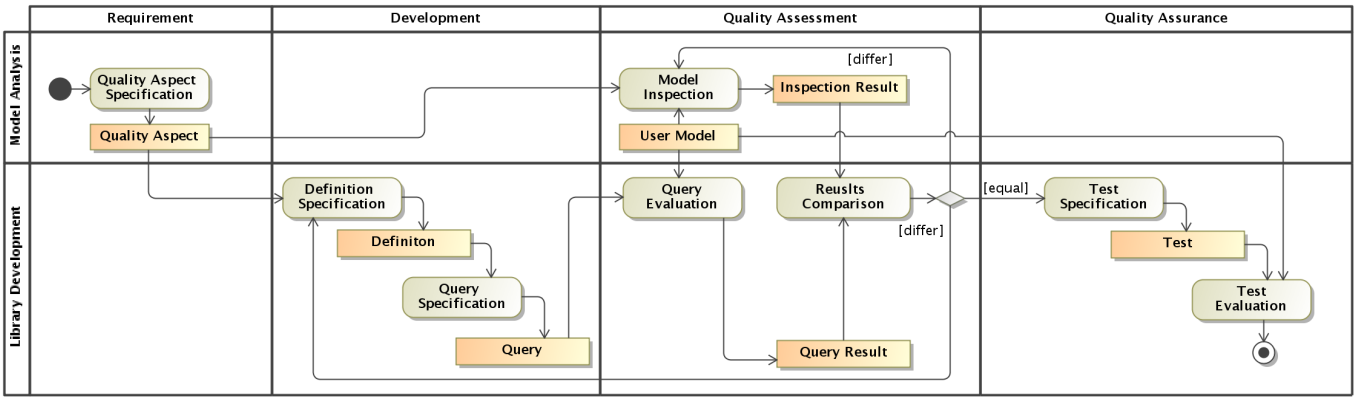


Fig. 8. The model analysis and library development process (from [21]).

```
context TTS::Service
  def isServiceUnique:
    isServiceUnique() : Boolean =
      TTS::Service.allInstances()
        ->select(r|r.base_Class.name=base_Class.name)
        ->size() = 1

context Model
  def allServicesUnique:
    allServicesUnique() : Boolean =
      TTS::Service.allInstances().
        isServiceUnique()->forAll(s | s = true)
```

Listing 1. OCL definition for unique service

```
context TTS::Assertion
  def isAssertionConsistent:
    isAssertionConsistent() : Boolean =
      not (pass = fail)

context Model
  def allAssertionsConsistent:
    allAssertionsConsistent() : Boolean =
      TTS::Assertion.allInstances().
        isAssertionConsistent()->forAll(a | a = true)
```

Listing 2. OCL definition for consistent assertion

`allAssertionsConsistent` checks the criterion `isAssertionConsistent` for all assertions in a specific model.

b) Completeness: We have defined several completeness criteria which guarantee that artifacts contain all essential information. In our respect the completeness of test models is very important because otherwise no meaningful test code can be generated. In Listing 3 a criterion to check the completeness of a test story is defined.

A test story is complete if it has at least one assertion to compute a verdict and a service call to interact with the system. The definition `hasAssertion` checks whether a test story has at least one assertion and the definition

```
context TTS::Teststory
  def hasAssertion:
   hasAssertion() : Boolean =
     base_Activity.allOwnedElements()->select(o |
      o.profileIsTypeOf('Assertion'))->size() > 0

context TTS::Teststory
  def hasServicecall:
    hasServicecall() : Boolean =
      base_Activity.allOwnedElements()
        ->select(o|o.profileIsTypeOf('Servicecall'))
        ->size() > 0

context TTS::Teststory
  def isTeststoryComplete:
    isTeststoryComplete() : Boolean =
          hasAssertion() and hasServicecall()
```

Listing 3. OCL for completeness of test stories

`hasServicecall` checks whether a test story has at least one service call. Finally, the definition `isTestComplete` checks whether a test story has at least one assertion and one service call by invoking `hasAssertion` and `hasServicecall`.

c) Coverage: We have developed several coverage criteria and metrics based on the coverage criteria from testing [10] and model–driven testing [11]. In the next paragraphs we demonstrate how coverage criteria and metrics are implemented in our approach.

As first example we consider the *all requirements coverage* (ARC) defined in [11]. In our context this represents the coverage between the requirement and the test model. In [11] ARC says only that *all requirements are covered*. It is refined in the context of the TTS metamodel to *each requirement has a test story*. “Requirement” in this regard is a model element that applies the `TTS::Requirement` stereotype and “has a test story” means that it has either an action, or an activity defined, i.e., at least one tagged value referring to a test story is set. This informal definition results in the OCL definitions

```
context TTS::Requirement
  def hasTestStory:
    hasTestStory() : Boolean =
      (not self.action.oclIsUndefined()) or
      (not self.activity.oclIsUndefined()) or
      (not self.class.oclIsUndefined())

context Model
  def allRequirementsCoverage:
    allRequirementsCoverage() : Boolean =
      TTS::Requirement.allInstances()
        .hasTestStory()->forAll(e | e = true)
```

Listing 4. OCL definitions for ARC

```
public queries
context Model
  query qRequirementCoverageMetric:
    severity 2
    let result : Integer =
      requirementsWithTestStories()->size()
    message result +' from '
      + TTS::Requirement.allInstances()->size()
      + ' requirements have a test story.('
      + (result/(TTS::Requirement.allInstances()
      ->size().max(1))*100).round()
      +'%)'
    endmessage
endqueries
```

Listing 5. OCL Metrics for ARC

presented in Listing 4.

In the listing both definitions return a value of the type `Boolean` which provides decision support but is not very informative. To gain more information from the model we have defined informative queries that compute a metric and are based on definitions. The query `qRequirementCoverageMetric` in Listing 5 extracts the total number of requirements and the number of all requirements which have a test story assigned. The number of all tested requirements is computed by the query `requirementsWithTests` which is based on the query `hasTestStory` defined in Listing 4. The metrics then computes the ratio between the total number of requirements and the number of tested requirements. It informs to which degree the coverage criterion is satisfied. From the metric we obtain a value between `0` and `1`. Alternatively, the ratio can be expressed as percentage.

For the callmanager example model (see Fig. 5) we obtained the following result: `6 from 6 requirements have a test story.(100%)`. A coverage metrics assigns a number to a coverage criterion measuring the degree of coverage. The coverage metrics `qRequirementCoverageMetric` in Listing 5 measures the requirements coverage by the ratio of requirements with an assigned test story to the overall number of requirements.

Another important coverage criterion is the all services coverage criterion (ASC) which means that *from every service*

```
  def isServiceInvokedByCall:
    isServiceInvokedByCall() : Boolean =
      self.provides -> exists(i
        | i.getAllOperations()
          -> exists(o
          | TTS::Servicecall.allInstances()
            -> collect (s | s.operation)
            -> includes(o)
          )
        )

  def isServiceInvokedByTrigger:
       /* the same as isServiceInvokedByCall
          but with TTS::Trigger.allInstances() */

  def isServiceInvoked:
    isServiceInvoked() : Boolean =
      self.isServiceInvokedByCall() or
      self.isServiceInvokedByTrigger()

context Model
  def allServicesCoverage:
    allServicesCoverage() : Boolean =
      TTS::Service.allInstances()
      .isServiceInvoked()->forAll(e | e = true)
```

Listing 6. OCL Definitions for ASC

at least one operation is invoked in at least one test story. This coverage criterion is defined between the system and the test model. This informal definition results in the OCL definitions shown in Listing 6.

For additional information we have defined a metrics based on the definition `allServicesCoverage` which computes the number of services covered by a set of tests. The metrics prints the number and the ratio of all covered services. For the callmanager example model (see Fig. 6) we obtained the following result: `3 from 3 services have a test story regarding All Services Coverage.(100%).`

C. System Validation

After the test model quality has been validated, the test execution phase starts. Based on RMI adapters, which have been provided by the system developer, test code in Java has been generated from the test model and afterward executed by the test controller. The evaluation of test run be assigning verdicts is based on the explicitly defined assertions, the implicitly defined preconditions and postconditions of the service calls, and errors originated in the test environment. Details on these components are explained in the next section.

Finally, the test results are evaluated. In Fig. 10 the evaluation result is depicted by coloring test cases in the test data table, by coloring test sequence elements, and by annotating test sequence elements.

The evaluation is based on arbitrations which define criteria for a sequence of test results in an OCL–like language defined in [22].

TTS allowed us to perform system wide tests on the application. In a first step, the system model was developed,

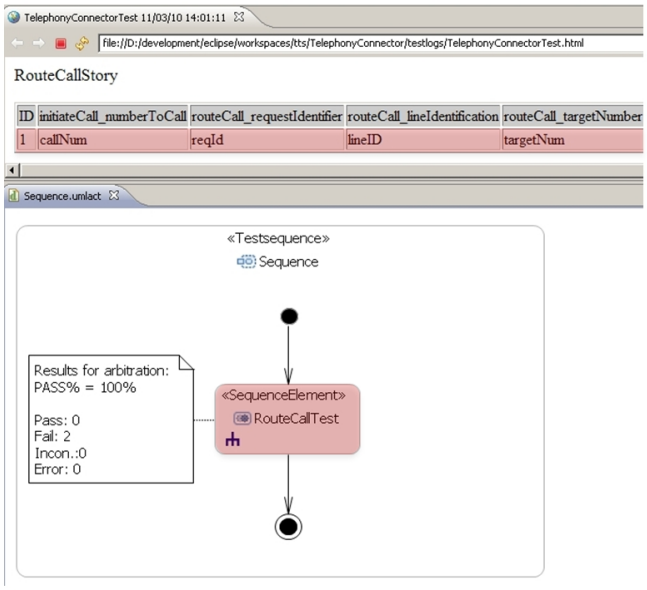


Fig. 10. Test Result of a Test Run

needed for the modeling of the various test cases in a next step. After generating the test code and preparing the test data, finally the tests had been executed against the SUT.

Among the lessons learned during this case study, the three most important are:

1) providing complex test data in a proper way to the testing framework,
2) the communication with asynchronous message exchange patterns, and
3) developing an assertion language capable of iterating complex object structures for test evaluation.

IV. Tool Implementation

Our methodology is tool–driven and based on Telling TestStories providing the modeling and testing environment (presented in Section IV-A) and SQUAM providing the validation environment (presented in Section IV-B).

A. TTS

In this section, we describe the TTS tool implementation that has been applied on the case study in the previous section and developed in an industrial cooperation within the Telling TestStories project [23]

Designed as a set of Eclipse [24] plug-ins, the tool consists of various components setting up the whole environment, to keep a high level of modularity. The architecture of the TTS tool is shown in Fig. 11.

The main components correspond to the activities of our testing methodology depicted in Fig. 2 and are as follows:

The **Modeling Environment** is used for designing the requirements model, the system model and the test model. It processes the workflow activities `Requirements Definition`, `System Model Design`, `Test Model Design`, and `Data Pool Definition`.

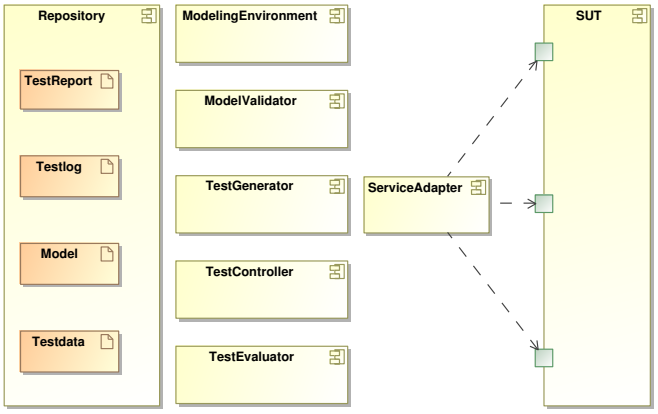


Fig. 11. Components of the TTS Tool

The **Model Evaluator** is based on the SQUAM framework (Section IV-B) and uses OCL as constraint language. It processes the workflow activity `Validation, Coverage, Transformation`.

The **Test Code Generator** generates executable Java code from the test model. It processes the workflow activity `Test Code Generation`.

The **Service Adapters** are used by the test controller to invoke services on the system under test (SUT). They can be created manually or generated automatically depending on the service technology. Adapters correspond to the workflow activity `Adapter Implementation/Generation`.

The **Test Controller** executes the tests against the SUT. It processes the workflow activity `Test Execution`.

The **Test Evaluator** generates test reports and visualizes test results within the models. It corresponds to the workflow activity `Test Analysis`.

In its modularity, the tool is only bound to the Eclipse framework. The various components can be exchanged by more custom triggered extensions as the tool follows established practices (test data modeling in XML, test case modeling in XMI). In the following, the components will be outlined in more detail.

1) Modeling Environment: The requirements, system and test models are denoted as UML models and stored in the XMI format. Our language is defined via an UML profile implementing the metamodel of Section II-C. Therefore stereotypes are used to label model elements and tagged values are used to define additional attributes. The modeling of requirements and the system model is then straightforward with our customized editor.

The modeling of tests is more complicated because test data has to be considered. In a first step, test stories are described using activity diagrams containing control flow elements, service invocations and assertions. Not only test stories, but also test sequences are described using a UML activity diagram, yet contained in another package of the test model. Valid test models which can be checked via the model

```
public interface IAdapter {
 public Object
   invoke(String servicename, Object... arguments);
}
```

Listing 7. Service Adapter Interface

evaluator are the input for the test code generator.

After describing a test, the tool allows for generating test data tables for each test case as a second step of the test modeling process. These test tables follow the approach of the FIT framework [7], which allows for assigning concrete test data (in our case concrete object IDs) to every input parameter and free variable of assertions occurring in a test story. The various test data objects referred in the test tables are provided from a data context, holding instances of the concrete data objects needed for system testing. By making use of the *Inversion of Control* (IoC) container of the Spring Framework [25], TTS allows for modeling complex data objects and not only primitive types.

Hence, these test data tables allow for describing and modeling the execution flow of test cases in a very fine grained and deterministic way, by manually manipulating the object contents. Additionally, it is possible to trace down the erroneous execution of test cases to the test data level by making use of the IoC's user-defined object IDs.

2) Model Evaluator: The model evaluator is based on the SQUAM framework, which is presented in Section IV-B.

3) Service Adapters: The communication of the test controller with the SUT is encapsulated inside generated or manually implemented adapters, tailored to the concrete SUT. By encapsulating service invocations and data mapping, the modeling of test stories and the generation of test code is eased (see sections IV-A1 and IV-A4) because it can be abstracted from technical details.

Currently the tool supports the automatic generation of adapters for services providing a proper WSDL description of their operation interfaces. The WSDL description itself can be generated from a proper system model.

In Listing 7 the root interface for every adapter is denoted. A tailored adapter only has to implement one single method `invoke` to be ready–for–use in the tool environment during the testing process. It support the manual or automatic development of an adapter for different technologies such as web services, RMI or CORBA as much as possible.

4) Test Code Generation: The test code generation constitutes one of the core components of the tool environment. The code generator is implemented based on oAW [26] which is a framework for domain modeling and model driven development, allowing to realize model–to–text transformations as needed in our case.

The test code generator enables to generate executable Java code out of the modeled test stories (activity diagrams). In its implementation the generator visits the contained model elements of each distinct activity diagram in the order of the modeled execution flow and produces equivalent source code (in our case Java). The generated test code is composed of predefined code templates, called and evaluated during the visitation of the various model elements. For pre– and postconditions aspects in the notation of AspectJ [27] are generated to be evaluated as aspects on service calls during the test execution.

The above mentioned evaluation of the code templates focuses on the processing of the applied stereotypes (see section IV-A1) defined as a part of the tool environment. As already mentioned earlier, those stereotypes define element specific tagged values, containing the required information for proper test case generation. To assure that the test model meets the requirements posed by the test code generator, prior to test code generation, the test model is checked for consistency as explained in section IV-A2. The consistency rules assure that on the one hand side, the test model only calls services defined by the SUT and uses data types processable by the SUT. On the other side, those OCL rules are used to check, that the test model is valid, in a sense, that the tagged values of the test specific stereotypes are set.

5) Test Controller: The test controller processes the test code which is executed with concrete test data and logged afterwards. Additionally, the engine also provides a communication interface to the SUT to realize asynchronous service communication, i.e., the execution of a workflow in the SUT whose completion is indicated by a callback method onto the invoking client.

Considered from an architectural point of view, the test controller itself again consists out of various components: **Data Management** Provides the test data objects referenced in the various test data tables (see section IV-A1). Again, the IoC container of Spring is used to provide the test data objects to the test engine. Inside this container the objects are retrieved by their unique object ID used in the test data tables.

Event Handling This component is used by the whole tool environment to process events thrown during test execution, i.e., a *VerdictEvent* to indicate the evaluation of an assertion during test execution. Additionally, this component generates tables as in the FIT framework [7], illustrating the successful or erroneous execution of a set of test data.

Assertion Evaluation Inferring the outcome of a test run is provided by this component. Yet, as in contradiction to JUnit and the like, Telling TestStories allows complex test data to be used, and hence, also this component allows for iterating through complex object structures for test evaluation.

Timing Dealing with asynchronous services requires ensuring that timeouts are met for service responses. This is encapsulated inside this component by ensuring that responses to specific service invocations receive the test controller within a pre–specified duration. Responses are assigned to corresponding service invocations by their method signatures.

```
context Model query checkIsValidTestModel:
let result:Boolean =
 if Package.allInstances()->
  any(o|o.profileIsTypeOf('Test')).oclIsUndefined()
 then false
 else
  Package.allInstances()->
  any(o|o.profileIsTypeOf('Test')).isValidTestModel()
 endif
 message result endmessage
```

Listing 8. OCL Query for Testmodel Validity

In the execution phase, the test controller passes through three states. In an initial state, the test workflow is parsed and according to its contents, *test tasks* are generated encapsulating the modeled test cases. After completion of creating the tasks the engine enters its second, main state in which the tests are executed against the SUT. In a third and final state, the engine generates the above mentioned result table containing the test outcomes, after all events have been processed.

6) Test Evaluator: The test evaluator is responsible for evaluating the results of a test run. As in the FIT framework, our test evaluator colors test case lines in test tables green, red or yellow depending on the test result. If a failure can be assigned to a specific model element, our tool is able to color it in the activity diagram. We have also integrated high–level test reporting based on BIRT [28] which generates different types of graphical test summary reports.

B. SQUAM

In this section, we describe the SQUAM tool implementation that has been used for the model evaluation within the TTS framework. SQUAM (Systematic QUality Assessment of Models) is an integrated framework for UML/OCL–based model development and OCL–based quality analysis.

It provides support for consistency and coverage checks in OCL, and supports the definition and generation of high–quality system and test models. Coverage checks guarantee that the test models are complete with respect to the system model and the requirements model. Additionally, coverage checks are useful exit criteria for test generation. Consistency checks guarantee model validity which is a prerequisite for test code generation. Therefore the test code generator uses the model evaluator to check test model validity prior to test code generation. In Listing 8 a sample top–level query for assuring test model validity is denoted.

The SQUAM tool has a plug–in architecture incorporating in–house developed and existing open source solutions. There are two editions of SQUAM, a community and a professional edition. The community edition is integrated into the TTS framework, whereas the professional one provides features that can be used to obtain an integrated and user–friendly OCL development process for criteria and metrics within the TTS framework. Below we describe the part of the SQUAM framework integrated into the TTS framework.

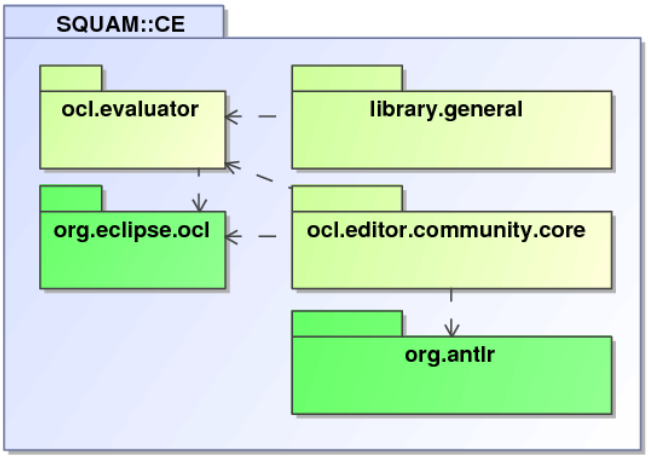


Fig. 12. Architecture of the SQUAM Community Edition Tool

The community edition is the core of the framework. It provides the basic features for writing and editing OCL expressions supporting definitions, (running) queries, (running) unit tests and their documentation (OCLDoc). It consists of several plug–ins (see Fig. 12): 3 in–house plug–ins: core, general library, and OCL evaluator and 2 third plug–ins: Eclipse OCL evaluator and Antlr.

The **core** (*ocl.editor.community.core*) forms the backbone of the SQUAM application. It contains the basic functionality like two–step parsing of libraries: using **Antrl** (*org.antlr*) for pre–parsing our OCL extensions and the **OCL evaluator** (*ocl.evaluator*) with **Eclipse OCL evaluator** (*org.eclipse.ocl*) for parsing standard OCL. Moreover, the core provides basic features for editing OCL expressions (like syntax highlighting, code completion or code formatting), managing UML/ECORE/XML models (and meta–models) and UML profiles (loading, removing and creation of qualified names).

The **general library** (*library.general*) defines an abstract OCL library as an Eclipse extension point to access OCL definitions and queries. The basic principle of the general library is to give third party plug–ins the ability to retrieve all definitions and queries for further processing.

There are two integration points of the SQUAM framework into the TTS framework, one for the evaluation of interactive checks and one for the evaluation of automatic checks. For the interactive checks, the validation view from the core is used, and for the automatic checks, the OCL evaluator is used.

V. Related Work

In this section we present tool–based testing frameworks, model validation techniques, and coverage criteria related to TTS.

A. Tool–based Testing Frameworks

Model–based testing approaches always have a methodological and a tool aspect [11]. There are already some industrial

tools available [29]. TDE/UML [30], a tool suite for test generation from UML behavioral models, is related to our approach but focuses on GUI–based systems whereas TTS focuses on service–centric systems.

FIT/Fitnesse [7] is the most prominent framework which supports system test–driven development of applications allowing the tabular specification, observation and execution of test cases by system analysts. Our framework is due to the tabular specification of test data based on the ideas of FIT/Fitnesse but integrates it with model–based testing techniques.

Although test sheets [31] define a fully tabular approach, support for model–driven testing is missing there.

In [32] a model–driven system testing approach and a tool implementation that enables test engineers to graphically design complex test cases based on METAFrame Technologies' Application Building Center [33] has been defined. The approach is similar to TTS but not based on UML and its generic profiling mechanism.

The PLASTIC framework [34] provides a collection of tools for online and offline testing both functional and non–functional properties of service–oriented applications. Some of the tools are model–based but the tools are not integrated and do not follow a model–driven testing approach as in the TTS tool implementation.

B. Model Validation

Validating consistency of UML models has received greater attention by researchers in recent years [35], but completeness has also been addressed [16]. Even the interplay between consistency and completeness has been investigated [36].

Only a small number of approaches such as [37] consider consistency and completeness of test models, i.e., behavioral descriptions of operations. But consistency and completeness between requirements, system and test models as in TTS is not considered.

C. Coverage Criteria

Comprehensive collections of coverage criteria are defined in [11] and [10]. In [10] a coverage–driven approach to software testing is discussed. There are four different types of testing and corresponding coverage criteria distinguished, i.e. graph coverage, logical expression coverage, input space partitioning, and syntax–based coverage. In our integrated approach, we use artifacts of our metamodel as sources for test coverage. Behaviors provide graphs, decisions provide logical expressions and types or services provide partitions of the input space. We do not have explicit grammar definitions and therefore syntax–based coverage is not relevant in our approach.

A collection of structural UML–based coverage criteria for class diagrams, sequence diagrams, communication diagrams, state machines, activity diagrams and use case diagrams is provided in [38]. It includes coverage criteria for state machines and activity diagrams from [39] where test generation from UML specifications is discussed. Our approach provides specific model–based coverage criteria for service–centric systems supporting the manual or automatic test definition.

VI. Conclusions and Future Work

In this article, we have outlined the model–driven and tabular system testing methodology Telling TestStories and its tool implementation. TTS is based on traceable requirements, system, and test models which are validated even before test code is generated and executed. The TTS tool used for model development and system validation, plus the SQUAM tool used for model validation are explained by a case study from the telecommunication domain. The TTS tool and the SQUAM tool consist of a set of Eclipse plug–ins [24] and are integrated to implement the TTS methodology.

Prior to the use of TTS the system test design has been done ad–hoc in an unsystematic way mainly by testers themselves. TTS allows for designing tests in an effective way because its intuitive graphical and tabular notation supports the design of tests that can be validated by consistency, completeness and coverage checks. The TTS methodology naturally integrates domain experts and customers into the process of formal and executable test design. The integration of domain experts and customers may be helpful to reveal specific test scenarios that would not have been detected otherwise. TTS is also efficient because the tests can be defined on an abstract visual level with tool support. After the initial effort of system model design the advantages of TTS can be applied. Our implementation of the methodology has shown that the checks provide additional support for the validation of the requirements, system, and test model, but also raises the failure detection rate due to the higher test quality. In the model of the callmanager case study we found inconsistencies and incompletenesses that have been detected with our criteria and removed afterwards. With our validation checks the effectiveness and efficiency of the approach has been improved because the quality of test models is higher and failures are detected earlier. The coverage criteria and metrics provide useful information to all stakeholders whether additional tests have to be defined manually or not. Our validation checks are defined statically on the metamodel and do not support the dynamic simulation of a model. But due to our experience this is replaced by early test execution in iterative software testing and therefore not a severe restriction compared to dynamic approaches like model–checking which need additional modeling effort and more knowledge in formal modeling. Additionally, the information provided in this process supports the system engineers who are not experts in test design when defining tests.

Based on research results and user feedback we have planned further extensions to our methodology, the tool, and its application.

In the case study at hand functional and performance requirements are considered. But TTS will also be applied to test other non–functional requirements, e.g., security requirements.

We have already tested positive security requirements with TTS [40], but testing negative security requirements with TTS has not been considered yet. So far coverage criteria in TTS only check the quality of the test model as adequacy criteria but are not applied for the generation of test cases as selection criteria. Our OCL–based coverage criteria can be applied to integrate selective test generation into the TTS methodology.

The TTS tool is already quite mature and on its way to a practically usable open–source tool for model–driven system testing of service–centric systems [41]. For this step the usability of the tool and the interoperability to integrate TTS with other testing tools has to be improved. For instance, the usability of modeling tests can be improved by an additional textual representation of test models which is synchronized with the graphical representation, to support fast text–based editing of test models. TTS is suitable for arbitrary service–centric systems and therefore many application domains are arising. Applications to an industrial service–centric system from the health care domain and to the upcoming paradigm of cloud computing which can also be considered as a specific service–centric system are planned.

ACKNOWLEDGEMENTS

This work was largely performed as part of the project MATE under support of the FWF (project number P17380), as part of the project QE LaB – Living Models for Open Systems under support of the FFG (project number FFG 822740), and as part of the SecureChange project under support of the EU (project number ICT-FET-231101).

REFERENCES

[1] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J.-P. Richter, M. Voß, and J. Willkomm, “A Method for Engineering a True Service-Oriented Architecture,” 2008, ICEIS 2008.

[2] OASIS Standard, “Web Services Business Process Execution Language Version 2.0 - OASIS Standard,” 2007, http://docs.oasis-open.org/wsbpel/2.0/.

[3] W3C, “Web Services Choreography Description Language Version 1.0,” 2005, http://www.w3.org/TR/ws-cdl-10/.

[4] G. Canfora and M. D. Penta, “Service-oriented architectures testing: A survey,” in *ISSSE*, ser. Lecture Notes in Computer Science, A. D. Lucia and F. Ferrucci, Eds., vol. 5413. Springer, 2008.

[5] P. Baker, P. Ru Dai, J. Grabowski, O. Haugen, I. Schieferdecker, and C. E. Williams, *Model-Driven Testing - Using the UML Testing Profile*. Springer, 2007.

[6] OMG, *MDA Guide Version 1.0.1*, www.omg.org/docs/omg/03-06-01.pdf [accessed: June 30, 2011].

[7] R. Mugridge and W. Cunningham, *Fit for Developing Software: Framework for Integrated Tests*. Prentice Hall, 2005.

[8] M. Felderer, P. Zech, F. Fiedler, and R. Breu, “A Tool-based methodology for System Testing of Service-oriented systems,” in *The Second International Conference on Advances in System Testing and Validation Lifecycle (VALID 2010)*, 2010, pp. 108–113.

[9] M. Felderer, B. Agreiter, R. Breu, and A. Armenteros, “Security Testing By Telling TestStories,” 2010, Modellierung 2010.

[10] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge, UK: Cambridge University Press, 2008.

[11] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[12] M. Felderer, F. Fiedler, P. Zech, and R. Breu, “Flexible Test Code Generation for Service Oriented Systems,” 2009, QSIC’2009.

[13] M. Felderer, R. Breu, J. Chimiak-Opoka, M. Breu, and F. Schupp, “Concepts for Model–Based Requirements Testing of Service Oriented Systems,” 2009, IASTED SE’2009.

[14] ISO/IEC, *Information technology – open systems interconnection – conformance testing methodology and framework*, 1994, international ISO/IEC multi–part standard No. 9646.

[15] OMG, *OMG Systems Modeling Language*, 2007, http://www.omg.org/docs/formal/2008-11-01.pdf.

[16] Lange, C. F. J. and Chaudron, M. R. V. , “An empirical assessment of completeness in UML designs,” in *In Proc. of the 8th International Conference on Empirical Assessment in Software Engineering (EASE‘04)*, 2004.

[17] M. Lanza, R. Marinescu, and S. Ducasse, *Object-Oriented Metrics in Practice*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[18] OMG, *Object Constraint Language Version 2.0*, 2006, http://www.omg.org/docs/formal/06-05-01.pdf [accessed: June 30, 2011].

[19] J. Chimiak-Opoka, “OCLLib, OCLUnit, OCLDoc: Pragmatic Extensions for the Object Constraint Language,” in *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, Colorado, USA, October 4-9, 2009, Proceedings. LNCS 5795*, A. Schuerr and B. Selic, Eds. Springer, 2009, pp. 665–669.

[20] Object Mentor, “Junit,” 09 2009, http://www.junit.org/.

[21] J. Chimiak-Opoka, B. Agreiter, and R. Breu, “Bringing Models into Practice: Design and Usage of UML Profiles and OCL Queries in a showcase,” in *Proc. of the 16th Int. Conf. on Information and Software Technologies,IT’2010*, 2010, pp. 265–273.

[22] J. Chimiak-Opoka, S. Loew, M. Felderer, R. Breu, F. Fiedler, F. Schupp, and M. Breu, “Generic Arbitrations for Test Reporting,” 2009, IASTED SE’2009.

[23] “Telling TestStories,” http://teststories.info [accessed: June 30, 2011].

[24] “Eclipse,” http://www.eclipse.org/ [accessed: June 30, 2011].

[25] “Spring,” http://www.springsource.org/ [accessed: June 30, 2011].

[26] “oAW,” http://www.openarchitectureware.org/ [accessed: June 30, 2011].

[27] “AspectJ,” http://www.eclipse.org/aspectj/ [accessed: June 30, 2011].

[28] “BIRT,” http://www.eclipse.org/birt/ [accessed: June 30, 2011].

[29] H. Götz, M. Nickolaus, T. Roßner, and K. Salomon, *iX Studie Modellbasiertes Testen*. Heise Zeitschriften Verlag, 2009.

[30] J. Hartmann, M. Vieira, H. Foster, and A. Ruder, “A UML–based approach to system testing,” *ISSE*, vol. 1, no. 1, 2005.

[31] C. Atkinson, D. Brenner, G. Falcone, and M. Juhasz, “Specifying High-Assurance Services,” *Computer*, vol. 41, 2008.

[32] T. Margaria and B. Steffen, “Lightweight coarse-grained coordination: a scalable system-level approach,” *STTT*, vol. 5, no. 2-3, 2004.

[33] B. Steffen and T. Margaria, “Metaframe in practice: Design of intelligent network services,” in *Correct System Design, Recent Insight and Advances*. London, UK: Springer-Verlag, 1999, pp. 390–415.

[34] A. Bertolino, G. D. Angelis, L. Frantzen, and A. Polini, “The plastic framework and tools for testing service-oriented applications,” in *ISSSE*, ser. Lecture Notes in Computer Science, A. D. Lucia and F. Ferrucci, Eds., vol. 5413. Springer, 2008, pp. 106–139.

[35] M. Elaasar and L. Briand, “An Overview of UML Consistency Management,” Department of Systems and Computer Engineering, University of Ottawa, Tech. Rep. SCE-04-18, 2004.

[36] D. Zowghi and V. Gervasi, “On the interplay between consistency, completeness, and correctness in requirements evolution,” *Information and Software Technology*, vol. 45, no. 14, pp. 993 – 1009, 2003.

[37] Amit Paradkar and Tim Klinger, “Automated Consistency and Completeness Checking of Testing Models for Interactive Systems,” *Computer Software and Applications Conference, Annual International*, vol. 1, pp. 342–348, 2004.

[38] J. McQuillan and J. Power, “A Survey of UML-Based Coverage Criteria for Software Testing,” National University of Ireland, Maynooth, Tech. Rep., 2005.

[39] A. J. Offutt and A. Abdurazik, “Generating Tests from UML Specifications,” in *UML*, R. B. France and B. Rumpe, Eds. Springer, 1999, pp. 416–429.

[40] M. Felderer, B. Agreiter, and R. Breu, “Security Testing by Telling TestStories,” in *Modellierung 2010*, 2010.

[41] M. Felderer and P. Zech, “Telling teststories – a tool for tabular and model-driven system testing,” *Testing Experience*, vol. 12, 2010.

[42] A. D. Lucia and F. Ferrucci, Eds., *Software Engineering, International Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures*, ser. Lecture Notes in Computer Science, vol. 5413. Springer, 2009.

Quality-Oriented Design of Services

Michael Gebhart, Sebastian Abeck
Research Group Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{gebhart | abeck} @kit.edu

***Abstract*—With the shift to a service-oriented architecture, goals concerning the IT of an organization, such as an increased flexibility and maintainability, are expected to be attained. For this purpose, the building blocks of the service-oriented architecture, the services, have to be designed that certain quality attributes, such as loose coupling or autonomy, are fulfilled. Existing design processes for services name these quality attributes and consider them as important. However, they do not explain their usage within a design process in order to create services that verifiably fulfill these quality attributes. This article shows an enhancement of existing design processes that on the one hand comprehensibly describes how to derive service designs from artifacts of the business analysis and on the other hand integrates quality attributes in order to enable a verifiably quality-oriented design of services. The approach is applied to design services for a system at the Karlsruhe Institute of Technology that guides students across the campus of the university.**

Keywords-service design; design process; quality attribute; design decision; soaml

I. INTRODUCTION

Today, several companies structure their information technology (IT) service-oriented, where functionality is encapsulated and provided in form of services. The shift to a service-oriented architecture is mostly associated with the achievement of goals concerning the IT, such as an increased flexibility and maintainability [3, 4, 24].

To support the achievement of these goals, quality attributes could be identified, a service within a service-oriented architecture should fulfill. Wide-spread attributes are a unique categorization, loose coupling, autonomy and discoverability of a service. After the analysis of the business, the services are designed before they are implemented. Thus, during the so-called service design phase, the IT architect has to design the services in a way that the implementation results in services that fulfill these quality attributes. The service design phase consists of two sub phases: the identification and specification phase [5]. Within the identification phase, service candidates as preliminary services and their dependencies are identified [3, 5]. Service candidates consist of operation candidates that represent preliminary operations. They constitute the structural basis for the following specification phase. During this phase, the service designs for each service are modeled. They describe the service interfaces for accessing the provided functionality and the service components that perform the functionality.

Existing design processes in the context of service-oriented architectures, as introduced by Erl [3], Engels et al. [4], the Rational Unified Process [19] for Service-Oriented Modeling Architecture (RUP SOMA) [5, 6, 7], and the Service Oriented Architecture Framework (SOAF) [8], focus on the steps that are necessary to design services at a high level of abstraction. They even name an excerpt of quality attributes and consider them as important. However, they do not describe how the design of the services has to be performed in order to verifiably fulfill the quality attributes. Additionally, the design processes are mostly only described abstractly, so that a detailed description about how to derive service designs based on a standardized modeling language from artifacts of the business analysis is missing. Other work, as introduced by Erl [9, 22], Engels et al. [4], Reussner et al. [10], Josuttis [11], Maier et al. [12, 13], Perepletchikov et al. [14, 15], Hirzalla et al. [16], Choi et al. [17] and SoaML [18], focuses on quality attributes a service should fulfill. However, the authors of this work do not address how these quality attributes can be used within a design process in order to create services with these quality attributes.

This article introduces an enhancement for design processes as they are introduced in existing work in order to verifiably design services with certain quality attributes. For this purpose, the derivation of service designs from artifacts of the business analysis is described in detail. Additionally, an iterative analysis and revision phase is added subsequently to the identification and specification of services for ensuring the fulfillment of certain quality attributes. During the analysis phase, the quality attributes of the current service designs are evaluated by measuring quality indicators that represent the quality attribute and give hints about their current value. Afterwards, if the quality attributes do not correspond to the desired values, the revision phase is performed. This phase consists of two steps. First, the design flaws within the current service designs are identified as they give the IT architect hints about the model elements within a service design that should be revised. Afterwards, action alternatives are derived and presented to the IT architect. They represent design decisions the IT architect should consider in order to create revised and improved service designs.

To illustrate our approach, services of a service-oriented system that guides students across the campus of the

Karlsruhe Institute of Technology (KIT) are designed. This system has its origin in a service-oriented surveillance system developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [33, 34] we already designed services for [1, 2, 32]. The services are designed with respect to loose coupling, autonomy, unique categorization, and discoverability as desired quality attributes. The service designs are modeled using the Service-oriented architecture Modeling Language (SoaML) [18] as standardized UML profile [38] and metamodel for describing and formalizing service-oriented architectures. Though SoaML is a very new UML profile and metamodel and still under development, it is becoming increasingly accepted and employed.

The article is organized as follows: Section 2 presents the fundamentals in the context of design processes, quality attributes, and modeling service designs. In Section 3, the entire approach for a quality-oriented design of services is introduced and exemplarily applied for designing services of the service-oriented KITCampusGuide. Section 4 concludes the article and offers suggestions for future research.

II. FUNDAMENTALS

A quality-oriented design of services consists of three essential parts that have to fit together: First, the design process as framework for the entire quality-oriented design has to be specified. The design process describes the necessary phases within the design process and specifies the derivation of elements created during the business analysis into elements of the service design phase and transformations within the design phase. Additionally, the design process describes when and how to consider quality attributes to guarantee the fulfillment of quality requirements. The availability of measureable quality attributes constitutes the second part of a quality-oriented design of services. The quality attributes, such as loose coupling and autonomy, have to be described in a way that the IT architect can verifiably measure them. The modeling of service designs represents the third and final part of a quality-oriented design. The created service designs have to be modeled, i.e. formalized, that it is possible to evaluate them with respect to quality attributes and derive implementation artifacts as starting point for the implementation phase. Existing work mostly focuses on one of these three aspects.

A. Design Processes

In [3], Erl introduces the service-oriented analysis and design phases that describe the steps necessary to design services. According to Erl, first, service candidates, the included operation candidates, and dependencies between these service candidates are identified. Afterwards, for each service candidate an entire service design can be created that specifies the service in detail. Even though the identification and specification is described, the comprehensible transformation of artifacts that have been created during the business analysis phase into service candidates and afterwards into service designs is missing. Also quality attributes, such as loose coupling, are considered as important but explained textually only. Information how to evaluate a formalized service design regarding these quality attributes in order to create service designs with verifiable quality attributes is not provided. The service candidates and service designs are also described using an own informal notation. There is no formal language used.

In [4] Engels et al. describe a method to derive services from prior described business services. For each business service a service within the service-oriented architecture is created. But also in this case, some quality attributes are only mentioned as important and not explained in a way that they could be measured on a formalized service design. Additionally, the design process does not explain how to use the quality attributes to gain services with certain quality attributes. Engels et al. also do not use a formal language to model created services. The services are mostly described textual.

The Rational Unified Process for Service Oriented Modeling and Architecture (RUP SOMA) as introduced by IBM [5, 6, 7] provides a detailed description about how to derive preliminary service candidates from prior modeled business processes and how to transfer these candidates into final service designs. However, also in this case quality attributes are only mentioned and not further considered. For modeling service candidates and service designs the proprietary UML profile for software services is applied [25]. But in current work [26], there is also a usage of the standardized SoaML introduced.

In [8], the Service Oriented Architecture Framework (SOAF) is introduced. This framework describes steps that result in services with the prescribed quality attribute business it alignment. The process does not consider own preferences. Information about how to transfer artifacts created during the business analysis phase into artifacts of the service design phase is missing and for modeling service designs an own notation is used.

B. Measureable Quality Attributes

Other work focuses on the description of quality attributes and their measurement. Erl presents in [9, 22] design principles and patterns for services. These principles and patterns are explained in detail, but the concrete measurement on formalized service designs is not explained. Also the integration into an entire design process is missing.

Similarly, Engels et al. [4], Reussner et al. [10], Josuttis [11], Maier et al. [12, 13], Perepletchikov et al. [14, 15], Hirzalla et al. [16], Choi et al. [17] and SoaML [18] introduce important and partially even measurable quality attributes. But also in this case, the description of them is addressed. How to use these quality attributes in order to create quality-oriented service designs is not further explained. In [2] we presented the evaluation of service designs based on SoaML. This work already helps IT architects to evaluate service designs according to the informal description of quality attributes as described in existing work. In [1] we introduced how this measurement can be used for supporting design decisions within a design process in order to create improved service designs.

C. Modeling Service Designs

According to Erl [3, 9, 22, 23] and IBM [5, 6, 7], the design process consists of two phases, the identification and the specification of services. During the identification phase, service candidates as preliminary services are identified. In a next step, final service designs are specified. Thus, to support the design process with a formal modeling language, the modeling of service candidates and service designs is necessary. Erl does not use any formal language whereas IBM uses an own proprietary UML profile for software services [25]. In the meanwhile, SoaML [18] has emerged as a standardized UML profile for modeling services within a service-oriented architecture. However, the SoaML standard does not explain how to use this modeling language within a design process and how to evaluate service designs that have been created using this language. SoaML supports several elements of service-oriented architectures. In the following, we introduce the modeling elements that are of interest in this article for modeling service candidates and service designs.

1) Modeling Abstract Capabilities: In SoaML a Capability element exists that represents a collection of capabilities. These capabilities describe the functionality a service provides. The Capability element is a stereotyped UML class, whilst the capabilities inside are modeled using operations. Additionally, dependencies between these Capability elements can be specified. They are modeled by means of usage dependencies and represent that a group of capabilities requires other capabilities to be performed. The following figure shows three Capability elements and their dependencies.

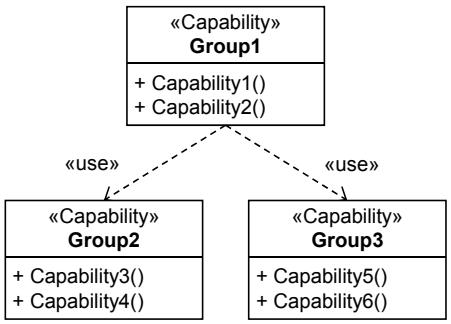


Figure 1. Modeling abstract capabilities

2) Modeling Service Designs: According to Erl [3, 23], Engels et al. [4], and IBM [6], a service design includes the design of a service interface and of a service component. The service interface describes the service and the service component realizes its functionality. To model a service interface, in SoaML the ServiceInterface element exists in form of a stereotyped UML class. A ServiceInterface describes the operations the service provides for potential service consumers. This is specified by a UML interface that is realized by the ServiceInterface. Additionally, it includes a specification of operations a service consumer has to provide for example in order to receive callbacks. For that purpose a second interface has to be created that is used by the ServiceInterface. A ServiceInterface also allows the description of participating roles in form of UML parts and of an interaction protocol. Latter can be specified by an UML Activity that is added as OwnedBehavior to the ServiceInterface. An exemplary service interface is shown in the following figure. This service interface describes that one operation is provided and one operation is required to be provided by the service consumer in order to receive callbacks. The interaction protocol specifies the order of operation calls for gaining a valid result.

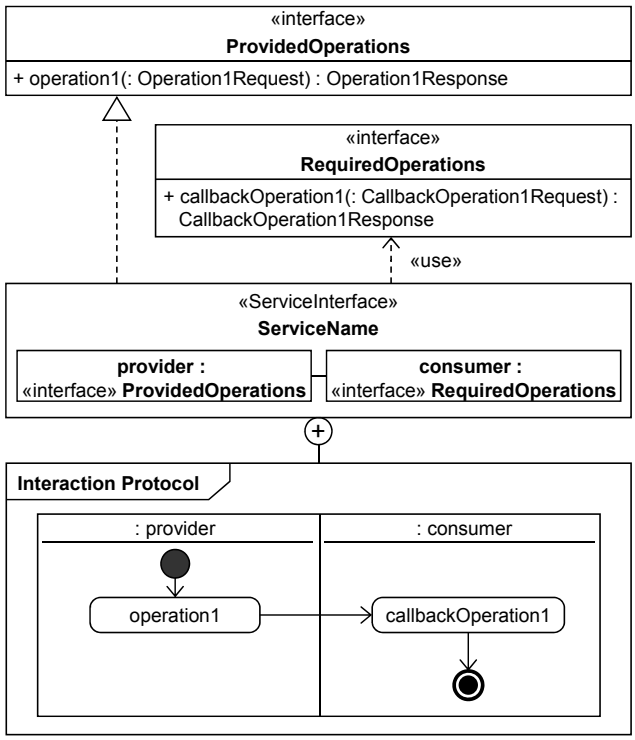


Figure 2. Modeling a service interface

When calling one of the provided or required operations, messages are exchanged. These messages are described by MessageType elements that extend the UML dateTypes. A MessageType represents a document-centric message and can contain several dataTypes. In context of specifying service designs, also these messages have to be described. In the following, an example for a message is depicted.

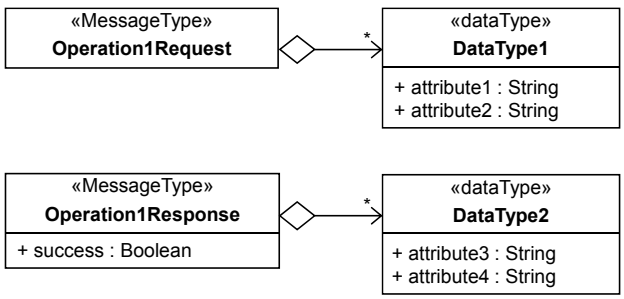


Figure 3. Modeling message types

Finally, for each service design the service component has to be specified that realizes the provided functionality. For service components, SoaML includes the Participant element that represents an organization, system, or software component. For modeling a Participant in UML, the UML component can be extended by an according stereotype. For each provided service a ServicePoint is added and typed by the describing ServiceInterface. If the service component requires other services to fulfill its functionality, RequestPoints can be added to the service component. They specify required services and are also typed by the describing ServiceInterface element. To model the internal behavior of the service component, for each provided operation an OwnedBehavior in form of an UML Activity can be added. For each ServicePoint and RequestPoint a UML Partition is added that is typed by this ServicePoint or RequestPoint and for each operation of a service a CallOperationAction is assigned to the according Partition. An AcceptEvent describes that the service component waits for a callback operation being invoked. For internal functionality that is not performed by required services an OpaqueAction is added to the Partition that represents the ServicePoint. An exemplary service component is depicted in Figure 3. The service component provides one service and requires two services.

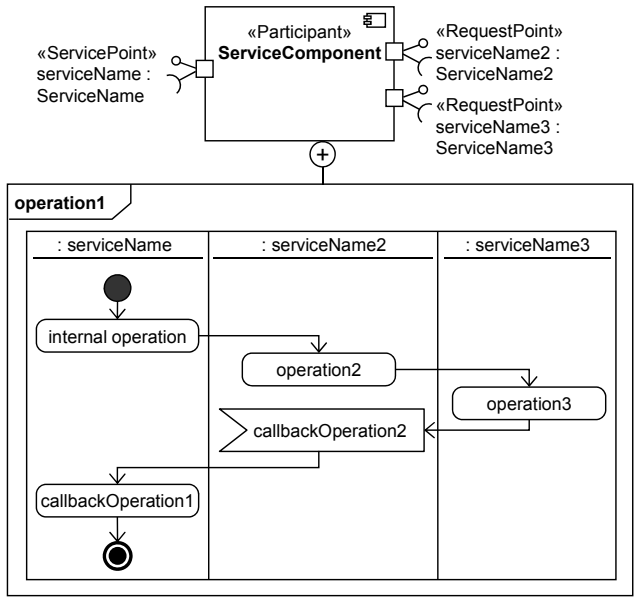


Figure 4. Modeling a service component

D. Discussion

The analysis of the existing work shows that each work focuses mainly on one aspect. Work focusing on design processes describes the necessary steps within the process. The concrete derivation of service candidates and final service designs with a concrete modeling language is not addressed. Also quality attributes are only considered as important but it is not obvious how to measure them and how to use this knowledge to create quality-oriented service designs.

Other work focuses on exactly these quality attributes and shows metrics that enable their measurement. But in this case, it is not obvious how to measure the quality attributes on a standardized modeling language, such as SoaML. The textual descriptions have to be interpreted and the formalized metrics require information that is mostly not part of service designs. Finally, the quality attributes are not integrated into an entire design process. Thus, there exist only detailed descriptions of quality attributes but their usage to create service designs with certain quality attributes is missing.

Modeling languages for service designs, such as the UML profile for software services and SoaML, focus on modeling elements and do not provide any information about how to use this language within an entire design process. Only IBM gives some hints about how to derive artifacts from prior modeled business processes [6, 25, 26]. But how to use this language to model service designs with certain quality attributes and how to evaluate a modeled service design regarding these attributes is not explained.

Our quality-oriented service design approach combines these different approaches. We use the design processes as described in existing work and add additional phases for ensuring certain quality attributes. During these phases, our approach to evaluate service candidates and services designs based on SoaML [2] is applied. Afterwards, the service candidates and service designs are revised in order to improve chosen quality attributes [1]. Additionally, we add detailed information about how to derive service candidates in SoaML from modeled artifacts of the business analysis phase and how to transfer service candidates into service designs also based on SoaML. As result, a guideline is provided that enables the IT architect to comprehensibly create service designs with certain quality attributes.

III. Quality-Oriented Design of Services

The design process of this article enhances design processes discussed in Section 2 by details about how to derive service candidates from artifacts of the business analysis phase and service designs from service candidates. Furthermore, subsequent phases for ensuring the fulfillment of quality attributes are added.

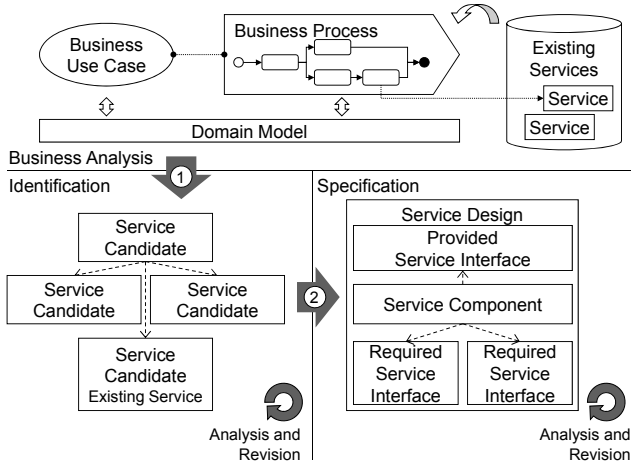


Figure 5. Design process

The design process requires a prior analysis of the business. This means that a domain model, business use cases and business processes are created. These artifacts are then transferred into preliminary service candidates as part of the identification phase. Afterwards, these service candidates are analyzed in regard to quality attributes. If the current attributes do not correspond to the preferred values, a subsequent revision is performed. During the specification phase, first, the service candidates are transferred into preliminary service designs. Also in this phase, afterwards, the service designs are analyzed in regard to quality attributes and if required revised. As result, service designs are created that fulfill certain quality attributes. Since the created artifacts of the business analysis phase constitute the basis for the design process, they are explained in the following.

A. Scenario

To illustrate the artifacts of the business analysis and the subsequent design process, in this article the human-centered environmental observation domain referring to the network-enabled surveillance and tracking system as introduced by the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [33, 34] is treated. In this context the KITCampusGuide, a project at the Karlsruhe Institute of Technology (KIT) to provide a guide for students, lecturers and guest, is chosen as scenario. A person can ask for a person or a room on the campus of the university and the KITCampusGuide calculates the route. The following figure illustrates the scenario in action.

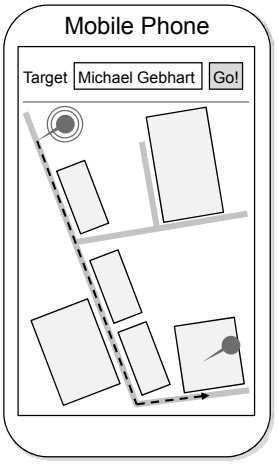


Figure 6. KITCampusGuide in action

The goal is, to create service designs for this scenario that fulfill the quality attributes of an unique categorization, loose coupling, autonomy and discoverability as introduced in [2].

B. Business Analysis

During the business analysis phase, the following threw artifacts are created: The domain model captures all relevant concepts of the domain and their relations. It determines the relevant terms when designing the business processes and also unifies the terminology of the services. The business use cases describe the external visible business services that are expected to be supported by IT. The business processes describe the processes behind the business use cases, thus describes their implementation.

For modeling the domain, an ontology can be used. In this case, the ontology is created using the Web Ontology Language (OWL) [30] by means of Protégé [37]. As illustration, we choose a notation similar to the OntoGraf in Protégé. For each concept a rectangle is depicted and the relations between these concepts are represented by lines between them. If a concept or relation is available in various languages, this information can be added as labels. Each label can have a suffix specifying the language of the label, such as "@de" for German. An excerpt of the domain model for the human-centered environmental observation is depicted in Figure 7.

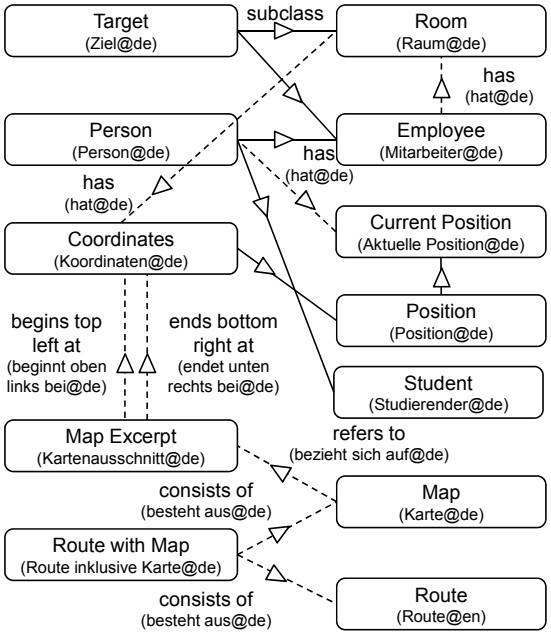


Figure 7. Excerpt of the domain model

The business use cases can be seen as entry points for the service design phase. They describe the externally visible business services [4] that are supposed to be supported by IT [6]. As notation, the UML profile for business services can be applied [20, 27, 28]. The use case describes that a student requests a route from his current position to a room or an employee. Additionally to the route, the map that covers the route is returned.

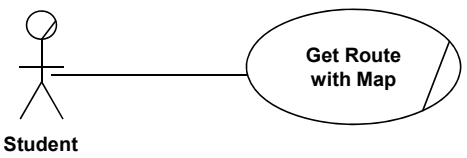


Figure 8. Considered business use case

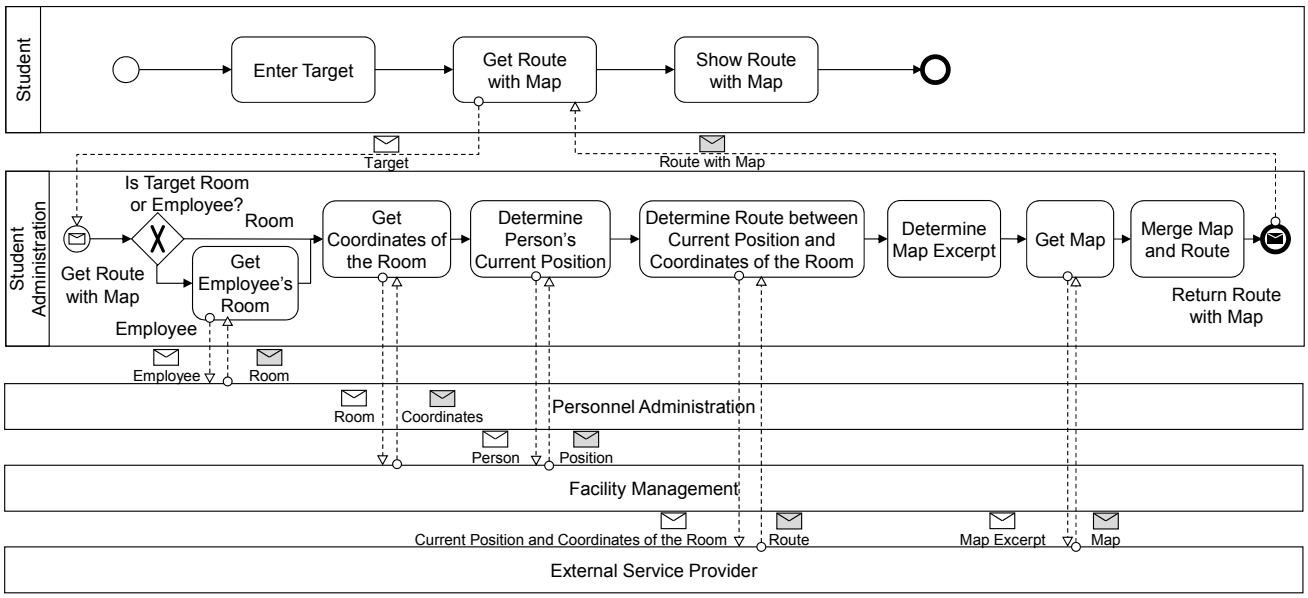


Figure 9. Business process to get a route with a map

Since each business use case or business service is realized by a business process [4, 6], the underlying business process has to be modeled. For this purpose the business process model and notation (BPMN) [29] can be used. The business process that realizes the considered business use case is depicted in Figure 9.

C. Service Design

The service design phase starts with the identification of service candidates. According to Figure 5, the identification includes two steps: First, preliminary service candidates are derived from artifacts of the business analysis phase. Afterwards, these service candidates are analyzed regarding quality attributes and if necessary they are revised in order to improve the quality attributes. To derive the service candidates, the business processes are considered. For our scenario the business process as shown in Figure 9 is used to derive service candidates.

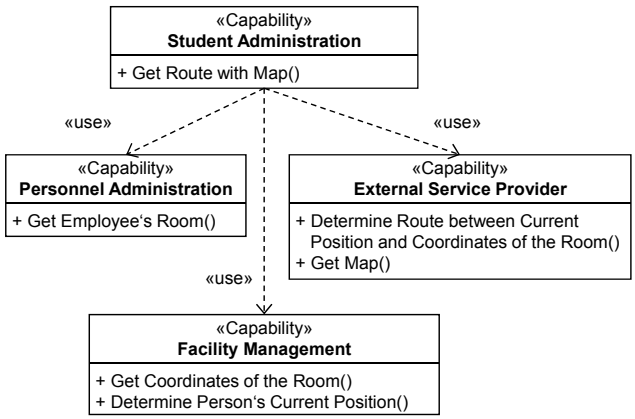


Figure 10. Derived service candidates

For each pool representing an organizational unit a new service candidate is created and for each message flow between the pools, an operation candidate is added. Since a service candidate represents a group of abstract capabilities, the Capability element of SoaML corresponds with the understanding of service candidates and thus can be used for modeling service candidates and their dependencies. Figure 10 shows the derived service candidates for our scenario.

To evaluate these service candidates regarding a unique categorization, loose coupling, autonomy and discoverability, the evaluable quality indicators as introduced in [2] are used and extended. For service candidates only a subset of the quality indicators is evaluable. The following table shows the quality indicators for each service candidate. A "+" represents that the quality indicator is optimal and a "-"describes that there is need for improvement. If a quality indicator is not evaluable, a "0" is set.

TABLE I. EVALUATION OF SERVICE CANDIDATES

Quality Indicator	SA	PA	FM	ESP
Unique Categorization				
Division of Business-Related and Technical Functionality	+	+	+	+
Division of Agnostic and non-Agnostic Functionality	+	+	-	+
Data Superiority	0	+	+	+
Usage of Common Business Entities	+	+	-	-
Loose Coupling				
Compensation	0	0	0	0
Autonomy				
Dependencies	-	+	+	+
Overlapping Functionality	+	+	+	+

Since the service candidates only provide business-related functionality, the quality indicator to divide business-related and technical functionality is optimal for all service candidates. The division of agnostic and non-agnostic functionality can be improved for the Facility Management service candidate. Whilst the functionality to get coordinates of a room is very agnostic functionality, the determination of person's current position is very process specific and will not be used in many further scenarios. Since all service candidates are explicitly responsible for the management of used business entities, they fulfill the requirement for data superiority. Since the Student Administration does not manage any business entity, the data superiority is not evaluable for this service candidate. The operations of the Facility Management and of the External Service Provider do not use common business entities. The former uses the business entities room and person and both business entities can exist for their own. In case of the External Service Provider also different and independent business entities are used by the operations. Since there are no state-changing operations performed by any service candidate, there is no compensating functionality required. The Student Administration depends on other service candidates, thus the dependency quality indicator is not optimal. Since every service candidate is explicitly responsible for a functional scope, there is no functional overlap.

In a next step, the IT architect has to revise the service candidates, in order to improve their quality attributes. For that purpose, in a first step, design flaws in form of weak points have to be identified. They represent parts of the service candidate model that are responsible for a specific non-fulfilled quality attribute. Since each quality indicator refers to one main artifact within service candidates, this information can be used to identify the weak points. The following table lists the quality indicators and the model elements that represent the responsible part and thus the weak point.

TABLE II. WEAK POINTS IN SERVICE CANDIDATES

Quality Indicator	Weak Point
Division of Business-Related and Technical Functionality	If at least the half of the operation candidates provide business-related functionality, then the operation candidates that provide technical functionality represent the weak point, else the operation candidates that provide business-related functionality.
Division of Agnostic and non-Agnostic Functionality	If at least the half of the operation candidates provide agnostic-related functionality, then the operation candidates that provide non-agnostic functionality represent the weak point, else the operation candidates that provide agnostic functionality.
Data Superiority	The operation candidates of other service candidates that manage business entities that are also managed by own operations represent the weak point.
Usage of Common Business Entities	First, the biggest set of used and depending business entities is determined. The operation candidates that use business entitites that are not part of this set represent the weak point.
Compensation	The operation candidates that provide state-changing functionality and do not have a compensating operation candidate represent the weak point.
Dependencies	The operation candidates that require other service candidates represent the weak point.
Overlapping Functionality	The operation candidates with overlapping functionality to operation candidates of other service candidates represent the weak point.

The table above helps the IT architect to analyze the derived service candidates and to identify weak points that should be revised. Thus, for the Facility Management service candidate the operation candidate for determining person's current position represents a weak point, thus a design flaw, as shown in Figure 11.

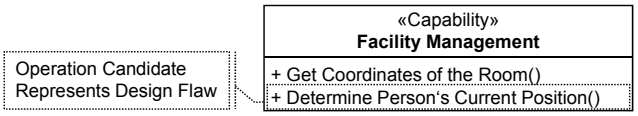


Figure 11. Identified design flaw

In a next step, the IT architect has to decide, how to revise this service candidate in order to fix the weak point. To support his decision, possible design decisions are analyzed and associated with the prior identified weak point. The quality indicators base on elements of service candidates or service designs that can represent weak points. Design decisions on the other hand influence these elements. This enables the association of design decisions with quality indicators. This association can be used to identify design decisions that affect certain quality indicators and can such be considered in order to improve weak points. The following figure shows the approach.

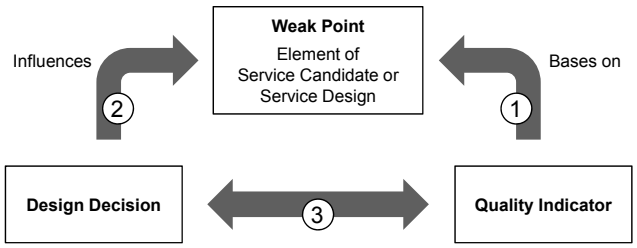


Figure 12. Approach for design decision identification

Possible design decisions can be taken from existing work that describes how to design services. Afterwards, these design decisions have to be adapted for a revision of service designs. For example, Erl describes in [3] that it is necessary to decide the operation candidates within a service candidate. Thus, a revision design decision is whether to move an operation candidate into another service candidate or not. For service candidates there are no further design decisions. During the specification phase there will be some more. The design decision whether to move an operation candidate can be further refined. The decision tree for our scenario including the various concrete action alternatives (AA) is shown below.

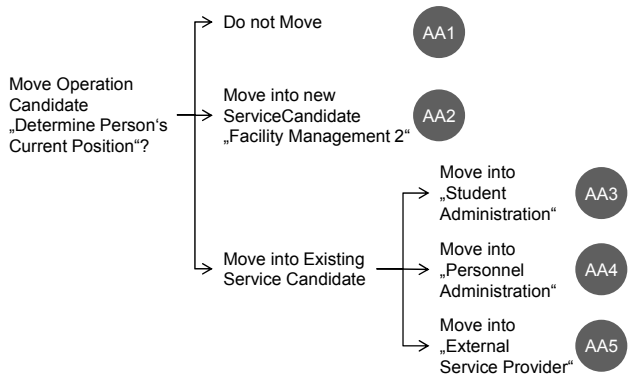


Figure 13. Action alternatives

It is important to notice that only possible concrete action alternatives can be considered that result in valid service designs. This means that for example the deletion of an operation candidate is not considered for this decision results in service designs that do not fulfill the business requirements. This restriction enables the convincing evaluation of different action alternatives.

Afterwards, each of the different action alternatives can be evaluated with regard to the quality attributes. This means that the service candidates are evaluated for each action alternatives. The following table shows this evaluation for each service candidate and the action alternatives two to five. For each quality indicator and action alternative, it is displayed whether it improves (↗), gets worse (↘) or does not change (➔) compared to the action alternative one that has been evaluated in Table 1. For the optional service candidate Facility Management 2, the new value is shown for there is no existing value that could be compared.

TABLE III. EVALUATION OF ACTION ALTERNATIVES

Quality Indicator		SA	PA	FM	FM_2	ESP
Division of Business-Related and Technical Functionality	AA2	➔	➔	➔	+	➔
	AA3	➔	➔	➔	n.a.	➔
	AA4	➔	➔	➔	n.a.	➔
	AA5	➔	➔	➔	n.a.	➔
Division of Agnostic and non-Agnostic Functionality	AA2	➔	➔	↗	+	➔
	AA3	➔	➔	↗	n.a.	➔
	AA4	➔	↘	↗	n.a.	➔
	AA5	➔	➔	↗	n.a.	➔
Data Superiority	AA2	0	➔	➔	0	➔
	AA3	0	➔	➔	n.a.	↗
	AA4	0	➔	➔	n.a.	➔
	AA5	0	➔	➔	n.a.	➔
Usage of Common Business Entities	AA2	➔	➔	↗	+	➔
	AA3	↘	➔	↗	n.a.	➔
	AA4	➔	➔	↗	n.a.	➔
	AA5	➔	➔	↗	n.a.	➔
Compensation	AA2	0	0	0	0	0
	AA3	0	0	0	n.a.	0
	AA4	0	0	0	n.a.	0
	AA5	0	0	0	n.a.	0
Dependencies	AA2	↘	➔	➔	+	➔
	AA3	➔	➔	➔	n.a.	➔
	AA4	➔	➔	➔	n.a.	➔
	AA5	➔	➔	➔	n.a.	➔
Overlapping Functionality	AA2	0	0	0	0	0
	AA3	0	0	0	n.a.	0
	AA4	0	0	0	n.a.	0
	AA5	0	0	0	n.a.	0

According to this table, action alternative two and four are the most improving ones. Now, the IT architect has to decide how to weight the quality indicators. In our case we decide that dependencies are less harmful. Thus, the IT architect chooses action alternative two. For the other weak points this procedure is repeated adequately. As result, service candidates are created that fulfill the four quality attributes best. The service candidates are displayed in the following figure.

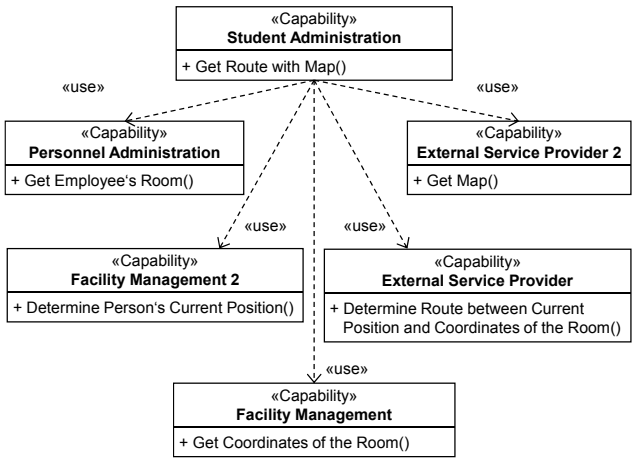


Figure 14. Revised service candidates

Subsequently to the identification phase, the specification follows. During this phase, first, preliminary service designs are derived and afterwards, they are revised if necessary. To derive the service designs, each service candidate is transferred into one ServiceInterface with one realized interface containing the provided operations and one interface containing required operations for receiving callbacks. The operation candidates are directly added as operations within the realized interface and if there is an end event within the corresponding business process that calls an operation, this operation call is added within the interface containing the required operations. Figure 15 shows the derived service interface for the service candidate Student Administration.

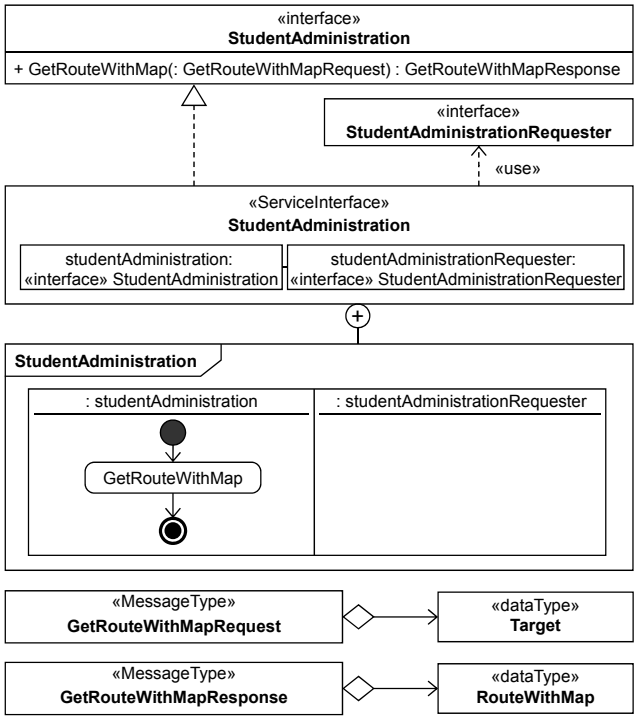


Figure 15. Derived service interface

For each operation adequate message types are created. They are named according to the operation with the suffix Request or Response. For the concepts exchanged as messages within the business process data types are generated and assigned to the respective message type. The interaction protocol of the service interface can be derived by the exchanged messages of the business process. All these artifacts are kept within a package named after the service interface. During this step already some information, such as potential naming conventions, can be considered. For example white spaces in the names of the service interface and the operations can be removed if this is a convention. Another convention could be the translation into another language. For example, if the business has been analyzed in German and it is convention to use English for service design artifacts, the artifacts can be translated according to the domain model and its labels containing the names of the concepts in various languages.

Additionally to the service interface, a service component is generated. The service component contains one ServicePoint typed by the derived ServiceInterface. If the service candidate where the service component was derived from requires other service candidates, appropriate RequestPoints are added. The following figure shows the service component for the Student Administration service candidate. The service component provides one service and thus includes one ServicePoint. To realize its functionality, ir requires five other services that are added as RequestPoints. Both the ServicePoint and the RequestPoints are named and typed by the ServiceInterface that describes the service. The internal behavior of the service component equals the business process, thus it is not further depicted.

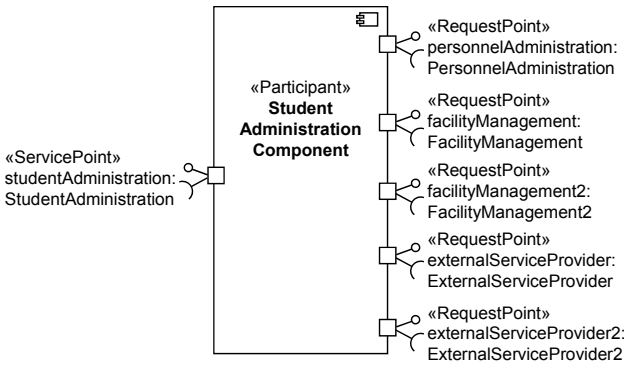


Figure 16. Derived service component

This systematic derivation is performed for all service candidates. Afterwards, the analysis and revision phase follows, similar to analysis and revision within the identification phase. During the specification phase, further quality indicators can be considered that were not of interest during the identification phase. The quality indicators are again taken from our previous work [2]. The service design for the Student Administration is evaluated in the following table.

TABLE IV. EVALUATION OF SERVICE DESIGNS

Quality Indicator	SA	PA	FM	FM_2	ESP	ESP_2
Unique Categorization						
Division of Business-Related and Technical Functionality	+	+	+	+	+	+
Division of Agnostic and non-Agnostic Functionality	+	+	+	+	+	+
Data Superiority	0	+	+	0	+	+
Usage of Common Business Entities	+	+	+	+	+	+
Discoverability						
Functional Naming of the Service Interface	+	+	+	+	+	+
Functional Naming of the Roles	+	+	+	+	+	+
Functional Naming of the Operations	+	+	+	+	+	+
Functional Naming of the Parameters	+	+	+	+	+	+
Functional Naming of the Data Types	+	+	+	+	+	+
Naming Convention Compliance regarding the Service Interface	-	-	-	-	-	-
Naming Convention Compliance regarding the Roles	+	+	+	+	+	+
Naming Convention Compliance regarding the Operations	-	-	-	-	-	-

Naming Convention Compliance regarding the Parameters	+	+	+	+	+	+
Naming Convention Compliance regarding the Data Types	+	+	+	+	+	+
Information Extent	+	+	+	+	+	+
Loose Coupling						
Compensation	0	0	0	0	0	0
Asynchronity	+	+	+	+	+	+
Complexity of Common Data Types	+	+	+	+	+	+
Operation Abstraction	+	+	+	+	+	+
Data Type Abstraction	+	+	+	+	+	+
Autonomy						
Dependencies	-	+	+	+	+	+
Overlapping Functionality	+	+	+	+	+	+

According to this table, most of the quality indicators are already optimal due to the fact that the service designs were derived from already revised service candidates. Since the artifacts were generated from the service candidates that came from the business processes, also the naming of the artifacts already follows functional terms. Also most of the naming conventions have already been considered during the transfer of service candidates into service designs. Only the operations still do not follow our naming conventions. They should begin with a lowercased letter. This information could be considered during the derivation of the service designs too. But we consciously disregarded this convention in order to illustrate that the naming of the artifacts is an important aspect for the discoverability and even though some naming conventions have been already considered during the derivation of the service designs it should be reviewed and evaluated afterwards. Another example of naming conventions that cannot be fully regarded in an automatic transformation is the language of the artifacts. If the business has been analyzed in another language, such as German, the service candidates and the derived service designs are also in German. If the naming convention for the design artifacts is English, a translation is necessary. The domain model can contain several languages, so some information can already be used for an automatic transformation. However, mostly there is manual effort required for possibly not all concepts within the domain model are described in several languages. The quality indicators help to remind the IT architect that naming conventions, such as the correct language, have to be considered. The naming of the service interfaces is also not optimal. The reason is that a service interface should be named after what it is doing and numerations, such External Service Provider 2, should be avoided. A common naming convention for services that manage a certain business entity is to name this service interface after the managed business entity. For example, if the service manages the business entity room, a room service should be provided. Since all service designs contain all necessary information, the information extent is optimal. The asynchronity is optimal too, for there are no long-running operations that should be provided asynchronous. The complexity of common data types requires that all common data types are simple data types only. Since for each service design an own package has been generated, the complex data types are in separated packages and the service designs do not share any complex data types. On the one hand this requires a transformation of data types even if they are named equal, but on the other hand this supports the loose coupling. Since the operations hide the implementation and do not show any implementation details and the data types are only business-driven and not technical, the abstraction is also optimally fulfilled. The table shows that due to the systemic derivation of service designs from already revised and business-driven service candidates a lot of quality indicators are already optimally fulfilled. But the sum of quality indicators helps the IT architect to ensure that he has not forgotten any important aspect.

To revise the service designs, again the design flaws have to be identified and afterwards action alternatives have to be presented. The following table shows the weak points for the quality indicators considered during the specification of the service designs. At this, also the weak points that have been used during the identification phase are presented again, however they are adapted for the modeling elements within service designs instead of service candidates.

TABLE V. WEAK POINTS IN SERVICE DESIGNS

Quality Indicator	Weak Point
Division of Business-Related and Technical Functionality	If at least the half of the operations provide business-related functionality, then the operations within the realized interface of the service interface that provide technical functionality represent the weak point, else the operations that provide business-related functionality.
Division of Agnostic and non-Agnostic Functionality	If at least the half of the operations provide agnostic-related functionality, then the operations within the realized interface of the service interface that provide non-agnostic functionality represent the weak point, else the operations that provide agnostic functionality.
Data Superiority	The operations within the realized interface of other service interfaces that manage business entities that are also managed by own operations represent the weak point.
Usage of Common Business Entities	First, the biggest set of used and depending business entities is determined. The operations within the realized interface of the service interface that use business entitites that are not part of this set represent the weak point.
Functional Naming of the Service Interface	The name attribute of the service interface represents the weak point.
Functional Naming of the Roles	The name attribute of the not functionally named roles represents the weak point.
Functional Naming of the Operations	The name attribute of the not functionally named operations represents the weak point.

Functional Naming of the Parameters	The name attribute of the not functionally named parameters represents the weak point.
Functional Naming of the Data Types	The name attribute of the not functionally named data types represents the weak point.
Naming Convention Compliance Regarding the Service Interface	The name attribute of the service interface represents the weak point.
Naming Convention Compliance Regarding the Roles	The name attribute of the not functionally named roles represents the weak point.
Naming Convention Compliance Regarding the Operations	The name attribute of the not functionally named operations represents the weak point.
Naming Convention Compliance Regarding the Parameters	The name attribute of the not functionally named parameters represents the weak point.
Naming Convention Compliance Regarding the Data Types	The name attribute of the not functionally named data types represents the weak point.
Information Extent	The service interface represents the weak point.
Compensation	Operations within the realized interface of the service interface that provide state-changing functionality and do not have a compensating operation represents the weak point.
Asynchronity	The communication modes of the CallOperationActions within the interaction protocol that correspond to operations with long-running functionality and are not asynchronous yet represent the weak point.
Complexity of Common Data Types	The data types that are complex and commonly used represent the weak point.
Operation Abstraction	The operations that are not abstract represent the weak point.
Data Type Abstraction	The data types that are not abstract represent the weak point.
Dependencies	The Operations within the realized interface of the service interface that require operations of other services represent the weak point.
Overlapping Functionality	The Operations within the realized interface of the service interface with overlapping functionality to operations of other services represent the weak point.

According to this table, within our scenario, especially the name attributes of the different artifacts are marked as design flaws for they are responsible for the insufficient naming. In order to remove these weak points, action alternatives have to be identified in form of design decisions that are applicable during a revision and enable the improvement of the derived service designs. For service designs the following design decisions can be identified. They are again derived from existing design processes [3, 4, 5, 6, 7] and adapted for a revision of existing service designs.

TABLE VI. DESIGN DECISIONS DURING THE SPECIFICATION PHASE

Design Decision	Description
Moving an Operation	Similar to the design decision during the identification phase, the IT architect has to decide whether or not to move an operation from one interface that is realized by a service interface into an interface realized by another service interface.
Renaming a Service Interface	Especially for influencing the discoverability, the IT architect can rename a service interface, i.e. the name attribute is changed. In this case, concrete action alternatives cannot be identified for the set of possible renamings is unlimited.
Renaming a Role	Similarly to the decision before, this design decision influences the name attribute of a role within a service interface.
Renaming an Operation	Whilst the naming of operation candidates was not of interest, the naming of the operations directly influences the discoverability. This design decision changes the name attribute of an operation.
Renaming a Parameter	This design decision changes the name attribute of a parameter that is used within an operation.
Renaming a Data Type	The IT architect has to decide, whether or not to rename a data type in order to increase the understanding and thus the discoverability.
Changing the Communication Mode of an Operation	The communication mode of an operation within an interaction protocol determines, whether the operation can be called asynchronously or not. The IT architect can change this communication mode subsequently.
Changing a Data Type	The data types represent information that can be used within parameters of operations. These data types can be changed.
Changing Parameter Types of an Operation	The parameter types of an operation represent the information that is exchanged between a service consumer and a service provider when a certain operation is called. The IT architect can change this amount and kind of information.

In our scenario, especially the renaming of the operations and of the service interfaces are identified as action alternatives for they affect the name attributes of these artifacts that have been identified as weak points. Finally, the revised service designs can be created. Additionally, during the revision, further details of the data types can be added, such as detailed attributes. The following figure shows the revised service interface for the student administration and an excerpt of the used data types. The used KML data type represents the Keyhole Markup Language (KML) [43] that has been developed by Google for Google Earth. In the meanwhile, KML is a wide-spread markup language for geological data that has been standardized by the Open Geospatial Consortium (OGC). The other service designs are analyzed and revised equivalently. Also in these cases, mostly the names of the artifacts are changed and details are added to the data types for the service designs were derived from already revised service candidates.

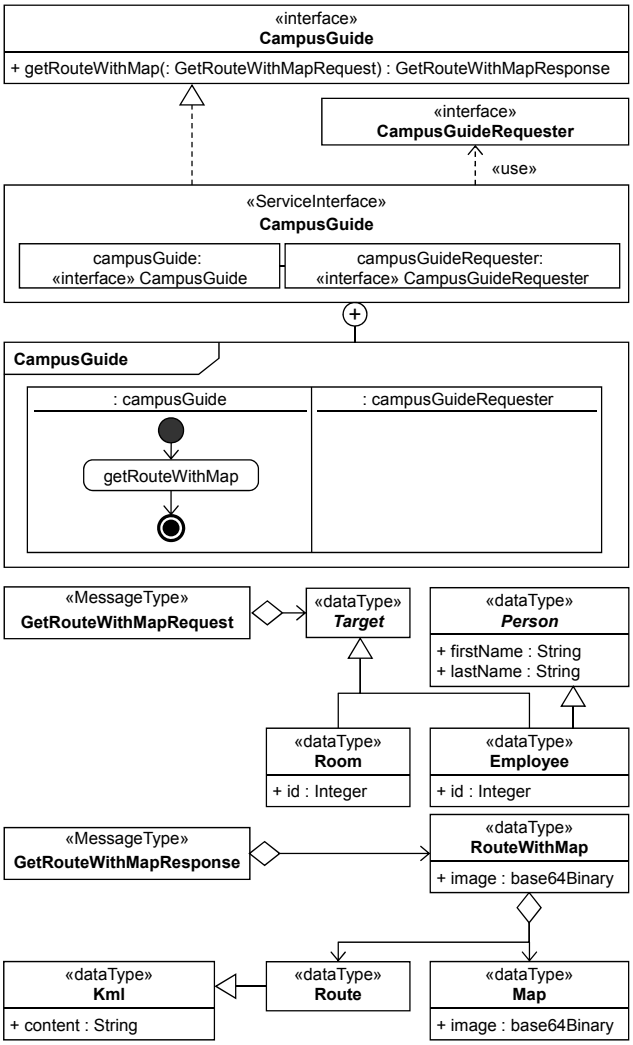


Figure 17. Revised service interface

The resulting service component for the Student Administration, now named Campus Guide, is shown below. The ServicePoints and RequestPoints have been renamed after the newly named service interfaces. The internal behavior, described as UML Activity, includes one partition for each ServicePoint and RequestPoint. Each partition that represents a RequestPoint contains CallOperationActions for the operations provided by this external service. Also in this case, the newly named operation names are used. Within the partition that represents the ServicePoint, two OpaqueActions are included. They represent internal behavior that is performed by the service component itself and is not called by external services. Since the OpaqueActions were not part of the revision, they are still named after the activities within the business process. The IT architect has to decide whether to rename these OpaqueActions. However, since they were not identified as weak points, their naming does not influence one of the considered quality attributes. Thus, a renaming would only increase the consistency within the design artifacts.

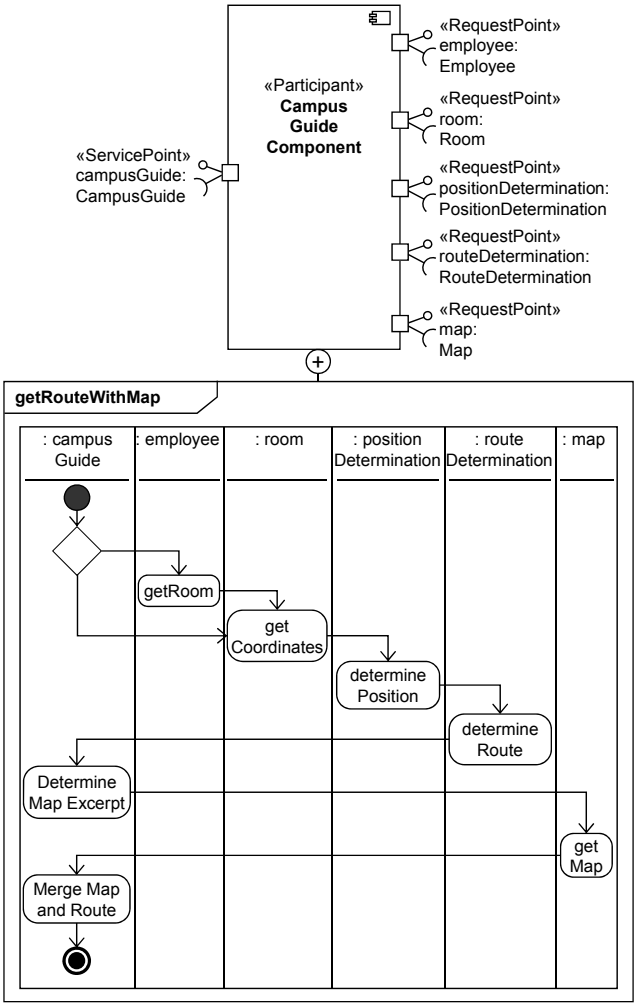


Figure 18. Revised service component

D. *Recursive Continuation*

Till now, the design process focused on the interaction between various pools. In a next step, the activities within a pool are considered in order to increase the flexibility of the service components and their implementation. This means that the created service components are further decomposed into internal service components by recursive continuing the design process. This enables that functionality within the service components can be easily provided for external consumers or can easily be replaced by functionality that is provided by external service providers.

For this purpose, instead of the invoked activities and interaction between pools, the activities within one pool are considered and with these activities the design process is performed equivalently. First all activities within one pool are collected within one service candidate that can be named after the pool with the suffix Internal. Afterwards, the service candidate is revised according to the quality attributes and their quality indicators as described above. For our scenario, the following figure shows the derived and revised internal service candidates.

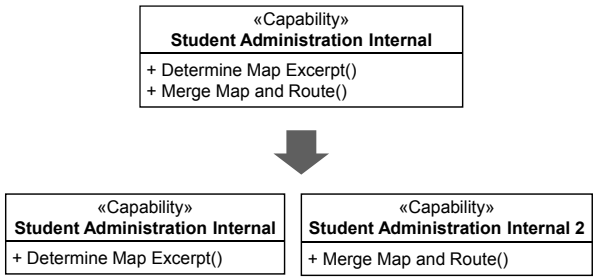


Figure 19. Internal service candidates

Afterwards, for each internal service candidate a service design is created and revised. The resulting service components are assigned to the superior service component. Since within a business process internal and external functionality is composed, a composition component is added to the superior service component. Within SoaML these internal service components are connected using the ServiceChannel element. A ServiceChannel can either be a delegation of an external ServicePoint to an internal ServicePoint respectively an internal RequestPoint to an external RequestPoint, or an assembly of two internal service components by connecting one ServicePoint with one RequestPoint. The revised service component for the Student Administration with its internal service component is depicted in Figure 20. Since there is no further decomposition necessary, the design process ends with this recursive continuation. As result, service designs are created that support the business requirements and fulfill certain quality attributes.

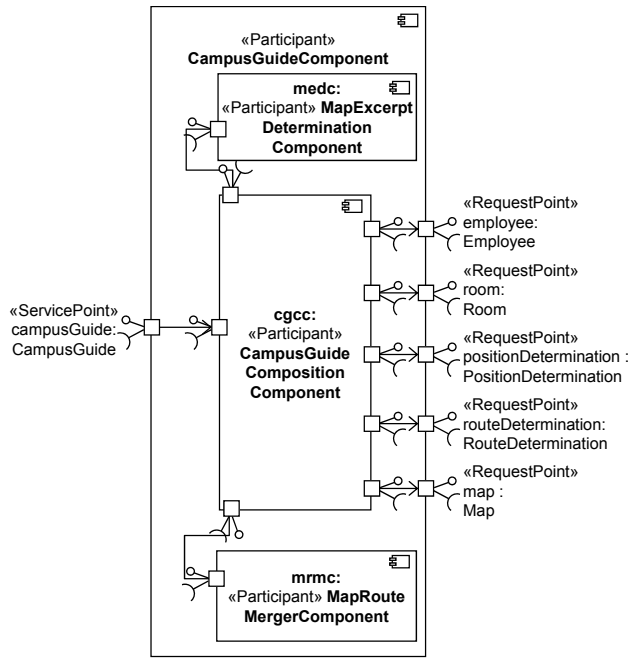


Figure 20. Internal service candidates

IV. Conclusion and Outlook

In this article, we presented an approach for a quality-oriented design of services. The approach enhances existing design processes with a detailed description about how to transfer artifacts of the business analysis phase into service candidates and how to transfer these service candidates into service designs. Additionally, an iterative analysis and revision phase ensures the fulfillment of certain quality attributes. Due to the subsequent analysis and revision, our approach can be used in combination with other design processes and allows also the revision of already existing service designs. Due to the subsequent recursive continuation of the design process, one of the frequent questions when to use pools and when to use lanes when modeling business processes with BPMN is also solved. The recursive continuation results in the same service designs regardless of whether pools or lanes have been used. The service designs support the business requirements and fulfill a desired set of quality attributes.

The detailed description of transformations of artifacts enables the IT architect to comprehensibly derive service designs from prior created artifacts of the business analysis. Additionally, instead of only naming important quality attributes, the design process also helps to ensure their fulfillment. The usage of SoaML as language to model service candidates and service designs enables the integration of our approach into existing tool chains. SoaML represents an emerging standard for modeling service-oriented architectures. Its availability as XMI [42] enables the usage in any UML-capable development tools, although some vendors already provide built-in SoaML support.

To illustrate our approach, services of a service-oriented campus guide system as it is developed at the Karlsruhe Institute of Technology (KIT), the KITCampusGuide, have been designed. The services for this scenario could be derived comprehensibly and fulfill verifiably the quality attributes of a unique categorization, loose coupling, discoverability and autonomy. The system has its origin in the Network Enabled Surveillance and Tracking (NEST) system, developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [21, 22]. Currently, the approach is also applied for the domain campus management in order to create a catalog of services for universities and their administrative processes. These services follow national and international specifications that came up with the Bologna Process [31]. Additionally, the approach is applied at the Personalized Environmental Service Configuration and Delivery Orchestration (PESCaDO) project [35, 36], a project co-funded by the European Commission, in order to design the required services with verifiably fulfilled quality attributes.

In parallel to this article, we work on a formalization of the quality attributes and their quality indicators. Our goal is to improve our guidelines for IT architects so that the quality indicators can be measured exactly, either manually or partially even automatically. The automatically evaluable quality indicators are then formalized using the Object Constraint Language (OCL) [39] in order to enable

integration into existing development tools and thus realize a tool support for the quality-oriented design of services. Finally, we work on derivation rules to transfer design attributes into implementation artifacts [21] using technologies, such as the Service Component Architecture (SCA) [40] and the Business Process Execution Language (BPEL) [41]. While in both cases, already a lot of good work has been published, verification and if necessary an adaptation for SoaML and the semantic of service designs is required. This enables the integration of the quality-oriented design process into an entire development process.

REFERENCES

[1] M. Gebhart, M. Baumgartner, and S. Abeck, "Supporting service design decisions", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 76-81.

[2] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck, "Evaluation of service designs based on soaml", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 7-13.

[3] T. Erl, Service-Oriented Architecture – Concepts, Technology, and Design, Pearson Education, 2006. ISBN 0-13-185858-0.

[4] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J.-P. Richter, M. Voß, and J. Willkomm, Quasar Enteprise, dpunkt.verlag, 2008. ISBN 978-3-89864-506-5.

[5] IBM, "RUP for service-oriented modeling and architecture", IBM Developer Works, http://www.ibm.com/developerworks/rational/downloads/06/rmc_soma/, 2006. [accessed: January 04, 2011]

[6] U. Wahli, L. Ackerman, A. Di Bari, G. Hodgkinson, A. Kesterton, L. Olson, and B. Portier, "Building soa solutions using the rational sdp", IBM Redbook, 2007.

[7] A. Arsanjani, "Service-oriented modeling and architecture – how to identify, specify, and realize services for your soa", IBM Developer Works, http://www.ibm.com/developerworks/library/ws-soa-design1, 2004. [accessed: January 04, 2011]

[8] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: An architectural framework for service definition and realization", 2006.

[9] T. Erl, SOA – Principles of Service Design, Prentice Hall, 2008. ISBN 978-0-13-234482-1.

[10] R. Reussner and W. Hasselbring, Handbuch der Software-Architektur, dpunkt.verlag, 2006. ISBN 978-3898643726.

[11] N. Josuttis, SOA in der Praxis – System-Design für verteilte Geschäftsprozesse, dpunkt.verlag, 2008. ISBN 978-3898644761.

[12] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, and T. Winterberg, „Die soa-service-kategorienmatrix", SOA-Spezial, Software & Support Verlag, 2009.

[13] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, and T. Winterberg, „Was macht einen guten public service aus?", SOA-Spezial, Software & Support Verlag, 2009.

[14] M. Perepletchikov, C. Ryan, K. Frampton, and H. Schmidt, "Formalising service-oriented design", Journal of Software, Volume 3, February 2008.

[15] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-Oriented design", Australian Software Engineering Conference (ASWEC 2007), 2007.

[16] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, "A metrics suite for evaluating flexibility and complexity in service oriented architecture", ICSOC 2008, 2008.

[17] S. W. Choi and S. D. Kimi, "A quality model for evaluating reusability of services in soa", 10th IEEE Conference on E-Commerce Technology and the Fifth Conference on Enterprise Computing, E-Commerce and E-Services, 2008.

[18] OMG, "Service oriented architecture modeling language (SoaML) – specification for the uml profile and metamodel for services (UPMS)", Version 1.0 Beta 1, 2009.

[19] P. Kroll and P. Kruchten, The Rational Unified Process Made Easy, a Practitioner's Guide to the RUP, Addison-Wesley, 2003.

[20] M. Gebhart and S. Abeck, "Rule-based service modeling", The Fourth International Conference on Software Engineering Advances (ICSEA 2009), Porto, Portugal, September 2009, pp. 271-276.

[21] P. Hoyer, M. Gebhart, I. Pansa, S. Link, A. Dikanski, and S. Abeck, "A model-driven development approach for service-oriented integration scenarios", 2009.

[22] T. Erl, SOA – Design Patterns, Prentice Hall, 2008. ISBN 978-0-13-613516-6.

[23] T. Erl, Web Service Contract Design & Versioning for SOA, Prentice Hall, 2008. ISBN 978-0-13-613517-3.

[24] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA – Service-Oriented Architecture Best Practices, 2005. ISBN 0-13-146575-9.

[25] S. Johnston, "UML 2.0 profile for software services", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005. [accessed: January 04, 2011]

[26] J. Amsden, "Modeling with soaml, the service-oriented architecture modeling language – part 1 – service identification", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html, 2010. [accessed: January 04, 2011]

[27] S. Johnston, "Rational uml profile for business modeling", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/5167.html, 2004. [accessed: January 04, 2011]

[28] J. Heumann, "Introduction to business modeling using the unified modeling language (UML)", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/360.html, 2003. [accessed: January 04, 2011]

[29] OMG, "Business process model and notation (BPMN)", Version 2.0 Beta 1, 2009.

[30] W3C, "OWL 2 web ontology language (OWL)", W3C Recommendation, 2009.

[31] European Commission, "The bologna process - towards the european higher education area", http://ec.europa.eu/education/higher-education/doc1290_en.htm, 2010. [accessed: January 04, 2011]

[32] M. Gebhart, J. Moßgraber, T. Usländer, and S. Abeck, „SoaML-basierter entwurf eines dienstorientierten beobachtungssystems", GI Informatik 2010, Leipzig, Germany, October 2010, pp. 360-367.

[33] A. Bauer, S. Eckel, T. Emter, A. Laubenheimer, E. Monari, J. Moßgraber, and F. Reinert, "N.E.S.T. – network enabled surveillance and tracking", Future Security 3rd Security Research Conference Karlsruhe, 2008.

[34] J. Moßgraber, F. Reinert, and H. Vagts, "An architecture for a task-oriented surveillance system", 2009.

[35] The PESCaDO Consortium, "Service-based infrastructure for user-oriented environmental information delivery", EnviroInfo, 2010.

[36] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, "D8.3 Specification of the pescado architecture", Version 1.0, 2010.

[37] M. Horridge, "A practical guide to building owl ontologies using protégé 4 and co-ode tools", http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/, Version 1.2, 2009. [accessed: January 04, 2011]

[38] OMG, "Unified modeling language (UML), superstructure", Version 2.2, 2009.

[39] OMG, "Object constraint language (OCL)", Version 2.2, 2010.

[40] Open SOA (OSOA), "Service component architecture (SCA), sca assembly model V1.00", http://osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf, 2009. [accessed: January 04, 2011]

[41] OASIS, "Web services business process execution language (WSBPEL)", Version 2.0, 2007.

[42] OMG, "XML metadata interchange (XMI) specification", Version 2.0, 2003.

[43] OGC, "Keyhole markup language (KML)", http://www.opengeospatial.org/standards/kml/, Version 2.2, 2008. [accessed: January 04, 2011]

CRUD-DOM:

A Model for Bridging the Gap Between the Object-Oriented and the Relational Paradigms - an Enhanced Performance Assessment Based on a Case Study

Oscar M. Pereira[1], Rui L. Aguiar[2]
Instituto de Telecomunicações
University of Aveiro
Aveiro, Portugal
{omp[1],ruilaa[2]}@ua.pt

Maribel Yasmina Santos
Algoritmi Research Center
University of Minho
Guimarães, Portugal
maribel@dsi.uminho.pt

***Abstract*—the development of database applications comprises three different tiers: application tier, database tier and finally the middle tier also known as the data access layer. The development of each tier *per-se* entails many challenges. Very often the most difficult challenges to be addressed derive from non-functional requirements, as productivity, usability, performance, reliability, high-availability and transparency. This paper is focused on defining and presenting a model for the data access layer aimed to integrate object-oriented application tiers and relational database tiers. The model addresses situations on which users need to explicitly write down complex static Create, Read, Update and Delete (CRUD) expressions and simultaneously get advantages regarding some non-functional requirements. The model, known as CRUD Data Object Model (CRUD-DOM), tackles the following non-functional requirements: performance, usability and productivity. The main contributions of this paper are threefold: 1) to present the CRUD-DOM model; 2) to carry out an enhanced performance assessment based on a case study; 3) to present a tool, called CRUD Manager (CRUD-M), which provides automatic code generation with complementary support for software test and maintenance. The main outcome of this paper is the evidence that the pair CRUD-DOM and CRUD-M effectively addresses productivity, performance and usability requirements in the aforementioned context.**

Keywords - CRUDDO; CRUD-DOM; database; impedance mismatch;, high-performance computing; measurement; middle tier.

I. INTRODUCTION

In order to bridge the gap between the object-oriented and the relational paradigms, a model, known as CRUD-DOM, has been already presented [1]. This paper is an extended version of [1].

In spite of their individual successes, object-oriented and relational paradigms are simply too different to bridge seamlessly, leading to difficulties informally known as *impedance mismatch* [2]. The diverse foundations of the object-oriented and the relational paradigms are a major hindrance for their integration, being an open challenge for more than 45 years [3]. The challenge derives from the multiplicity of aspects that need to be bridged across both paradigms: imperative languages versus declarative languages; compilation and execution performance versus search performance; classes, algorithms and data structures versus relations and indexes; transactions versus *threads*; null pointers versus null for the absence of value [3], and finally, inheritance versus specialization. The impedance mismatch thus presents several challenges for developers of database applications, where often both paradigms are found. These challenges are especially noticeable in environments where production code is under strict development deadlines, and where (timely) code development efficiency is a major concern. In order to cope with the impedance mismatch issue, several solutions have emerged, such as language extensions (SQLJ [4], LINQ [5]), call level interfaces [6] (JDBC [7], ODBC [8] ADO.NET [9]), object/relational mappings (O/RM) (Hibernate [10], TopLink [11], LINQ [5]) and persistence frameworks (JDO [12], JPA [13]). Language extensions may provide static syntax and type checking but always rely on proprietary standards. Call level interfaces, despite their performance, provide no static syntax or static checking. O/RM have the advantage of treating data as objects but do not take the advantage of the database engine performance and further rely on proprietary standards. Persistent frameworks have the same drawbacks as O/RM. Despite their individual advantages, these solutions have not been developed to effectively address situations where users need to write complex static CRUD (Create, Read, Update, Delete) expressions. Table I presents an example of a not very simple CRUD expression that is not easily supported by any current solution. The increasing of the query complexity increases the weaknesses of current solutions.

TABLE I. A CRUD EXPRESSION

```
Select pt.pt_id, pt.pt_fName, pt.pt_lName
    From pt_pilot pt,cc_circuit cc,cf_classif  cf
    Where pt.pt_id=cf.cfPt_id  and  cf.cf_date=cc.cc_date and
          cf.cf_position not between 1 and 3
    Group by pt.pt_id, pt.pt_fName, pt.pt_lName
    Having count(cf.cf_position) = (Select count(*) From Cc_circuit … )
Union
Select top 5 distinct( ….
    from …
    ….
Order by …
```

Not easily supported means that current solutions push users to deal with additional issues, as a decay of

performance, the usage of proprietary language extensions, the usage of proprietary mapping techniques, the absence of support to edit and dynamically test CRUD expressions and some SQL features not easily supported or even not supported at all.

A. Performance, Usability and Productivity

CRUD-DOM addresses mainly three non-functional requirements: performance, usability and productivity. In this section we introduce their basic concepts and also justify and emphasize the relevance of each one.

Performance

Performance is a non-functional software requirement that, among other issues, evaluates how well a system or a component copes with a set of requirements namely for timeless [14]. In this context, two dimensions may be considered: responsiveness and scalability. Responsiveness evaluates conformance with response time requirements evaluating the amount of time to accomplish a task or the number of tasks that can be accomplished in a given period of time. Scalability evaluates the capacity of a system to handle growing demand of power computation while keeping its responsiveness. In this paper we are focused on responsiveness. Scalability will be addressed in future works.

Performance is a pervasive outcome of software systems [15]. Everything affects it, as software design, programming paradigms, programming languages, compilers, operating systems, communication networks, hardware and third party software. As a pervasive quality, performance opens many opportunities to research contributions. Very often, it is one of the most challenging non-functional software requirements in database applications. System architects and system designers are called to decide upon many and difficult options. Each option has an impact on the overall performance. As an example, the middle tier may be built around distinct technologies and solutions, as previously mentioned, being CLI one of them. Despite CLI drawbacks, they cannot be discarded as an important and valid option whenever performance and SQL expressiveness are considered key issues [3]. CLI provide mechanisms to encode Create, Read, Update and Delete (CRUD) expressions inside strings, easily incorporating the power and the expressiveness of SQL. Thus, power and expressiveness are crucial advantages of CLI but this comes with unavoidable and important drawbacks (see detailed discussion in section III).

Usability

Usability is another non-functional software requirement in Software Engineering. It is linked to the software quality design [16]. Several definitions may be found for the usability concept [17-22]. We may accept the definition of Jacob Nielson [17], which is focused on concepts as *Learnability*, *Efficiency*, *Memorability*, *Errorless-pronability* and *Satisfaction*. The application of usability in this work is twofold. The first one is related to the development process of the data access layer. It is addressed by the CRUD-M, which must provide a GUI with improved usability. The second one is related to the usage of the data access layer. The access data layer should provide an improved usability to developers of the application tier.

Productivity

In this work, productivity comprises the factors that may influence costs during the three phases of the software application life-cycle: development, test and maintenance of access layers.

The development phase usually unlocks financial and material resources, and also motivates the involved human resources. On the contrary, the test and maintenance phases are usually neglected and therefore we will pay some additional attention to them.

The costs associated with software testing are very high and may exceed 30% of the total cost of a project [23]. Two of the most relevant sources for such a high cost comprise the attitude assumed by the development team [24] and also by the absence of an adequate infrastructure dedicated to software testing. In the U.S. in 2002 it was estimated a cost between $22 and $59.1 billion [25]. In opposite to what is commonly accepted, rather than an act of testing, the software testing should be seen as an overall strategy to be included in the entire life-cycle of a software system: "*the act of designing tests is one of the best bug preventers known*", Beizer in [26].

Software maintenance is well known for its very high costs and delays in its implementation. Despite being the aspect that consumes more resources during the product life-cycle [27], it has usually been neglected. Software maintenance is an inevitable activity resulting from requests for assistance derived from its usage and from its aging. Software maintenance is associated with different sources but it is generally classified into 4 categories, each one with different weights [28]: adaptive - 25% (changes in the environment where software works); perfective – 50% (adaption to new requirements); corrective – 21% (error correction); preventive – 4% (prevention of future errors). These values, although presented in 1980, still continue to be accepted and cited in several publications [27-29]. Some sources of software maintenance may not be easily controlled by the development team, as are the adaptive and perfective sources. But the other two, mainly the corrective one, have their basis and origin in flaws occurred during the development and test of the product.

B. Motivation

The motivation for this work is anchored in the fact that none of the available current solutions and technologies address effectively and simultaneously all the following features:

- Hand-written CRUD expressions - business logic in database applications very often rely on SQL statements that have to be hand-written. This may be derived from the fact that CRUD expressions are too complex and/or CRUD expressions cannot be inferred from any other data model.

- Decoupled data access layer - in order to completely decouple the three tiers of database applications, the data access layer should be developed as a separate component. This will ensure not only the decouple at its usage level (application level) but also at its development level (organizational/responsibility level).
- Technological independency - technological independency assures that: solutions may run on any environment; users are not compelled to learn new technologies.
- Productivity - productivity should be maximized by exempting users from writing any source code; source code should be automatically generated and tested. Maintenance activities should require minimum user effort.
- Usability: usability should always be presented as a key concern in all aspects: development of the data access layer and also the usage of the data access layer.
- Performance: the performance of the data access layer must always be the main concern.

This work aims to provide a solution that copes with the aforementioned features. For such, we developed a model, known as CRUD Data Object Model (CRUD-DOM) where each CRUD expression is wrapped into a type-safe and type-state object-oriented component, known as CRUD Data Object (CRUD-DO). Furthermore, we developed a tool addressing automatic CRUD-DO generation having as only input the standard SQL statements written by users. This tool is known as CRUD Manager (CRUD-M).

Throughout this paper, by default - unless explicitly referred, all examples are based on Java, SQL Server 2008 and JDBC (CLI) for SQL Server (sqljdbc4). Code snippets may not execute properly since we will only show the relevant code for the points under discussion. For conciseness, Figure 1 presents a partial view of a database schema, which will be used throughout the examples of this paper.

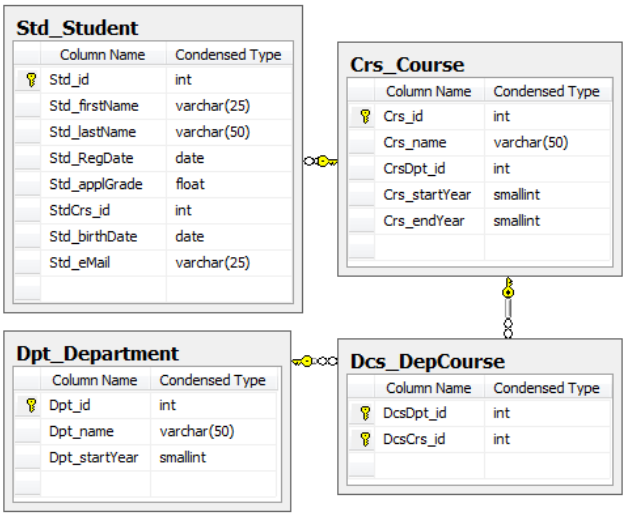


Figure 1. Partial view of the database schema

This paper is organized as follows. Section II presents related work. Section III highlights the impedance mismatch problem. Section IV describes our proposed model (CRUD-DOM), while section V presents the automatic code generation tool (CRUD-M). Section VI presents performance assessment and finally, Section VII presents the final conclusion.

II. Related Work

This section presents the different approaches for the integration of object-oriented and relational paradigms. As a well-known problem in industry, multiple techniques and solutions have been released to address the impedance mismatch problem. For some solutions we will present a real case but always dealing with a very simple query.

Embedded SQL [30] is a method for writing SQL statements in-line with regular source code of the host language inside source files. The SQL statements provide the database interface while the host language provides the remaining support needed for the application to execute. The files are then pre-processed (pre-compiled) in order to check the correctness of the SQL statements namely against the database schema, host language data type and SQL data type checking, and finally syntax checking of the SQL constructions. SQLJ [4] is an example of an Embedded SQL standard, which provides language extensions for embedding SQL statements in regular Java source files. Some SQLJ disadvantages, which are common to most Embedded SQL technologies: 1) SQLJ relies on an extra standard; 2) SQLJ does not decouple SQL statements from regular source code; 3) SQLJ is not suited for client-server environments; 4) SQLJ does not provide a clean object-oriented interface to the assisted application; 5) SQLJ does not provide assistance regarding the maintenance of SQL statements; 6) SQLJ requires a JVM (Java Virtual Machine) built in the database; 7) In practice, embedded SQL has never been widely adopted by end users. Table II shows an example using SQLJ. Examples of other languages that support embedded SQL are: C, C++, COBOL and Fortran.

Table II. SQLJ example

```
// Java
void getStudent( int id ) throws SQLException {
  String firstName = null;
  String lastName = null;
  #sql {
       Select Std_firstName, Std_lastName
          INTO: firstName, lastName
          From Std_Student
          Where Std_id = :id
  }
  System.out.println("Student's name: " + firstName + " " + lastName);
}
```

Despite the aforementioned general disadvantages, some embedded SQL features may be considered as advantages such as: it is based on single development environment with a strong interconnection between the two paradigms; unlike other solutions, embedded SQL does not need to be executed

to check the correctness of the SQL syntax. This task is executed by the pre-compiler.

Object-relational mapping [31, 32] is a programming technique aiming at enforcing relational data models to be closely aligned with the object-oriented paradigm. The relational to object-oriented translation is driven by an explicit mapping (generally in XML) or by schema annotations (inside the source code file). Much of the enforcement is on behalf of getting an object-oriented logic access layer coping with the impedance mismatch [2] issue. Every relational concept must, somehow, have its corresponding concept(s) in the object-oriented paradigm. Very often, mainly in legacy databases, the translation is not straightforward, leading to complex translations, as the case of the relationship and specialization concepts. In these cases, besides the aforementioned hindrance, the relational model lacks essential conceptual information obliging oneself to an extra effort on defining relationship direction, cardinality, etc. Nevertheless, O/RM techniques have been quite successful, either as commercial products (e.g., Oracle TopLink [11], ADO.NET Entity Framework [33], LINQ [5]) or as open source projects (e.g., Hibernate [10]). Albeit this achieved success, well known O/RM drawbacks are unavoidable: 1) each O/RM programming technique relies on proprietary standards introducing new mapping schemas and new SQL-equivalent manipulation languages; 2) O/RM entails an additional effort to map the relational model into the object-oriented model; 3) performance and expressiveness are the two main O/RM penalties; 4) complex CRUD expressions may be supported but they must be hand written and users have no support for their editing and testing. Table III shows a Hibernate example with HQL and Table IV shows an example with Hibernate without HQL (both in Java). Table V shows an example with LINQ in C#. For conciseness, the mapping schema and mapping classes are not explicitly presented.

TABLE III. HIBERNATE EXAMPLE WITH HQL

```
// Java
void getStudent(int id) {
  Session s=HibernateUtil.getSessionFactory().getCurrentSession();
  List list=s.createQuery("from Std_Student").
              setInteger("Std_id", id);
  Student std=(Student) list.get(0);
  firstName=std.Std_firstName();
  lastName=std.Std_lastName();
  System.out.println("Student's name: " + firstName + " " + lastName);
}
```

TABLE IV. HIBERNATE EXAMPLE WITHOUT HQL

```
// Java
void getStudent(int id) {
  Session s=HibernateUtil.getSessionFactory().getCurrentSession();
  Student std=(Student) s.load(Student.class,id);
  firstName=std.Std_firstName();
  lastName=std.Std_lastName();
  System.out.println("Student's name: " + firstName + " " + lastName);
}
```

TABLE V. LINQ EXAMPLE

```
// C#
void getStudent(int id) {
  Student std=from s in db.StdStudent where Std_id=id select s;
  firstName=std.Std_firstname;
  lastName=std.Std_lastname;
  Console.WriteLine("Student's name: " + firstName + " " + lastName);
}
```

Despite the aforementioned disadvantages, O/RM techniques are very powerful whenever the middle tier implementation relies on a direct object-oriented perspective of the relational model. In this particular context O/RM tools relieve programmers from most of the translation work between the two paradigms. CRUD-DOM is not tailored to tackle these situations. Its target is focused on middle tiers based on more complex CRUD expressions. Anyway, CRUD-M may be extended in other to provide an additional feature to automatically create the source code to execute the 4 basic SQL statements in each table: Select one row (by primary key), Insert one row, Update one row (by primary key) and Delete one row (by primary key).

Safe Query Objects [34] combine object-relational mapping with object-oriented languages to specify queries using strongly-typed objects and methods. They rely on Java Data Objects to provide strongly-typed objects and also to provide data persistence. Safe Query Objects are a promising technique to express queries but share most of the aforementioned drawbacks of O/RM, namely regarding performance and SQL expressiveness.

SQL DOM [35] generates a Dynamic Link Library containing classes that are strongly-typed to a database schema. These classes are used to construct dynamic SQL statements without manipulating any strings. As Safe Query Objects, SQL DOM does not take the full advantage of SQL expressiveness and also exhibits very poor results regarding performance.

Static Checking of Dynamically Generated Queries [36] presents a solution based on static string analysis of Java programs to find out where SQL statements are being constructed. The main idea is to find out all possible combinations of distinct SQL statements and then analyze them regarding their syntax and their type mismatch errors. This approach does not affect system performance but exhibits some drawbacks as: 1) all source code is hand written from string concatenation till JDBC execution context; 2) it does not provide any object-oriented view of the SQL statement execution context.

ADO.NET [9, 37] is part of the base class library included in the Microsoft .Net Framework. It is a set of classes that expose data access services to .NET programmers. The DataSet is the key component implementing a disconnected memory-resident representation of the data source. Some of the most important features are: it is aimed at integrating several and distinct data sources (XML, relational, etc.); it supports

several related tables, constraints and relationships between them. The representation of the data source may be as complex as necessary. Therefore, ADO.NET is tailored to meet distinct requirements from those here announced. Table VI depicts the code for an ADO.NET example written in C#.

TABLE VI. ADO.NET EXAMPLE

```
// C#
void getStudent(int id) {
  string sql=”select * from Std_student where Sd_id=” + id;
  SqlDataAdapter da=new SqlDataAdapter();
  da.SelectCommand=new SqlCommand(sql,conn);
  DataTable dt=DataTable();
  da.Fill(dt);
  DataRow dr=dt.Rows[0];
  string firstName=dr[“Std_firstName”];
  string lastName=dr[“Std_lastName”];
  Console.WriteLine(“Student’s name: “ + firstName + “ “ + lastName);
}
```

Call Level Interfaces (CLI) [6], as JDBC [7] and ODBC [8] are practically an unavoidable option whenever performance and SQL expressiveness are simultaneously considered key issues. CLI provide mechanisms to encode Create, Read, Update and Delete SQL expressions inside strings, easily incorporating the performance and expressiveness of SQL. Thus, performance and expressiveness are crucial advantages of CLI but this comes with unavoidable and important drawbacks, namely there is no easy way to link CRUD expressions and the applications they assist; the act of edit CRUD expressions is tricky and error-prone; CRUD expressions are awkward regarding their maintenance and CRUD expressions are vulnerable to SQL injection attacks. In order to overcome the drawbacks of these techniques, we aim to explore CLI, namely through JDBC. These drawbacks and other issues will be thoroughly addressed in the next section. Table VII shows an example using JDBC.

TABLE VII. JDBC EXAMPLE

```
// Java
void getStudent(Connection,conn, int id) throws SQLException {
  sql=”select * “ +
        “from Std_student “ +
        “where std_student=” + id + “);”;
  st=conn.createStatement();
  rs=st.executeQuery(sql);
  firstName=rs.getString(“Std_firstName”);
  lastName=rs.getString(“Std_lastName”);
  System.out.println(“Student’s name: “ + firstName + “ “ + lastName);
}
```

III. IMPEDANCE MISMATCH: COMMON JDBC DRAWBACKS

JDBC is a common tool for integrating relational databases with Java programming language. JDBC is also a representative of the typical challenges of CLI. As such, we will explore JDBC as a target tool. Thus, this section aims to emphasize common drawbacks regarding the utilization of JDBC focusing mainly on the *ResultSet* interface. The drawbacks may be split into four categories: 1) the process for editing SQL statements; 2) the process for retrieving data from returned relations; 3) the process of updating databases through *CONCUR_UPDATABLE ResultSets*; 4) protocols of *ResultSet* interface regarding its usability. Figure 2 presents a simple example, which comprises some of the drawbacks related to categories 1), 2) and 3). This example is used in the following paragraphs to describe JDBC drawbacks:

a) There is no easy way to link CRUD expressions and their results to the application they assist. CLI provide services to ease the integration of object-oriented applications and relational databases but relevant issues are not overcome as string concatenation (Figure 2: lines 22-24) and conversion between relational and object-oriented paradigms (Figure 2: lines 27, 28, 30).

```
20  void getCourses_( int dptId, int startYear )
21                                  throws Exception {
22    sql="select * from Crs_Course"+
23            "where CrsDpt_id="+dptId+"and "+
24            "Crs_startYear="+startYear+";";
25    rs=st.executeQuery(sql);
26    while ( rs.next() ) {
27      crsId = rs.getInt("Crs_id");
28      crsName = rs.getString("Csr_name");
29      // other attributes
30      rs.updateString( "Crs_name", "John Nace");
31      // other updates
32      rs.updateRow();
33    }
34  }
```

Figure 2. Some JDBC drawbacks

b) Editing CRUD expressions and access to their results is tricky and error-prone. CRUD expressions are constructed by concatenating strings and access to their results is achieved by reading attribute by attribute in a row by row basis. Some of the most usual errors are: a) concatenation errors - missing space between lines (Figure 2, lines 22, 23) and missing space before “and” (Figure 2: line 23); b) type mismatch error - argument *startYear* and column *Crs_startYear* (Figure 2: lines 20, 24); c) retrieving data - misspelled column name (Figure 2: line 28);

c) Errors cannot be checked for correctness at compile time, addressed in [36]. None of the previous errors can be caught at compile time demanding great accuracy while editing the source code in order to prevent additional time on testing, debugging and future maintenance.

d) CRUD expressions are awkward regarding their maintenance, addressed in [38]. CRUD expressions (construction and execution) comprise many different entities grouped in three classes: SQL syntax, CLI services and database schema. While SQL syntax and CLI services can be considered stable, database schema is a dynamic entity. Database schema may change for many reasons, as initial error on conceptual model or the emerging of new requirements, which usually happens several times during the development process and even also after application

deployment. Any simple change in the database schema may involve a huge work on updating the strings that encode the affected SQL statements.

e) CRUD expressions are vulnerable to SQL injection attacks, addressed in [39]. This issue is not addressed in the current version of CRUD-DOM.

f) *ResultSet* usability, *ResultSet* interface has dozens of states, dealing with different combinations of *ResultSet* instantiations, directions, accesses, updates, etc. The developer is before a huge task to become aware of how to use the *ResultSet* interface. *ResultSet* interface comprises several distinct protocols not organized in interfaces, conveying the idea that everything is possible in anytime. ResultSet interface is composed by more than 200 methods and 10 attributes. Figure 3 presents a partial view of the *ResultSet* interface. Each *ResultSet* state has its own usage protocol gathering a subgroup of all methods of the ResultSet interface. Figure 4 depicts the most relevant protocols for this work: Read, Update, Insert and Delete actions. While Read and Delete protocols do not comprise a start and an end instruction, Update and Insert protocols always have a start instruction (implicitly for Update and explicitly for Insert) and an end instruction. Besides the starting and the ending instructions, the main issue for Update and Insert protocols is that the cursor cannot be moved from the current selected row while the protocol is being executed. If the cursor is moved from the selected row while the protocol is being executed, the protocol will be aborted and previous changes are discarded from the in-memory of the ResultSet. In order to overcome some of these difficulties we will present an approach where each protocol is executed through a dedicated interface improving this way ResultSet usability.

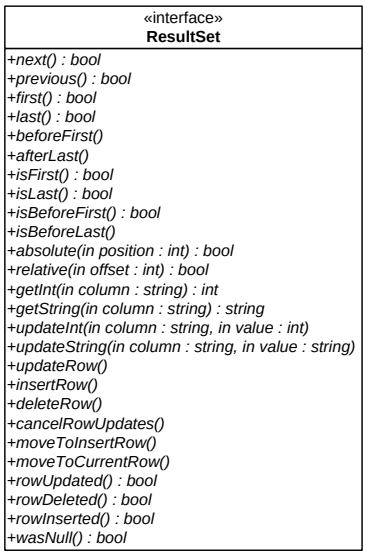


Figure 3. Partial view of the *ResultSet* interface

Some of the aforementioned drawbacks have already been individually addressed as previously cited. In this paper we will present a simple, integrated and unified alternative to overcome all the aforementioned drawbacks, except for the SQL injection attack. The alternative comprises both the CRUD-DOM and the CRUD-M.

```
59      void read(ResultSet rs,int row) throws SQLException {
60          rs.absolute(row);
61          // protocol starts and end with each reading
62          fName=rs.getString("Std_firstName");
63          // ..
64          crsId=rs.getInt("StdCrs_id");
65          // ..
66      }
67      void update(ResultSet rs,int row) throws SQLException {
68          rs.absolute(row);
69          // protocol starts on first update
70          rs.updateString("Std_firstName","Steve");
71          // ..
72          // protocol ends with updateRow
73          rs.updateRow();
74      }
75      void insert(ResultSet rs) throws SQLException {
76          // protocol starts with  moveToInsertRow
77          rs.moveToInsertRow();
78          rs.updateString("Std_firstName","John");
79          // .. more attributes
80          // protocol ends with insertRow
81          rs.insertRow();
82      }
83      void delete(ResultSet rs,int row) throws SQLException {
84          rs.absolute(row);
85          // protocol starts and ends with the deleteRow
86          rs.deleteRow();
87      }
```

Figure 4. Read, Update, Insert and Delete protocols

IV. CRUD-DOM

CRUD-DOM is our abstract model aimed at bridging the gap between relational databases and object-oriented applications. The CRUD-DOM goals are manifold, which were described in section I.B Motivation. Before we delve into the CRUD-DOM issue, we will present a concise overview of Statement/ResultSet interfaces and CRUD expressions.

A. Statement and ResultSet

The Statement interface [40] is used to execute SQL statements and to return the possible results they produce (only for Select statements). The returned results are managed by the ResultSet interface [41]. Loosely speaking, ResultSet interface provides two orthogonal functionalities: *scrollability* and *updatability*. Scrollability defines the ability to scroll over the rows retrieved from the database. There are two options: *forward only* – in this case cursors may only move forward one row at a time; *scrollable* – cursors may move in any direction and jump several rows at a time. Updatability defines the capacity to change the in-memory data managed by the ResultSet interface and therefore the content of the host database. There are two main possibilities: *read only* – the content of the ResultSet is read only and, therefore, no changes are allowed; *updatable* – changes may be performed over the in-memory data, as Insert, Update and Delete. These functionalities are defined at instantiation time of the parent Statement or PreparedStatement [42] object. The combination of these

two functionalities influences the performance of many actions that are executed. This analysis will be carried out in section VI.

B. CRUD Expressions

CRUD expressions are the basic entities from which CRUD-DOM specification must evolve. Therefore, before proceeding with the CRUD-DOM specification, it is advisable to briefly survey CRUD expressions in order to be aware of the JDBC context in which they are used. CRUD expressions comprise the four basic SQL statements for accessing information in databases: *Select*, *Insert*, *Update* and *Delete*. While *Insert*, *Update* and *Delete* statements are used to alter the state of databases, *Select* statements allow the implementation of several views of the database. Hence, CRUD expressions may be grouped into two categories: "query CRUD expressions" (Q-CRUD) whenever involving a *Select* statement; and "execute CRUD expressions" (E-CRUD) whenever involving an *Insert*, *Update* or *Delete* statement. The corresponding CRUD-DOs share some source code but relevant differences must be emphasized. The most relevant difference is that Q-CRUD expressions return one or more relations from the database therefore requiring specific processing, as seen in Figure 2 (lines 26-28). Additionally, in some circumstances and also for certain Q-CRUD expressions it is possible to instantiate updatable *ResultSet*s. Updatable *ResultSet*s provide embedded protocols to update, to delete and to insert data in databases. Figure 2 (lines 30-32) concisely presents a case for the update situation. Other examples are presented in Figure 4 for Update, Insert and Delete actions. Thus, two types of CRUD expressions may be defined. Q-CRUD expressions executed on updatable *ResultSet* are named as Active Q-CRUD expressions (AQ-CRUD). Q-CRUD expressions executed on non-updatable *ResultSet* are named as Passive Q-CRUD expressions (PQ-CRUD).

C. CRUD-DOM Objectives

As previously mentioned CRUD-DOM addresses three main objectives: high performance, high usability and high productivity. In this section we will describe the most relevant features to be included and that are dependent on CRUD-DOM architecture: performance and usability (access layer usability). The remaining features as usability (CRUD Manager usability) and productivity depend on CRUD Manager.

Performance

To comply with the performance objective, the following features were established:

- Pool of CRUD-DOs: CRUD-DOs rely on statically crated CRUD expressions. CRUD-DOs exist inside the access layer and are supposed to be reused over and over again. Therefore, a pool of CRUD-DOs should minimize CRUD-DO instantiation time.
- Prepared statements: for the reasons pointed in the previous feature (reuse of CRUD-DOs), it is advisable to use prepared statements (PreparedStatement [42]) instead of Statements (Statement [40]).

Usability

To comply with the usability, the following features were established:

- Type-state [43] oriented interfaces: For each main ResultSet protocol (Read, Update, Insert and Delete) CRUD-DOM makes available a type-state oriented interface.
- Semantic interfaces: all interfaces defined by CRUD-DOM aimed to deal with query parameters and attributes of the returned relations are always semantically oriented. This means that the names of their methods and their arguments are always derived from the associate queries.
- Factory: from users' perspective, all CRUD-DOs are created and managed through a factory.

D. CRUD-DOM Details

We will present CRUD-DOM architecture by enumerating and describing the fundamental features for each type of CRUD expression: E-CRUD, PQ-CRUD and AQ-CRUD. Afterwards, we will present class diagrams for each type of CRUD expression. For all presented examples we assume that:

- "CruddoName" is the name for all types of CRUD expressions used as examples.
- Q-CRUD expression is "*select co1A, colB from table where colA>param*" where *colA* is *integer* and *colB* is *String*.
- E-CRUD is any *delete*, *update* or *insert* SQL statement with one parameter (*param*) of type integer.

All CRUD-DOs share the following features:

- Every CRUD-DO has a unique name.
- Every CRUD-DO is built around one class, known as the invocation class, and among other things, the class is responsible for the execution of the CRUD expression.
- The name of the invocation class is the same as the one given to the CRUD-DO.
- The invocation class has only one constructor with no arguments. Its visibility is protected.
- The invocation class has one method with the following signature *void config(Connection conn)*. This method is responsible for setting the connection to be used during the query execution.
- The invocation class has one method named *execute*, which is responsible for the execution of the CRUD expression. This method returns no value and has as many arguments as the number of the CRUD expression parameters. The name, type and order of the arguments depend on the name, type and order of CRUD expression parameters. For our example, *execute* has one parameter named as *param* and its type is *int*.

All CRUD-DOs derived from E-CRUD expressions share the following feature:

- The invocation class has a method with the following signature: *int getAffectedRows()*; this method returns the number of rows affected by the execution of the E-CRUD expression.

Figure 5 presents the class diagram for the E-CRUD expression example.

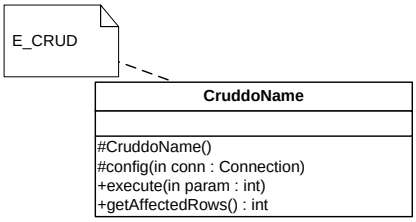


Figure 5. Class diagram for E-CRUD expressions

All CRUD-DOs derived from Q-CRUD expressions share the following feature:

- The invocation class must implement all the scrollable methods in accordance to its instantiation criterion.
- If the ResultSet is scrollable, provide a method with the following signature: *int getRowCount()*; this method returns the number of rows retrieved by the Select statement;
- Q-CRUD expressions have no concrete instances. They are super types for PQ-CRUD and AQ-CRUD expressions.

All CRUD-DOs derived from PQ-CRUD expressions share the following features:

- Extend features of Q-CRUD expressions;
- The invocation class has one method with the following signature: *CruddoName_readTuple beginRead();*
- *CruddoName_readTuple* class, known as the access class, implements one method, generally known as access method, for each attribute of the returned relation. Each access method has the following signature *javaDataType gAttributeName()* where *JavaDataType* is the correspondent java data type for SQL data type and the method's name is built by concatenating the name of the attribute (first letter converted to uppercase) with the prefix *g*.

Figure 6 presents the class diagram for the PQ-CRUD example.

All CRUD-DOs derived from AQ-CRUD expressions share the following features:

- Extend features of Q-CRUD expressions;
- The invocation class may provide any subset of the following four features: *readable*, *updatable*, *insertable* and *deletable*; whenever provided, the *readable* feature may also be included in the remaining features to improve their usability;
- If CRUD-DO is *readable* it implements one method with the following signature: *CruddoName_readTuple beginRead()*;
- If CRUD-DO is *updatable* it implements one method with the following signature: *CruddoName_updateTuple beginUpdate( )*;
- If CRUD-DO is *insertable* it implements one method with the following signature: *CruddoName_insertTuple beginInsert( )*;
- If CRUD-DO is *deletable* it implements one method with the following signature: *void delete()*;
- *CruddoName_readTuple* class: previously explained for PQ-CRUD;
- *CruddoName_updateTuple* and *CruddoName_insertTuple* classes provide functionalities easily perceived from *CruddoName_readTuple* class: access methods have *s* as prefix instead of *g*;.
- The *delete* method, deletes the current row from the *ResultSet*.

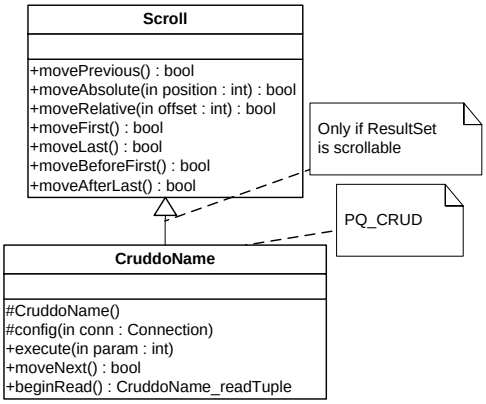


Figure 6. Class diagram for PQ-CRUD expressions

Figure 7, Figure 8, Figure 9 and Figure 10 present the class diagrams for AQ-CRUD expressions.

Class diagrams have been presented for each type of CRUD expression. To completely understand the class diagrams it is necessary to have an understanding of how the *ResultSet* interface is implemented. Original *ResultSet* method names have been renamed and some new ones have been included. Renamed methods are easily identified: *next->moveNext*, *previous->movePrevious*, etc. Only a subgroup of all methods has been presented in order to avoid overcrowding the class diagrams.

There is a factory responsible for creating the correct instances and also for managing the pool of CRUD-DO

instances. Figure 11 depicts the factory source code for managing CruddoName. After their utilization, CRUD-DOs may be released for future reutilization (Figure 11, line 25) maintaining this way an active pool of CRUD-DOs. Before creating a new instance, the factory checks if there is any instance available inside the pool (Figure 11, line 31).

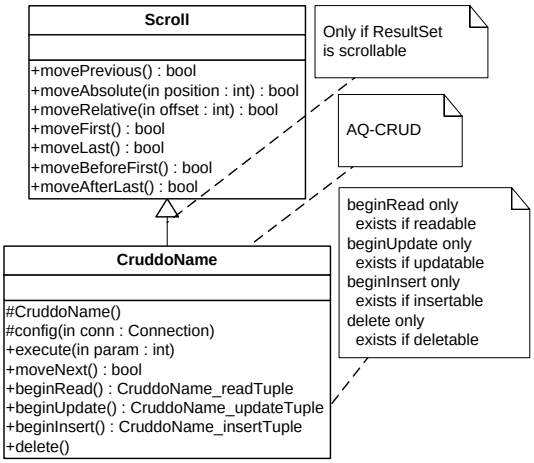


Figure 7. Class diagram for AQ-CRUD expressions

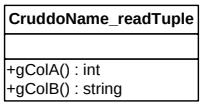


Figure 8. Readable class diagram for Q-CRUD expressions

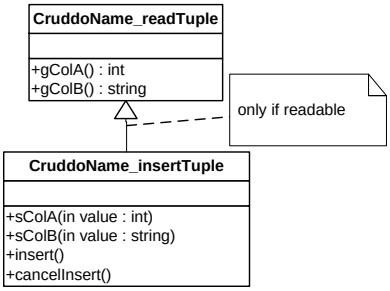


Figure 9. Insertable class diagram for AQ-CRUD expressions

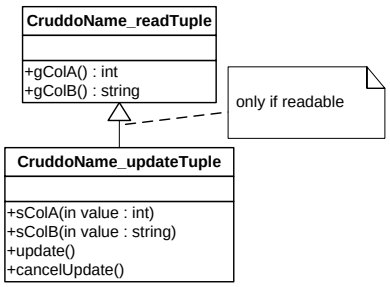


Figure 10. Updatable class diagram for AQ-CRUD expressions

```
21  private static ArrayList<CruddoName> cns=
22                                  new ArrayList<CruddoName>();
23  private static CruddoName cruddoName;
24
25  public static void release(CruddoName cn)
26                                  throws SQLException {
27      cns.add(cn);
28  }
29  public static CruddoName
30              CruddoName(Connection conn) throws Exception {
31      if (cns.size()>0)
32          cruddoName=cns.remove(0);
33      else cruddoName=new CruddoName();
34      cruddoName.config(conn);
35      return cruddoName;
36  }
```

Figure 11. Factory: pool management

V. CRUD MANAGER

CRUD-M addresses productivity objectives (automatic code generation, semi-automatic test and also maintenance) and usability objectives. No special programming skills should be required to use CRUD-M and learning time should be minimal. CRUD-M usage is centered in a GUI component presented in Figure 12. Figure 12 shows a concrete example for an AQ-CRUD expression, called *Courses*, which was created as readable, updatable and insertable but not deletable. Figure 13 shows the usage of CRUD-DO *Courses* from the application tier point of view. As one can see, the integration is seamless regarding impedance mismatch. Additionally, an approach for the implementation of *ResultSet* as a typestate [44] component is provided improving this way CRUD-DO usability. This may be verified, as an example, by the definition of the *Courses_readTuple* interface (Figure 13, lines 68, 69), which provides a coherent protocol for retrieving data from the *ResultSet*.

CRUD-M encompasses five main blocks as depicted in Figure 14. User launches CRUD-M and defines which database is going to be used. Then, "Schema Reader" reads the schema of the database. From now on, users may edit and/or maintain CRUD expressions. "CRUD Editor" provides a context where CRUD expressions may be edited. "CRUD Execution Unit" may help "CRUD Editor" in some specific tasks as defining SQL parameters and executing statements against the database. After executing successfully an SQL statement against the database, users are allowed to create CRUD-DO, which will be accomplished by "CRUD-DO Generator". "CRUD Maintenance" parses CRUD-DO and retrieves the underlying CRUD expression to be reedited by "CRUD Editor". A more detailed description for each bock follows:

Schema Reader: this component reads the schema of the database, which is mainly used to automatically suggest the Java data types for the parameters of CRUD expressions.

CRUD Editor: CRUD Editor is a text editor where CRUD expressions may be written from scratch. Parameters defined in runtime must be identified through a unique name preceded by a '@' character. These names will be used for

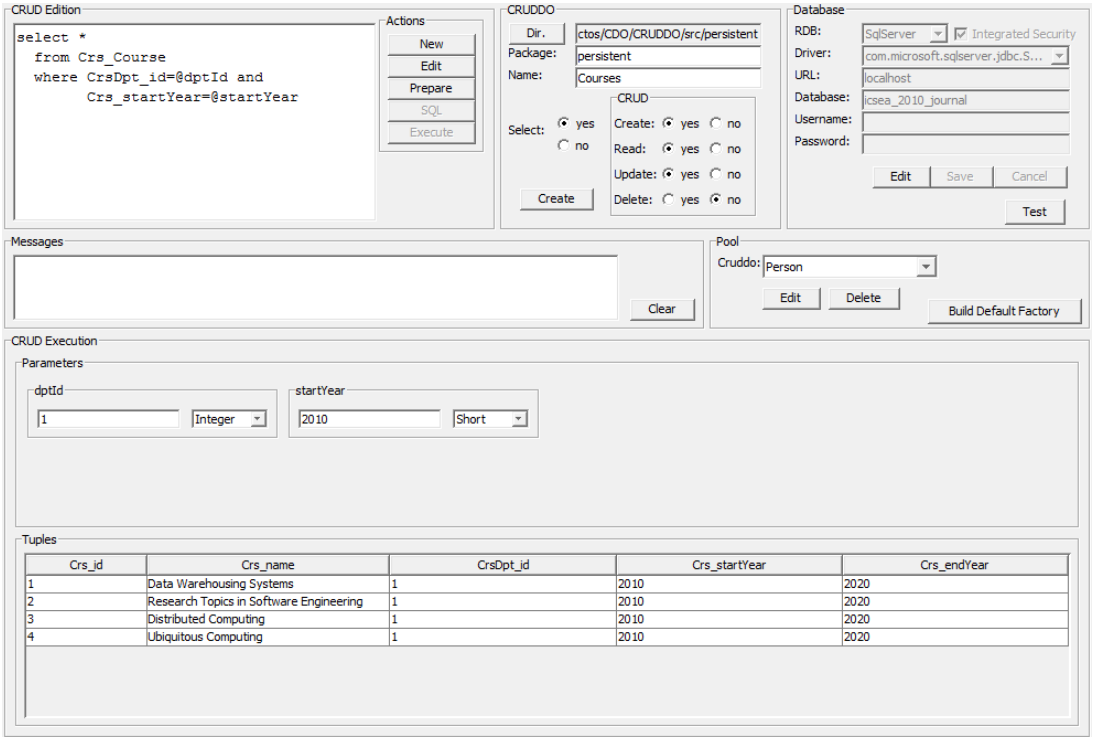


Figure 12. CRUD-M GUI

```
60  public void Courses(int dptId,short startYear)
61                                  throws SQLException {
62      Courses c = Factory.Courses(conn);
63      c.execute(dptId,startYear);
64
65      Courses_readTuple r=c.getRead();
66      Courses_updateTuple u=c.getUpdate();
67      while (c.moveNext()) {
68          crsId=r.gCrs_id();
69          crsName=r.gCrs_name();
70          // .. more attributes
71          u.sCrs_name("Computer Vision");
72          // .. more attributes
73          u.update();
74      }
75  }
```

Figure 13. *Courses* from the application point of view

the arguments of the *execute* method of the invocation classes. In our example we have defined two parameters: *dptId* and *startYear*.

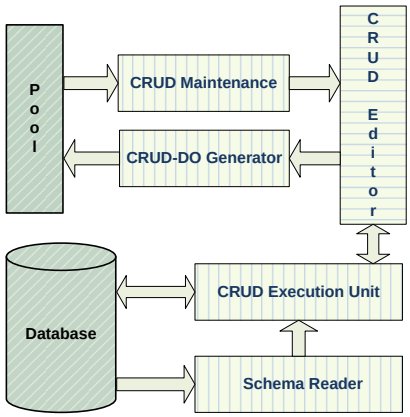


Figure 14. Block diagram of the CRUD-M

CRUD Execution Unit: CRUD Execution Unit is responsible for three tasks: 1) providing, whenever necessary, input data components for SQL parameters. Each input component is identified by the name of the associated parameter and has a default Java Data Type derived from the database schema. Users may select another Java Data Type becoming responsible for their decision; 2) executing the edited CRUD expression against the database proving this way an expedite and integrated tool for evaluating the correctness of CRUD expressions and also for testing the outcome of CRUD expressions. Developers are relieved to write source code to test and debug their CRUD expressions; 3) formatting a table in runtime to present the content of returned relations, whenever the underlying CRUD is a Q-CRUD expression. This visualization allows developers to have an immediate visual feedback about the retuned data and easily evaluate the outcome of Q-CRUD expressions execution. In our example, the returned relation has 4 rows and 5 attributes.

CRUD-DO Generator: CRUD-DO Generator creates automatically all the necessary source code for the underlying CRUD expressions. For all types of CRUD expressions, users must input some additional information, as: CRUD-DO's name, package's name, type of CRUD expression, pool directory for CRUD-DOs, etc. Some additional information is required if the CRUD expression is

of type AQ-CRUD, as which of the following functionalities should be implemented: *readable*, *insertable*, *updatable* and *deletable*.

CRUD Maintenance: this component keeps track of all existing CRUD-DOs in the pool directory. Any CRUD-DO in the pool directory may be selected for editing or to be deleted. If it is selected for editing, the underlying CRUD expression is retrieved from the invocation class and presented in the CRUD editor. From now on, the CRUD expression may be retested or reedited to update the current CRUD-DO or even to create a new one.

VI. PERFORMANCE ASSESSMENT

As mentioned in section I, two dimensions may be considered for performance: responsiveness and scalability. Hereafter, performance should be understood as the responsiveness dimension. The first version of CRUD-DOM was presented in [1]. CRUD-DOM performance was evaluated by measuring the responsiveness for a particular situation: a fixed block of code (one for each protocol) was repeatedly executed for a specific number of times. In this new approach, we will get a more dynamic view about how CRUD-DOM and JDBC behave. This will be achieved by stressing them under several conditions. Details are explained in the next sub-sections.

A. The valuation testbed

All measurements share the same platform: PC - Dell Latitude E5500; CPU - Intel Duo Core P8600 @2.40GHz; RAM - 4.00 GB; OS - Windows Vista Enterprise Service Pack 2 (32bits); Java SE 6; JDBC(sqljdbc4) and SQL Server 2008 version 10.0.1600.22. In order to promote an ideal environment the following actions were taken: the running threads were given the highest priority and all non-essential processes/services were canceled. Transactions were not used and *auto-Commit* was used in all connections.

A new database was created in conformance with the schema presented in Figure 1. In order to avoid some overhead added by SQL Server, some default properties were changed as, Auto Update Statistics = false and Recovery Model=Simple.

The performance assessment addresses two goals: the first one, known as standard JDBC assessment (S-JDBC), is to understand the behavior of the standard Statement and ResultSet interfaces; the second one, based on a component relying on CRUD-DOM (C-CRUD), aims to assess C-CRUD and compare it with S-JDBC. S-JDBC and C-CRUD are from now on generally known as entities and formally represented by the letter *E*.

Part of the results of both assessments is influenced by the Microsoft TDS protocol and also by the implemented mechanisms on both sides (JDBC and SQL Server) to support it. Some key notes are provided to help on the understanding of the collected results:

- selected data through forward-only and read-only ResulSets are always transferred to the client side in a single batch. Sql Server does not implement any mechanism to supervise or control client behavior. On the other side, for other types of ResultSets, Sql Server transfers data in blocks and keeps track of clients' operations. This is achieved by a cursor and a dataset that keeps all the selected data and also keeps track on which row clients are pointing to. This means that it is expected that forward-only and read-only ResultSets should get better performance results than the other types of ResultSets.
- forward-only ResultSets require a simpler mechanism to scroll over the selected data. This means that JDBC and Sql Server have optimized algorithms and therefore improved performance for forward-only ResultSets.
- Read-only ResultSets do not create, explicitly, any concurrency constraint on the database and, therefore, their implementations are more effective on both sides.
- Scrollable and updatable ResultSets are expected to have the worst performance. They are the sum the most complex implementations of TDS: not forward-only and nor read-only.

The size of blocks to be retrieved from the Sql Server may be controlled by setting the block fetch size. Thus, in order to impose a similar environment to all the collected measures, the fetch size has been set to guaranty that all rows are retrieved from Sql Server in a single block.

Sql Server supports more ResultSet types than those defined in the standard JDBC. A more detailed description about Microsoft implementation of JDBC may be found here [45].

In [1], the context in which the assessment took place was characterized by: 1) the type of Statement {Forward-only Read-only (FR), Forward-only Updatable (FU), Scrollable Read-only (SR) and Scrollable Updatable (SU)}; 2) the type of operation {Read (R), Update (U), Insert (I) and Delete (D)} and finally 3) by defining a normalized metric based on the number of cycles that was possible to compute in a second. In spite of its simplicity and validity, we have adopted a new strategy that provides a better evaluation for both entities.

The environment in which the assessment here presented took place is characterized by: CRUD expression, scenarios, contexts, units and data. These items are explained in the next paragraphs.

CRUD expressions: All measurements derive from the AQ-CRUD expression "*Select * from Std_student*".

Scenarios (S): Four scenarios were defined for each operation to be evaluated: Read (S_{re}), Update (S_{up}, S_{cu}), Insert (S_{in}, S_{ci}) and Delete (S_{de}). The S_{re} consists in select a certain number of tuples from the database and then read all attributes of all tuples. The update scenario comprises two variants: a) S_{up} consists in selecting a certain number of

tuples from the database and then update all attributes (except the primary key – Std_id) of all tuples without committing the changes to the database; b) S_{cu} consists in selecting a certain number of tuples from the database and then update all attributes (except the primary key – Std_id) of all tuples and commit the changes to the database. The insert scenario comprises two variants: a) the S_{in} consists in selecting zero rows from the database and then insert all attributes of a certain number of rows into the ResultSet without committing them to the database; b) the S_{ci} consists in selecting zero rows from the database and then insert all attributes of a certain number of rows into the ResultSet committing them to the database. The S_{de} scenario consists in select a certain number of tuples from the database and then to delete all tuples one by one. Table VIII concisely describes all scenarios. These scenarios are only one possibility among an infinity of others. Thus, it was decided to only assess S-JDBC in these scenarios because the most relevant assessment is carried out for the individual units (see Units).

TABLE VIII. FORMAL DESCRIPTION OF ALL SCENARIOS

S	Description
S_{re}	Delete all tuples from the table Std_Student Insert *n* tuples into the table Std_Student Start clock Select all (*n)* tuples from the table Std_Student For each tuple Read all attributes Stop clock
S_{up}	Delete all tuples from the table Std_Student Insert *n* tuples into the table Std_Student Start clock Select all(*n*) tuples from the table Std_Student For each tuple Update all attributes except the pk // without committing them Stop clock
S_{cu}	Delete all tuples from the table Std_Student Insert *n* tuples into the table Std_Student Start clock Select all(*n*) tuples from the table Std_Student For each tuple Update all attributes except the pk Commit changes Stop clock
S_{in}	Delete all tuples from the table Std_Student Start clock Select all (*zero*) tuples from the table Std_Student While insert more tuples Insert all attributes // without committing them Stop clock
S_{ci}	Delete all tuples from the table Std_Student Start clock Select all (*zero*) tuples from the table Std_Student While insert more tuples Insert all attributes Commit new tuple Stop clock
S_{de}	Delete all tuples from the table Std_Student Insert *n* tuples into the table Std_Student Start clock Select *n* tuples from the table Std_Student For each tuple Delete tuple Stop clock

Unit (U): A unit is a task whose execution time is relevant to understand the behavior of any of the four scenarios. The following units were defined: time to execute the select statement (U_{se}), time to read all returned tuples (U_{re}), time to update all returned tuples but not to commit them to the database (U_{up}), time to insert tuples into the ResultSet but not to commit them (U_{in}), time to update tuples and commit them to the database (U_{uc}), time to insert tuples and commit them to the database (U_{ic}) and finally time to delete tuples from the database (U_{de}). Table IX concisely describes all units. Each scenario may be seen as an aggregation of individual units.

Now let's present the composition for each scenario in terms of units: $S_{re}=U_{se}+U_{re}$, $S_{up}=U_{se}+U_{up}$, $S_{cu}=U_{se}+U_{uc}$ $S_{in}=U_{se}+U_{in}$, $S_{ci}=U_{se}+U_{ic}$ and $S_{de}=U_{se}+U_{de}$.

TABLE IX. FORMAL DESCRIPTION OF ALL UNITS

C	Description
U_{se}	Delete all tuples from the table Std_Student Insert *n* into the database table Std_Student Start clock Select all (*n)* tuples from the table Std_Student Stop clock
U_{re}	Delete all tuples from the table Std_Student Insert *n* into the table Std_Student Select all (*n)* tuples from the table Std_Student Start clock For each tuple Read all attributes Stop clock
U_{up}	Delete all tuples from the table Std_Student Insert *n* tuples into the table Std_Student Select all(*n*) tuples from the table Std_Student Start clock For each tuple Update all attributes without commit (except the pk) Stop clock
U_{cu}	Delete all tuples from the table Std_Student Insert *n* tuples into the table Std_Student Select all(*n*) tuples from the table Std_Student Start clock For each tuple Update all attributes (except the pk) Commit changes Stop clock
U_{in}	Delete all tuples from the table Std_Student Select all (*zero*) tuples from the table Std_Student Start clock While insert more tuples Insert all attributes without commit Stop clock
U_{ci}	Delete all tuples from the table Std_Student Select all (*zero*) tuples from the table Std_Student Start clock While insert more tuples Insert all attributes Commit new tuple Stop clock
U_{de}	Delete all tuples from the table Std_Student Insert *n* tuples into the table Std_Student Select all (*n*) tuples from the table Std_Student Start clock For each tuple Delete tuple Stop clock

Context (C): Four contexts were defined for the Statement interface: forward-only and read-only (C_{fr}), forward-only and updatable (C_{fu}), scrollable and read-only (C_{sr}) and finally scrollable and updatable (C_{su}). All contexts were used to explicitly assess S-JDBC. C-CRUD was only assessed in the C_{su}. The justification for this option is that S-JDBC and C-CRUD architectures do not depend on the running context. The collected differences between S-JDBC and C-CRUD in one context should be equivalent in all the other contexts. This means that if for C_{su} the difference between C-CRUD and S-JDBC is Δt then it will remain Δt for the other contexts. Therefore, the behavior of C-CRUD for the remaining contexts may be inferred from the behavior of S-JDBC in those contexts and from the collected differences between S-JDBC and C-CRUD in C_{su}. This assertion has been confirmed by several collected measurements in the other remaining contexts. Table X describes all contexts.

TABLE X. DESCRIPTION OF ALL CONTEXTS

C	Description
C_{fr}	Forward-only and read-only
C_{fu}	Forward-only and updatable
C_{sr}	Scrollable and read-only
C_{su}	Scrollable and updatable

Data: In order to promote a dynamic view about the behavior of each entity, it was decided not to measure the number of cycles that is possible to be computed in a second but to measure the required time to execute each scenario/unit for a set of numbers of rows. The chosen set of numbers of rows is: 5, 10, 15, 25, 50, 75, 100, 150, 250, 350 and 500 rows. This approach gives a dynamic perspective about the behaviors of all entities and is applied to all scenarios, contexts and units. A simple formalization of both entities may be expressed as:

$$S(\alpha,\eta) \quad (1)$$

$$U(\alpha,\eta) \quad (2)$$

Table XI describes each symbol of equations (1) and (2).

TABLE XI. MEANING OF EQUATIONS (1) AND (2)

	Description	Domain
S	Any subset of all scenarios. All scenarios is represented by S_{all}.	$S \in \{S_{all},S_{re},S_{up},S_{cu}, S_{in},S_{ci},S_{de}\}$
U	Any subset of all units (valid for the defined scenario and context). All units is represented by U_{all}.	$U \in \{U_{all},U_{se},U_{sc},U_{re},U_{up},U_{cu},U_{in}, U_{ci},U_{de}\}$
α	Any subset of all contexts. All contexts are represented by c_{all}.	$\alpha \in \{c_{all},c_{fr},c_{fu},c_{sr},c_{su}\}$
η	Any subset of the set of rows. All set is represented by n_{all}.	$\eta \in \{n_{all},n_5,n_{10},n_{15},n_{25},n_{50},n_{75}, n_{100},n_{150},n_{250},n_{350},n_{500}\}$.

Example, $S_{de}(c_{fu,su},n_{all})$ means: scenario delete, contexts forward-only updatable and scrollable updatable and the complete set of rows.

A slot is defined as the minimum granularity for which it is necessary to collected measures. Examples: $S_{re}(c_{fr},n_5)$, $S_{re}(c_{fr},n_{l0})$ and $U_{up}(c_{fu,su},n_{250})$. The distribution and the total number of different slots are presented in Table XII. The number of slots for S-JDBC for all scenarios is computed by multiplying the number of scenarios by the number of contexts by the number of sets of rows. The other values follow the same reasoning to be computed. The total number of slots for both entities is 649.

TABLE XII. NUMBER OF SLOTS

	S-JDBC	C-CRUD	Total
Scenarios	6x4x11=264	0	264
Units	7x4x11=308	7x1x11=77	385
Total	572	77	649

The measures used in all the following graphics for each slot were computed, as:

- At least 500 raw measures were collected.
- The 25 best raw measures were discarded.
- Measure=average of the 50 best remaining raw measures.

Thus, at least 649x500=324,500 raw measures were collected for this current assessment.

B. S- JDBC assessment

S-JDBC assessment comprises both the units and the scenarios. The assessment of units allows us to analyze and isolate the impact of each context by unit. The assessment of scenarios also allows us to analyze the impact by context but the simulation in closer to real situations because the starting point is always triggered by a *select* statement. Just to remind, AQ-CRUD expressions always comprise a Select statement.

S-JDBC assessment is carried out without any special architecture, avoiding this way any additional overhead. This will be confirmed in the following paragraphs.

1) Assessment of units

In section IV.A it was mentioned that each context (combination of functionalities) may influence the performance of each operation. In this section we will analyze the impact of the chosen contexts in each unit.

Figure 15, Figure 18, Figure 21, Figure 23, Figure 25, Figure 27 and Figure 30 depict the source code for each unit. Each unit is individually controlled in order to collect accurate measures for its execution time. These figures show that: the source code is exempt of any architecture and the source code is in line with the general description of all units, see Table IX. These units have some modifications when compared to the equivalent ones presented in [1] derived from the changes introduced in the current test-bed.

Figure 16, Figure 17, Figure 19, Figure 20, Figure 22, Figure 24, Figure 26, Figure 28, Figure 29, Figure 31 and Figure 32 show the performance of all units. The column bars represent the required time to execute the unit (T_{sj}) and the associated vertical axis is the left one. The dashed lines represent the required mean time to process one row (R_{sj}) and the associated vertical axis is the right one. R_{sj} is computed dividing T_{sj} by η.

U_{se} – Select unit

Figure 15 depicts the main source code for the $U_{se}(c_{all},n_{all})$. This unit is focused on measuring the required time to select each set of number of rows.

```
77  // Select unit
78  start=System.nanoTime();
79  rs=st.executeQuery("select * from std_student");
80  Use_time=System.nanoTime()-start;
81
```

Figure 15 . S-JDBC: source code for the $U_{se}(c_{all},n_{all})$.

Figure 16 presents the general behavior of the $U_{se}(c_{all},n_{all})$. The dashed lines are very close conveying the need for a more detailed graphic. Figure 17 presents a more detailed view of Figure 16 emphasizing the behavior of each context.

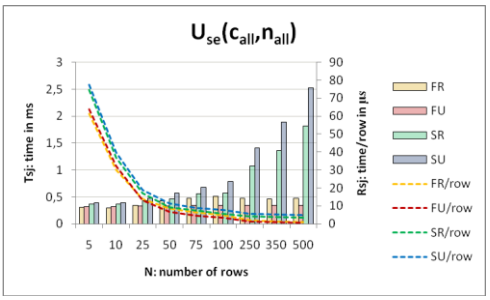


Figure 16. S-JDBC: behavior of $U_{se}(c_{all},n_{all})$

From Figure 16 and Figure 17 we may conclude that:

- T_{sj} of $C_{fr,fu}$ is weakly dependent on η.
- T_{sj} of $C_{sr,su}$ increases with η.
- R_{sj} decreases for all contexts when η increases; as an example, for C_{su}, T_{sj} varies from 77µs till 5.2µs.
- C_{fu}, C_{fr}, C_{sr} and C_{su} are ordered from the best to the worst R_{sj} score.

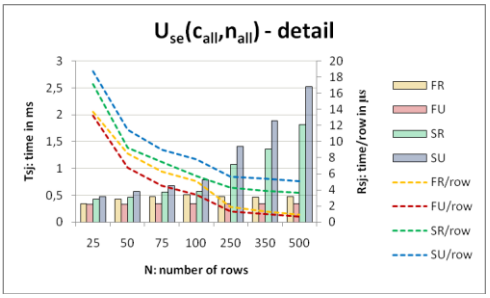


Figure 17. S-JDBC: detail of $U_{se}(c_{all},n_{all})$.

Main point: scrollable ResultSets should be avoided whenever possible, mainly when the number of rows is above 100.

U_{re} – Read unit

Figure 18 depicts the main source code for the $U_{re}(c_{all},n_{all})$. This unit is focused on measuring the required time to read all rows returned by the select statement.

Figure 19 presents the general behavior of the $U_{re}(c_{all},n_{all})$. The dashed lines for $C_{fu,sr,su}$ are very close conveying the need for a more detailed graphic. Figure 20 presents a more detailed view of Figure 19 emphasizing the behavior of the 4 contexts.

```
83  // Read unit
84  start=System.nanoTime();
85  while (rs.next()) {
86      id=rs.getInt( "Std_id" );
87      firstName=rs.getString("Std_firstName");
88      lastName=rs.getString("Std_lastName");
89      rDate=rs.getDate("Std_regDate");
90      applGrade=rs.getFloat("Std_applGrade");
91      crsId=rs.getInt("StdCrs_id");
92      bDate=rs.getDate("Std_birthDate");
93      eMail=rs.getString("Std_eMail");
94  }
95  Ure_time=System.nanoTime()-start;
```

Figure 18. S-JDBC: source code for the $U_{re}(c_{all},n_{all})$.

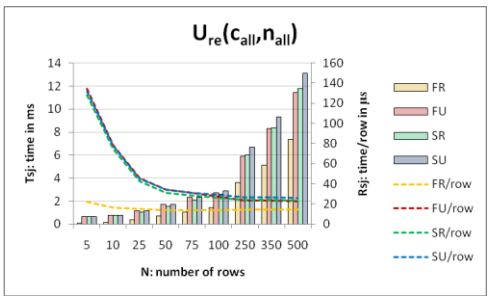


Figure 19. S-JDBC: behavior of $U_{re}(c_{all},n_{all})$.

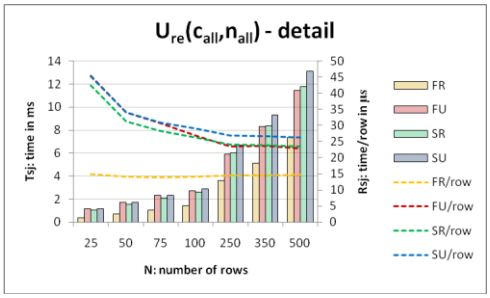


Figure 20. S-JDBC: detail of $U_{re}(c_{all},n_{all})$.

From Figure 19 and Figure 20 we may conclude that:

- T_{sj} of C_{all} increase with η; the C_{fr} is the most independent one.
- R_{sj} decreases for $C_{fu,sr,su}$ when η increases; as an example, for C_{su} the ratio varies from 135µs till

26µs. C_{fr} is independent of η revealing a constant performance.
- C_{fr} has by far the best scores for all η; $C_{fu,sr,su}$ present similar scores for all η.

Main point: C_{fr} is always the best option.

U_{up} – update unit without commit

Figure 21 depicts the main source code for the $U_{up}(c_{fu,su},n_{all})$. This unit is focused on measuring the required time to update all rows returned by the select statement but without committing those changes to the database.

```
97  // Update unit without commit
98  start=System.nanoTime();
99  while (rs.next()) {
100     rs.updateString("Std_firstName","firstName");
101     rs.updateString("Std_lastName","lastName");
102     rs.updateDate("Std_regDate", date);
103     rs.updateFloat("Std_applGrade",15F);
104     rs.updateInt("StdCrs_id",1);
105     rs.updateDate("Std_birthDate",bDate);
106     rs.updateString("Std_eMail","email@ua.pt");
107 }
108 Uup time=System.nanoTime()-start;
```

Figure 21. S-JDBC: source code for the $U_{up}(c_{fu,su},n_{all})$.

Figure 22 presents the general behavior of the $U_{up}(c_{fu,su},n_{all})$. The dashed lines for $C_{fu,su}$ are overlapped showing that the behaviors are the same for both contexts.

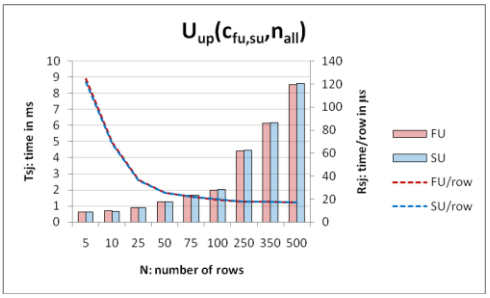


Figure 22. S-JDBC: behavior of $U_{up}(c_{fu,su},n_{all})$.

From Figure 22 we may conclude that:
- T_{sj} of $C_{fu,su}$ increases with η;
- The behavior is the same for both contexts.
- R_{sj} decreases for $C_{fu,su}$ when η increases; it ranges from 127µs till 18µs.
- For η>50, R_{sj} tends to be constant

Main point: there is no difference between C_{fu} and C_{su}.

U_{cu} – update unit with commit

Figure 23 depicts the main source code for the $U_{cu}(c_{fu,su},n_{all})$. This unit is focused on measuring the required time to update all rows returned by the select statement and also for committing those changes to the database.

```
110 // Update unit with commit
111 start=System.nanoTime();
112 while (rs.next()) {
113     rs.updateString("Std_firstName","firstName");
114     rs.updateString("Std_lastName","lastName");
115     rs.updateDate("Std_regDate", date);
116     rs.updateFloat("Std_applGrade",15F);
117     rs.updateInt("StdCrs_id",1);
118     rs.updateDate("Std_birthDate",bDate);
119     rs.updateString("Std_eMail","email@ua.pt");
120     rs.updateRow();
121 }
122 Uup time=System.nanoTime()-start;
```

Figure 23. S-JDBC: source code for the $U_{cu}(c_{fu,su},n_{all})$.

Figure 24 presents the general behavior of the $U_{cu}(c_{fu,su},n_{all})$. The dashed lines for $C_{fu,su}$ are overlapped when η>=10, showing that the behaviors are practically the same for both contexts.

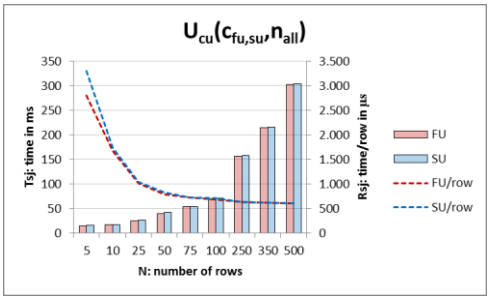


Figure 24. S-JDBC: behavior of $U_{cu}(c_{fu,su},n_{all})$.

From Figure 24 we may conclude that:
- T_{sj} of $C_{fu,su}$ increases with η;
- For η>=10, the behavior is the same for both contexts.
- R_{sj} decreases for $C_{fu,su}$ when η increases; it ranges from about 3,000µs till 600µs.
- For η>=50, T_{sj} tends to be constant.

From Figure 22 and Figure 24 we conclude that committing the changes to the database causes a significant increase in T_{sj} and R_{sj} in about 25 times for all η. This means that any improvement in CRUD-DOM will very probably convey a minor effect in $C_{fu,su}$.

Main point: for η>=10, there is no difference between C_{fu} and C_{su}.

U_{in} – insert unit without commit

Figure 25 depicts the main source code for the $U_{in}(c_{fu,su},n_{all})$. This unit is focused on measuring the required time to insert η tuples into the ResultSet but without committing them to the database.

Figure 26 presents the general behavior of the $U_{in}(c_{fu,su},n_{all})$. The dashed lines for $C_{fu,su}$ are always overlapped showing that the behaviors are the same for both contexts.

```
125  // Insert unit without commit
126  start=System.nanoTime();
127  while ( n-- > 0) {
128      rs.moveToInsertRow();
129      rs.updateString("Std_firstName","firstName");
130      rs.updateString("Std_lastName","lastName");
131      rs.updateDate("Std_regDate", date);
132      rs.updateFloat("Std_applGrade",15F);
133      rs.updateInt("StdCrs_id",1);
134      rs.updateDate("Std_birthDate",bDate);
135      rs.updateString("Std_eMail","email@ua.pt");
136  }
137  Uin_time=System.nanoTime()-start;
```

Figure 25. S-JDBC: source code for the $U_{in}(c_{fu,su},n_{all})$.

From Figure 26 we may conclude that:

- T_{sj} of $C_{fu,su}$ increases with η.
- The behavior is the same for both contexts.
- R_{sj} decreases for $C_{fu,su}$ when η increases; it ranges from 128µs till 19µs.
- For η >=50, T_{sj} tends to be constant.

Main point: there is no difference between C_{fu} and C_{su}.

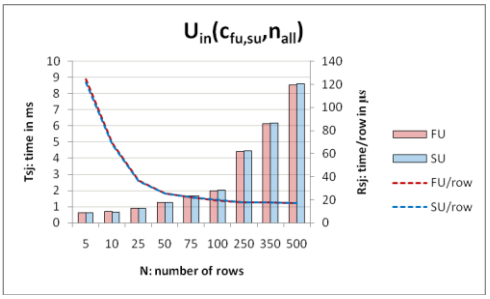


Figure 26. S-JDBC: behavior of $U_{in}(c_{fu,su},n_{all})$.

U_{ci} – insert unit with commit

Figure 27 depicts the main source code for the $U_{ci}(c_{fu,su},n_{all})$. This unit is focused on measuring the required time to insert η into the ResultSet and to commit them to the database.

```
139  // Insert unit with commit
140  start=System.nanoTime();
141  while (rs.next()) {
142      rs.moveToInsertRow();
143      rs.updateString("Std_firstName","firstName");
144      rs.updateString("Std_lastName","lastName");
145      rs.updateDate("Std_regDate", date);
146      rs.updateFloat("Std_applGrade",15F);
147      rs.updateInt("StdCrs_id",1);
148      rs.updateDate("Std_birthDate",bDate);
149      rs.updateString("Std_eMail","email@ua.pt");
150      rs.insertRow();
151  }
152  Uci time=System.nanoTime()-start;
```

Figure 27. S-JDBC: source code for the $U_{ci}(c_{fu,su},n_{all})$.

Figure 28 presents the general behavior of the $U_{ci}(c_{fu,su},n_{all})$. The dashed lines for $C_{fu,su}$ are very close conveying the need for a more detailed graphic. Figure 29 presents a more detailed view of Figure 28 emphasizing the differences between the behaviors of the 2 contexts.

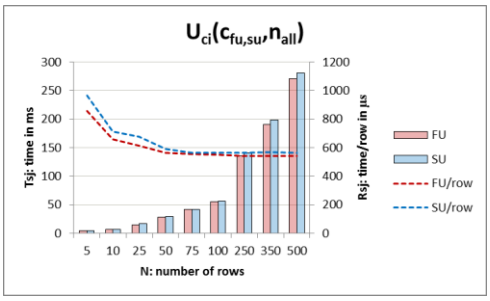


Figure 28. S-JDBC: behavior of $U_{ci}(c_{fu,su},n_{all})$.

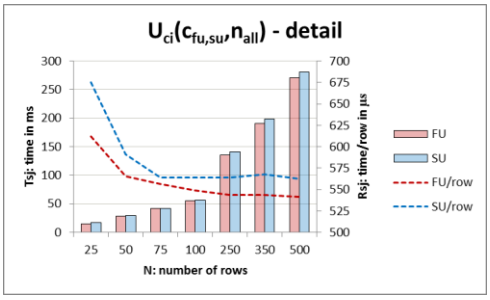


Figure 29. S-JDBC: detail of $U_{ci}(c_{fu,su},n_{all})$.

From Figure 28 and Figure 29 we may conclude that:

- T_{sj} of $C_{fu,su}$ increases with η.
- The behavior is very similar to both contexts.
- R_{sj} is weakly dependent on η for values of η >=75.
- C_{fu} gets better scores for all η.

From Figure 28 and Figure 29 we conclude that committing the new tuples to the database causes an increase in T_{sj} that ranges from 8 times for η =5 till 25 times for N=500.

Main point: scrollable ResultSets should always be avoided whenever possible.

U_{de} – delete unit

Figure 30 depicts the main source code for the $U_{de}(c_{fu,su},n_{all})$. This unit is focused on measuring the required time to delete all tuples returned by the select statement.

```
150  // Delete Unit
151  start=System.nanoTime();
152  while (rs.next())
153      rs.deleteRow();
154  Ude_time=System.nanoTime()-start;
```

Figure 30. S-JDBC: source code for the $U_{de}(c_{fu,su},n_{all})$.

Figure 31 presents the general behavior of the $U_{de}(c_{fu,su},n_{all})$. The dashed lines for $C_{fu,su}$ are very close conveying the need for a more detailed graphic. Figure 32 presents a more detailed view of Figure 31 emphasizing the differences between the behaviors of the 2 contexts. From Figure 31 and Figure 32 we may conclude that:

- T_{sj} of $C_{fu,su}$ increase with η.

- The behavior is very similar to both contexts.
- For η>=50, R_{sj} tends be independent from η.
- C_{fu} gets better scores for all η.

Main point: if possible, scrollable ResultSets should always be avoided mainly when the number of rows is low.

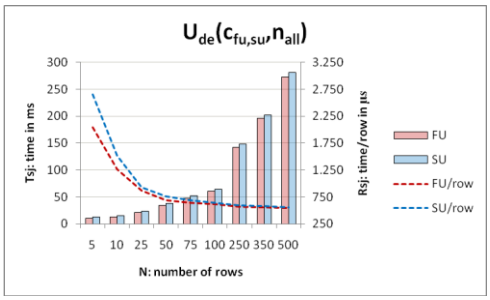


Figure 31. S-JDBC: behavior of $U_{de}(c_{fu,su},n_{all})$.

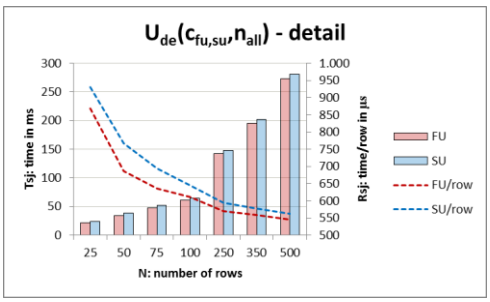


Figure 32. S-JDBC: detail of $U_{de}(c_{fu,su},n_{all})$.

Summary

Despite some particularities, as a summary of all units, we may say that:

- $C_{fr,fu}$ have better scores than $C_{sr,su}$.
- Most of the times, C_{sr} have better scores than C_{su}.
- T_{sj} increases with η except for $C_{fr,fu}$ in U_{se}.
- R_{sj} decays when η increases except for C_{fr} in U_{re}.
- R_{sj} decays rapidly from η=5 till η=50 or η>75.
- R_{sj} tends to be constant for η>=50 or η>=75

The collected measures come in line with the knowledge about the TDS protocol [46] and its implementation on the client side and on the server side. Scrollable and updatable ResultSets always use a cursor and a dataset inside the Sql Server. The cursor management and the selected row in the client side are always synchronized leading this way a decrease in the overall performance. This characteristic will also have impact in the next assessment.

2) Assessment of scenarios

In spite of being an important issue, the scenarios have been introduced only to simulate situations closer to a hypothetical situation. Therefore, we only briefly present some results for the assessment of the six scenarios. Figure 33, Figure 34, Figure 35, Figure 36, Figure 37 and Figure 38 present the individual behavior for each scenario.

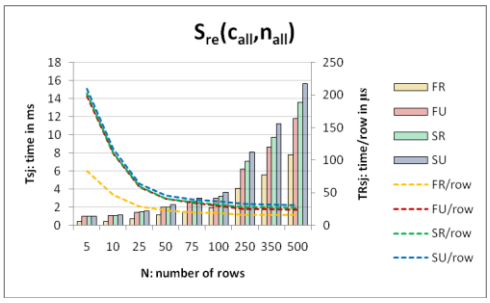


Figure 33. S-JDBC: behavior of $S_{re}(c_{all},n_{all})$

The main idea to be emphasized is that the global behavior of each scenario follows the global behavior of the correspondent unit. The measures for each η and each context are now increased by adding the correspondent collected value for U_{se}. The weight of U_{se} is almost unnoticeable for $S_{cu,ci,de}$. This derives from the fact that these contexts have very high T_{sj}. Anyway, the weight of U_{se} is not

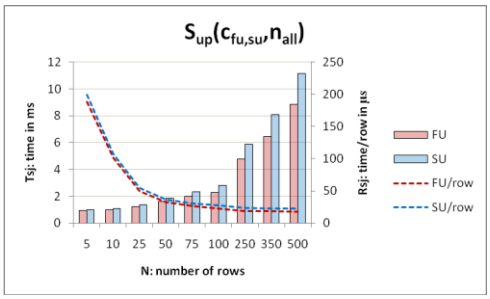


Figure 34. S-JDBC: behavior of $S_{up}(c_{fu,su},n_{all})$

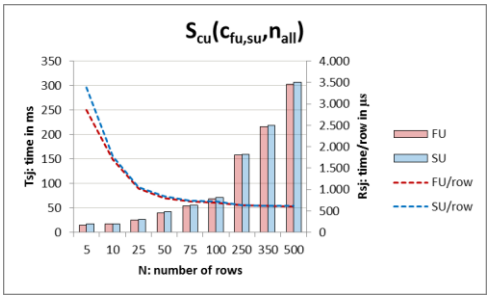


Figure 35. S-JDBC: behavior of $S_{cu}(c_{fu,su},n_{all})$

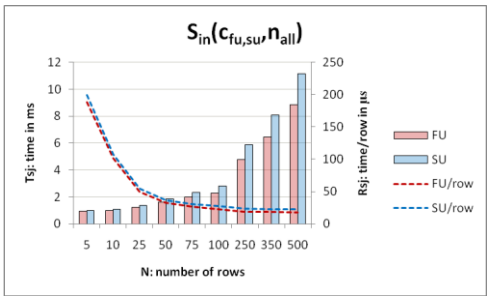


Figure 36. S-JDBC: behavior of $S_{in}(c_{fu,su},n_{all})$

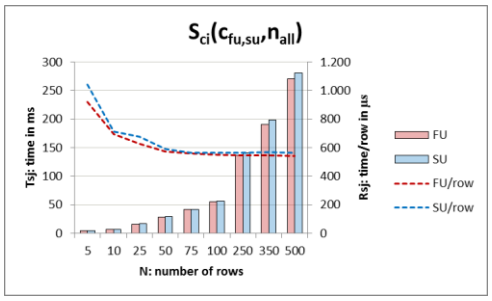


Figure 37. S-JDBC: behavior of $S_{ci}(c_{fu,su},n_{all})$

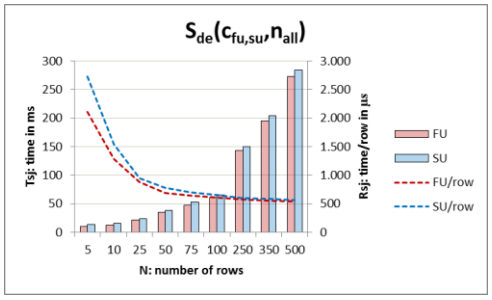


Figure 38. S-JDBC: behavior of $S_{de}(c_{fu,su},n_{all})$

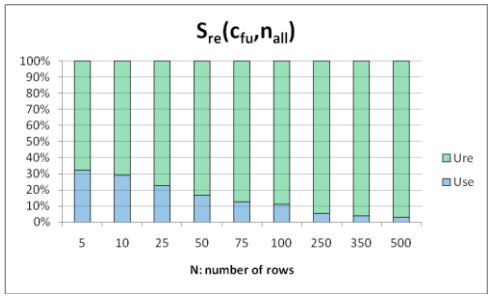


Figure 39. S-JDBC: weight of each unit in the $S_{re}(c_{fu},n_{all})$

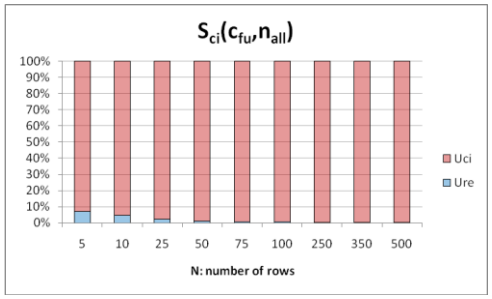


Figure 40. S-JDBC: weight of each unit in the $S_{ci}(c_{fu},n_{all})$

constant neither for each context nor for each η. Two examples for C_{fu} are shown in Figure 39 and Figure 40. In these graphics each column represents the relative weight of each unit for the total measured value. They show the relative weight of each unit in the $S_{re}(c_{fu},n_{all})$ and $S_{ci}(c_{fu},n_{all})$, respectively. As expected, U_{se} has a higher weight in S_{re} than in S_{ci} for all η.

C. C-CRUD assessment

C-CRUD assessment, as mentioned before, will only comprise units. Scenarios will not be addressed because the defined scenarios are only one among infinity of possibilities. Moreover, each scenario conveys a similar behavior as the correspondent main units (others than U_{re}) as has been shown for S-JDBC.

C-CRUD assessment will be presented through graphics that show the differences between S-JDBC and C-CRUD. In all graphics, the bars represent the time required to execute a unit (T_{cc}) and the dashed lines represent the % of the difference between S-JDBC and C-CRUD (V_{cc}) = (C-CRUD)-(S-JDBC)/(C-JDBC). The axis for the bars is the left one and the axis for the dashed lines is the right one.

The main source code for the implementation of C-CRUD basically differs from the depicted code for S-JDBC on the usage of the type-state interfaces. The main structure is equal on both entities. Anyway, we will always present the source code in order to provide a better context for the understanding of how each unit was assessed. The CRUD-DO's name is *Student*.

U_{se} – select unit

Figure 41 depicts the main source code for the $U_{se}(c_{all},n_{all})$. No differences were detected between S-JDBC and C-CRUD and therefore there is no need to present the correspondent graphic. $U_{se}(c_{all},n_{all})$ behavior for S-JDBC is presented in Figure 16 and Figure 17.

```
86  // select unit
87  start=System.nanoTime();
88  student.execute();
89  Use_time=System.nanoTime()-start;
```

Figure 41. C-CRUD: source code for the $U_{se}(c_{all},n_{all})$.

U_{re} – read unit

Figure 42 depicts the main source code for the $U_{re}(c_{all},n_{all})$. Figure 43 presents the general behavior of the $U_{re}(c_{all},n_{all})$.

```
92  // read unit
93  rStudent=student.beginRead();
94  start=System.nanoTime();
95  while (student.moveNext()) {
96      id=rStudent.gId();
97      firstName=rStudent.gFirstName();
98      lastName=rStudent.gLastName();
99      rDate=rStudent.gRegDate();
100     applGrade=rStudent.gApplGrade();
101     crsId=rStudent.gCrsId();
102     bDate=rStudent.gBirthDate();
103     eMail=rStudent.gEMail();
104 }
105 Ure_time=System.nanoTime()-start;
```

Figure 42. C-CRUD: source code for the $U_{re}(c_{all},n_{all})$.

From this figure we may conclude that:

- V_{cc} decreases for all contexts when η increases; C_{fr} is the most independent one.

- The variation of V_{cc} along η is very similar to all contexts
- The behavior of C_{fr} has the largest difference to S-JDBC. This derives from the fact that $U_{re}(C_{fr})$ in S-JDBC has by far the best scores leading to the situation where any C-CRUD overhead implies a stronger impact.
- $C_{fu,sr.su}$ have very similar differences to S-JDBC.
- Vcc for C_{fr} range from about 2.9% till 2.6%
- V_{cc} for the other contexts range from about 2.6% till 2.15%

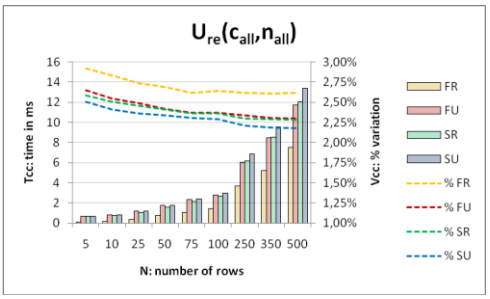


Figure 43. C-CRUD: behavior of $U_{re}(c_{all},n_{all})$.

U_{up} – update unit without commit

Figure 44 depicts the main source code for the $U_{up}(c_{fu,su},n_{all})$. Figure 45 presents the general behavior of the $U_{up}(c_{fu,su},n_{all})$.

```
// update unit without commit
start=System.nanoTime();
while (student.moveNext()) {
    uStudent=student.beginUpdate();
    uStudent.sFirstName( "firstName");
    uStudent.sLastName("lastName");
    uStudent.sRegDate(rDate);
    uStudent.sApplGrade(1F);
    uStudent.sCrsId(1);
    uStudent.sBirthDate(bDate);
    uStudent.sEMail("eMail@gmail.com");
    uStudent.update(); // empty method
}
Uup_time=System.nanoTime()-start;
```

Figure 44. C-CRUD: source code for the $U_{up}(c_{fu,su},n_{all})$.

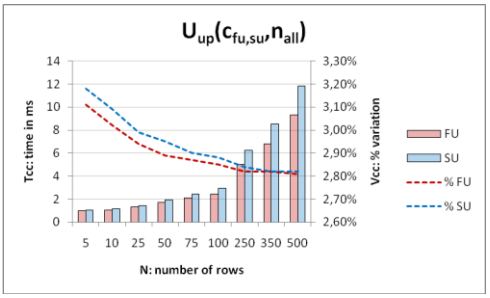


Figure 45. C-CRUD: behavior of $U_{up}(c_{fu,su},n_{all})$.

From this figure we may conclude that:

- V_{cc} decreases for all contexts when η increases.
- C_{su} has the largest difference to S-JDBC but they converge from η=5 till overlap for η>350.
- For C_{su}, V_{cc} ranges from about 3.2% till 2.8%.
- For C_{fu}, V_{cc} ranges from about 3,1% till 2.8%

U_{cu} – update unit with commit

Figure 46 depicts the source code for the $U_{cu}(c_{fu,su},n_{all})$. Figure 47 presents the general behavior of the $U_{cu}(c_{fu,su},n_{all})$. From this figure we may conclude that:

- The maximum variation of V_{cc} is 0.01% in each context.
- V_{cc} for C_{su} is always higher than for C_{fu}.
- V_{cc} ranges from 0.01% till 0.03%. The low impact of C-CRUD derives from the relative very low overhead introduced by C-CRUD. The *commit* operation is very slow weakening this way the relative weight of C-CRUD overhead.

```
// update unit with commit
start=System.nanoTime();
while (student.moveNext()) {
    uStudent=student.beginUpdate();
    uStudent.sFirstName( "firstName");
    uStudent.sLastName("lastName");
    uStudent.sRegDate(rDate);
    uStudent.sApplGrade(1F);
    uStudent.sCrsId(1);
    uStudent.sBirthDate(bDate);
    uStudent.sEMail("eMail@gmail.com");
    uStudent.update(); // not empty method
}
Ucu_time=System.nanoTime()-start;
```

Figure 46. C-CRUD: source code for the $U_{cu}(c_{fu,su},n_{all})$.

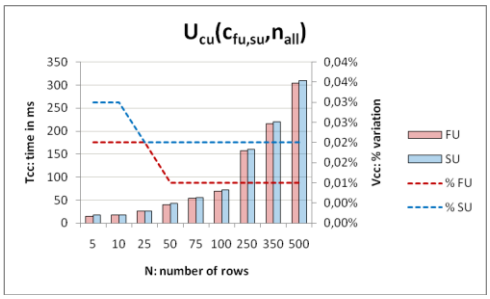


Figure 47. C-CRUD: behavior of the $U_{cu}(c_{fu,su},n_{all})$.

U_{in} – insert unit without commit

Figure 48 depicts the main source code for the $U_{in}(c_{fu,su},n_{all})$. The method *insert()* is an empty method avoiding this way committing new tuples to the database.

Figure 49 presents the general behavior of the $U_{in}(c_{fu,su},n_{all})$. From this figure we may conclude that:

- V_{cc} decreases for all contexts when η increases.
- C_{su} has the largest difference to S-JDBC but its difference to C_{fu} is minimal and converges to zero for η=500.
- For C_{su}, V_{cc} ranges from about 3.2% till <2.8%
- For C_{fu}, V_{cc} ranges from about 3,18% till <2.8%

```
139  // insert unit without commit
140  start=System.nanoTime();
141  while (n-- >0) {
142      iStudent=student.beginInsert();
143      iStudent.sFirstName("firstName");
144      iStudent.sLastName("lastName");
145      iStudent.sRegDate(rDate);
146      iStudent.sApplGrade(1F);
147      iStudent.sCrsId(1);
148      iStudent.sBirthDate(bDate);
149      iStudent.sEMail("eMail@gmail.com");
150      iStudent.insert(); // empty method
151  }
152  Uin_time=System.nanoTime()-start;
```

Figure 48. C-CRUD: source code for the $U_{in}(c_{fu,su},n_{all}$

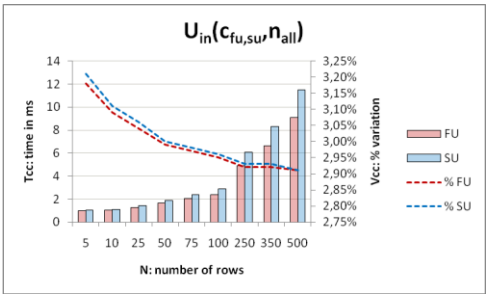


Figure 49. C-CRUD: behavior of the $U_{in}(c_{fu,su},n_{all})$.

U_{ci} – insert unit with commit

Figure 50 depicts the main source code for the $U_{ci}(c_{fu,su},n_{all})$. In opposite to $U_{in}(c_{fu,su},n_{all})$ the method *insert()* commits new tuples to the database.

```
155  // insert unit with commit
156  start=System.nanoTime();
157  while (n-- >0) {
158      iStudent=student.beginInsert();
159      iStudent.sFirstName( "firstName");
160      iStudent.sLastName("lastName");
161      iStudent.sRegDate(rDate);
162      iStudent.sApplGrade(1F);
163      iStudent.sCrsId(1);
164      iStudent.sBirthDate(bDate);
165      iStudent.sEMail("eMail@gmail.com");
166      iStudent.insert(); // not empty method
167  }
168  Uci_time=System.nanoTime()-start;
```

Figure 50. C-CRUD: source code for the $U_{ci}(c_{fu,su},n_{all})$.

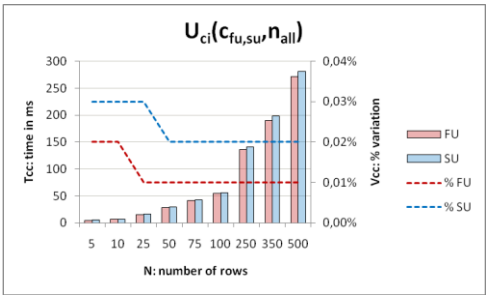


Figure 51. C-CRUD: behavior of $U_{ci}(c_{fu,su},n_{all})$.

Figure 51 presents the general behavior of the $U_{ci}(c_{fu,su},n_{all})$. From this figure we may conclude that:

- The maximum variation of V_{cc} along η for each context is at most 0.01%.
- V_{cc} for C_{su} is about twice the value of C_{fu}. Anyway, the involved absolute values are very small.
- V_{cc} ranges from 0.01% till 0.03%. The low impact of C-CRUD derives from the relative very low overhead introduced by C-CRUD. The *commit* operation is very slow weakening this way the relative weight of C-CRUD overhead.

U_{de} – Unit delete

Figure 52 depicts the main source code for $U_{de}(c_{fu,su},n_{all})$.

```
171  // delete unit
172  start=System.nanoTime();
173  while (student.moveNext())
174      student.delete();
175  Ude_time=System.nanoTime()-start;
176
```

Figure 52. C-CRUD: source code for the $U_{de}(c_{fu,su},n_{all})$.

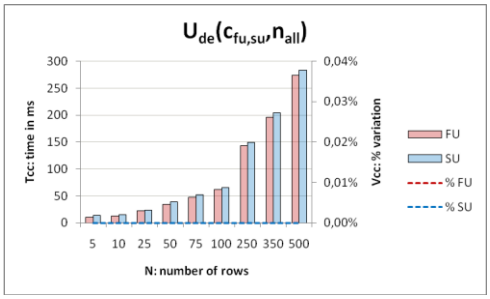


Figure 53. C-CRUD: behavior of $U_{de}(c_{fu,su},n_{all})$.

Figure 53 presents the general behavior of the $U_{de}(c_{fu,su},n_{all})$. From this figure we may conclude that V_{cc} is so small for both contexts that it is not possible to represent them in the graphic. This derives from the fact that the *delete* operation is too slow and also from the fact that S-JDBC and C-CRUD implementations are very similar.

Summary

Despite some particularities, as a summary of all units, we may say that:

- Between units, the weight of V_{cc} decreases when η increases.
- For slower units ($U_{cu,ci,de}$) the C-CRUD overhead is lower than 0.03%.
- For faster units ($U_{re,up,in}$) the C-CRUD overhead ranges from 3.2% till 2.4%.

VII. CONCLUSION

The solution here presented proved to be effective for bridging the gap between the object oriented and the relational paradigms in the context where programmers have no alternative but write the required CRUD expressions to implement the middle tier. This may occur in situations

where CRUD expressions cannot be derived from any data model and mainly in applications where CRUD expressions are complex or very complex. The span of its effectiveness relies on two main dimensions: the model itself and the CRUD-M.

The model itself: CRUD-DOM addresses the following issues:

- CRUD-DOM encapsulates CRUD expressions of any complexity and exposes an object-oriented interface to the assisted application translating this way the row/table oriented paradigm into the object-oriented paradigm; the encapsulation hides all the complexity for the communication between the two paradigms tackling this way the impedance mismatch issue for the present context, which is focused on static customized CRUD expressions;
- interfaces are strongly-typed and type-state oriented providing this way an improved usability and productivity;
- it is amenable to the development addressing automatic code generation improving this way programmers productivity;
- CRUD-DOM totally relies on JDBC and copes with requirements as SQL expressiveness and system performance;
- it does not rely on any complementary or proprietary technology; the version here presented is based on Java but CRUD-DOM may be implemented in any other object-oriented programming language;
- it promotes the development of intermediate access layers this way decoupling applications and databases tiers and, therefore, leveraging this way the separation of concerns.

CRUD-M: CRUD-M addresses the following issues:

- from user defined SQL statements CRUD-M automatically creates all the necessary source code to implement the correspondent CRUDDOs, promoting this way programmers productivity;
- CRUD-M provides the programmers an automatic mechanism to test SQL statements promoting this way their productivity;
- CRUD-M allows programmers to easily update existing CRUDDO promoting this way their productivity.

So, the collaboration and interdependence between CRUD-DOM and CRUD-M is a key issue to achieve the three announced goals: 1) programmers' productivity – less time to develop, test and maintain middle tiers; 2) middle tier performance is kept at a level very similar to the standard JDBC API and, 3) usability is significantly improved when compared with the standard JDBC.

Regarding CRUD-DOM performance, despite the limited range of tests, the obtained results show that in most database applications the induced overhead may be considered as perfectly acceptable. Anyway, for very demanding database applications some additional attention should be given to CRUD-DOM, mainly for faster units, in order to minimize its overhead. Improving the performance of the slower units is beyond the programmer's scope. Most of the time is spent on updating the state of the database. Thus, it is expected, for these slower units, that any improvement in the source code should have a negligible impact on performance.

The automatic source code development tool, CRUD-M, designed as proof of concept, proved to be an efficient tool addressing all features of CRUD-DOM in an integrated way. Programmers are only required to input customized SQL statements of any complexity. CRUD-M relieves programmers from writing and testing any source code addressing this way the productivity requirement. Additionally, it provides an interactive GUI where programmers are guided step by step, since the editing of CRUD expressions till the creation of CRUD-DO addressing this way the usability requirement.

Some small differences in the final results between this assessment and [1] derives from the fact that the environments in which they took place are slightly different. Anyway, the fundamental conclusions and the collected results are basically identical. CRUD-DOM induces an overhead that for most of the database applications may be considered as not significant. Anyway, some more attention is needed to minimize the CRUD-DOM overhead in order to address very demanding database applications.

A new version of CRUD-DOM is being prepared. This new version will support several mechanisms of concurrency promoting this way CRUD-DOM performance in new directions. Additional new features will be also included in order to support current JDBC features. Among them:

- to provide support for the execution of SQL statements in batch mode;
- to provide support to execute stored procedures;
- to provide support to allow programmers to choose at runtime between *statements* and *preparedStatements*;

It is expected that CRUD-DOM and CRUD-M may be used in database applications where the middle tier is not a direct object-oriented perspective of relational models as happens with O/RM tools. CRUD-DOM and CRUD-M impact may be significant in database applications where CRUD expressions are very complex. Without the support of CRUD-DOM and CRUD-M, complex CRUD expressions are not easy to write, test, maintain and wrapped in a structure identical to CRUD-DOM.

REFERENCES

[1] O. M. Pereira, R. L. Aguiar, and M. Y. Santos, "CRUD-DOM: A Model for Bridging the Gap Between the Object-Oriented

and the Relational Paradigms," in *ICSEA 2010 - International Conference on Software Engineering and Applications*, Nice, France, 2010, pp. 114-122.

[2] M. David, "Representing database programs as objects," in *Advances in Database Programming Languages*, F. Bancilhon and P. Buneman, Eds., ed N.Y.: ACM, 1990, pp. 377-386.

[3] ODBMS.ORG. (2011 May). *Integrating programming languages and databases: what is the problem?* Available: http://www.odbms.org/experts.aspx#article10

[4] *Part 1: SQL Routines using the Java (TM) Programming Language*, 1999.

[5] Microsoft Corporation. (2011 May). *The LINQ Project*. Available: http://msdn2.microsoft.com/en-us/netframework/aa904594.aspx

[6] ISO. (2011 May). *ISO/IEC 9075-3:2003*. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=34134

[7] Oracle. (2011 May). *JDBC Overview*. Available: http://www.oracle.com/technetwork/java/overview-141217.html

[8] Microsoft. (2011 May). *Microsoft Open Database Connectivity*. Available: http://msdn.microsoft.com/en-us/library/ms710252(VS.85).aspx

[9] Microsoft Corporation. (2011 May). *Overview of ADO.NET*. Available: http://msdn.microsoft.com/en-us/library/h43ks021(VS.71).aspx

[10] B. Christian and K. Gavin, *Hibernate in Action*: Manning Publications Co., 2004.

[11] Oracle. (2011 May). *Oracle TopLink*. Available: http://www.oracle.com/technetwork/middleware/toplink/overview/index.html

[12] Oracle. (2011 May). *Java Data Objects (JDO)*. Available: http://www.oracle.com/technetwork/java/index-jsp-135919.html

[13] D. Yang, *Java Persistence with JPA*: Outskirts Press, 2010.

[14] C. U. Smith and L. G. Williams, *Performance Solutions: a Practical Guide to Creating Responsive, Scalable Software*, 1st ed.: Addison Wesley, 2001.

[15] M. Woodside, G. Franks, and D. C. Petriu, "The Future of Software Performance Engineering," presented at the FOSE '07- Future of Software Engineering, Minneapolis,MN,USA, 2007.

[16] IEEE, *SWEBOK*, 2004 ed. Los Alamitos,CA: IEEE Computer Society.

[17] J. Nielson, *Usability Engineering*. San Francisco, CA: Morgan Kaufman, 1993.

[18] D. G. John and L. Clayton, "Designing for Usability: Key Principles and What Designers Think," *Communications of the ACM*, vol. 28, pp. 300-311, 1985.

[19] *ISO 9241-11: Ergonomic Requirements for Office Work With Visual Display Terminals*, fdew, 1998.

[20] *ISO 13407: Human-Centered Design Processes for Interactive Systems*, 1999.

[21] *ISO/IEC 9126-1: Software Engineering - Product Quality*, 2001.

[22] *ISO/TR 16982: Usability Methods Supporting Human Centered Design*, 2002.

[23] M. J. Suárez-Cabal and J. Tuya, "Using an SQL coverage measurement for testing database applications," presented at the FSE'04 - ACM SIGSOFT 12th International Symposium on Foundations of Software Engineering, Newport Beach-CA-USA, 2004.

[24] E. Vincent, "Is ISSTA research relevant to industrial users? panel - ISSTA 2002: empowering the developer to be a tester too!," *ACM SIGSOFT Software Engineering Notes*, vol. 27, pp. 203-204, 2002.

[25] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," ed: National Institute of Standards and Technology, 2002, pp. Planning Report 02-3.

[26] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," presented at the FOSE '07- Future of Software Engineering, Minneapolis,MN,USA, 2007.

[27] Y. Singh and B. Goel, "A step towards software preventive maintenance," *ACM SIGSOFT Software Engineering Notes*, vol. 32, 2007.

[28] B. P. Lientz and E. B. Swanson, *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Reading,MA: Addison Wesley, 1980.

[29] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," presented at the FOSE'00 - Future of Software Engineering, Limerick,Ireland, 2000.

[30] J. W. Moore, "The ANSI binding of SQL to ADA," *Ada Letters*, vol. XI, pp. 47-61, 1991.

[31] W. Keller, "Mapping Objects to Tables - A Pattern Language," in *European Conference on Pattern Languages of Programming Conference (EuroPLoP)*, Irsse, Germany, 1997.

[32] R. Lammel and E. Meijer, "Mappings Make data Processing Go 'Round: An Inter-paradigmatic Mapping Tutorial," in *Generative and Transformation Techniques in Software Engineering*, Braga, Portugal, 2006.

[33] C. Pablo, M. Sergey, and A. Atul, "ADO.NET entity framework: raising the level of abstraction in data programming," in *ACM SIGMOD International Conference on Management of Data*, Beijing,China, 2007, pp. 1070-1072.

[34] R. C. William and R. Siddhartha, "Safe query objects: statically typed objects as remotely executable queries," in *27th International Conference on Software Engineering*, St. Louis, MO, USA, 2005, pp. 97-106.

[35] A. M. Russell and H. K. Ingolf, "SQL DOM: compile time checking of dynamic SQL statements," in *27th International Conference on Software Engineering*, St. Louis, MO, USA, 2005, pp. 88-96.

[36] W. Gary, G. Carl, S. Zhendong, and D. Premkumar, "Static checking of dynamically generated queries in database applications," *ACM Transansactions on Software Eng. Methodology*, vol. 16, p. 14, 2007.

[37] Microsoft Corporation. (2011 May). *ADO.NET*. Available: http://msdn.microsoft.com/en-us/library/aa286484.aspx

[38] M. Andy, E. Wolfgang, and S. R. David, "Impact analysis of database schema changes," in *30th International Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 451-460.

[39] B. Gregory, W. W. Bruce, and A. G. S. Paolo, "Using parse tree validation to prevent SQL injection attacks," in *5th International Workshop on Software Engineering and Middleware*, Lisbon, Portugal, 2005.

[40] Oracle. (2011 May). *Interface Statement*. Available: http://download.oracle.com/javase/6/docs/api/java/sql/Statement.html

[41] Oracle. (2011 May). *Interface ResultSet*. Available: http://download.oracle.com/javase/6/docs/api/java/sql/ResultSet.html

[42] Oracle. (2011 May). *Interface PreparedStatement*. Available: http://download.oracle.com/javase/6/docs/api/java/sql/PreparedStatement.html

[43] R. E. Strom and S. Yemini, "Typestate: A programming language concept for enhancing software reliability," *IEEE Transactions on Software Engineering*, vol. 12, pp. 157-171, 1986.

[44] R. E. Strom and S. Yemini, "Typestate: A programming language concept for enhancing software reliability," *IEEE Trans. Softw. Eng.*, vol. 12, pp. 157-171, 1986.

[45] Microsoft. (2011 May). *SQL Server JDBC Driver 2.0 Documentation*. Available: http://technet.microsoft.com/en-us/library/ff928320(SQL.10).aspx

[46] Microsoft. (2011 May). *[MS-TDS]: Tabular Data Stream Protocol Specification*. Available: http://msdn.microsoft.com/en-us/library/dd304523(v=prot.13).aspx

Performance Evaluation of a High Precision Software-based Timestamping Solution for Network Monitoring

Peter Orosz, Tamas Skopko
Faculty of Informatics
University of Debrecen
Debrecen, Hungary
e-mail: oroszp@unideb.hu, skopkot@unideb.hu

***Abstract* — Widely used network measurement applications, such as tcpdump and Wireshark, use the same common libpcap packet capture library. Libpcap assigns a 10^{-6} second precision timestamp to all processed frames. Higher physical bandwidth implies shorter inter-arrival times between consecutive frames. Accordingly timestamp resolution must be proportional to the link speed. The latest version 1.1.x of libpcap provides 10^{-6} second native resolution, however pcap format supports a larger 2 x 32-bit timestamp value for each stored packet. On Gigabit Ethernet or faster networks, a timestamp resolution that works in the microsecond domain may not enable us to precisely reproduce the time-domain relation between consecutive frames. Therefore overall analysis of the data transmission could lead to a false result. For packet capturing with libpcap, it is assumed that the timestamp represents the time moment when a frame reaches the kernel's input packet queue. In an idealized case generated timestamps are always converging and suitably close to the real arrival or transmission time of each frame so as to conserve the original inter-arrival time values. The timestamp resolution of network measurement applications must be increased according to the requirements of advanced high speed data networks. In our paper, we are going to show and evaluate an alternative libpcap-based solution that features nanosecond precision timestamping.**

Keywords-libpcap; timestamp resolution; inter-arrival time; Linux kernel; high speed network.

I. INTRODUCTION

This paper is the extended version of our previous work [1] that focuses on a software solution based on the libpcap packet capture library and high resolution kernel-based timestamp generation. On Linux machines the libpcap library retrieves timestamps of captured frames from the kernel through some special kernel functions. Independently from one other, several impact factors could directly bias the generation of timestamps [2].

Some timestamp-related terms that will be used in the rest of this paper should be introduced here:

- Timestamp size *(TSS)*: bit length of the timestamp
- Timestamp precision *(TSP)*: sub-second resolution
- Timestamping time *(TST)*: time required to generate a timestamp value

High resolution timestamping of data packets on high speed networks is a challenging issue [3], which is even more critical on a software-based packet capture environment such as libpcap [4]. Libpcap relies on the operating system kernel to provide the arrival or transmission time moment of the processed data packets. Since timestamping is performed in the kernel space, several hardware and software factors impact the overall precision and accuracy of the generated timestamps. Furthermore, data structures in libpcap are designed for 32-bit TSS.

The precision requirement of TSP depends on:

- Link speed
- The minimum of packet inter-arrival times within a data stream

The following factors affect TST:

- Hardware architecture
- NIC driver design
- OS kernel (enqueing/dequeuing, handlers)
- Clock sources
- Libpcap

Let us suppose that two uniform sequences of minimum-sized and maximum-sized Ethernet frames are transmitted over Gigabit and Ten Gigabit Ethernet links at the theoretical maximum rate. Table 1 and Table 2 show the PHY (physical layer) level timing parameters of the Gigabit and Ten Gigabit Ethernet standards.

TABLE I. GIGABIT ETHERNET TIME PARAMETERS

Timing parameters 1 GbE	**Smallest Ethernet frame length: 72 Bytes**	**Largest Ethernet frame length: 1526 Bytes**
Bit time	1 ns	1 ns
Inter-frame gap	96 ns = 96 x bit time	96 ns = 96 x bit time
Δt between timestamps of two consecutive frames	576 ns + 96 ns = **672 ns**	12,208ns + 96ns = 12,304ns
Theoretical precision of NTP sync	≥1 msec	≥1 msec
Required time sync precision	≤600 ns (theoretical minimum)	≤12 µs (theoretical maximum)
Maximum number of frames per second	1,488,096	81,274

TABLE II. TEN GIGABIT ETHERNET TIME PARAMETERS

Timing parameters 10 GbE	Smallest Ethernet frame length: 72 Bytes	Largest Ethernet frame length: 1526 Bytes
Bit time	.1 ns	.1 ns
Inter-frame gap	9.6 ns = 96 x bit time	9.6 ns = 96 x bit time
Δt between timestamps of two consecutive frames	57.6 ns + 9.6 ns = **67.2 ns**	1,220.8 ns + 9.6 ns = 1,230 ns
Theoretical precision of NTP sync	≥1 msec	≥1 msec
Required time sync precision	≤60.0 ns (theoretical minimum)	≤1.2 µs (theoretical maximum)
Maximum number of frames per second	14,880,960	812,740

We assume that the indicated frame sizes include 8 bytes preamble, 6 bytes destination MAC address, 6 bytes Source MAC address, 2 bytes MAC Type/length, 4 bytes CRC and the payload (46-1500 bytes). The inter-frame gap is 12 bytes according to the Ethernet specification. Based on these values the minimum of packet inter-arrival time can be determined that is 672 ns for Gigabit Ethernet and 67.2 ns for Ten Gigabit Ethernet respectively.

IP Packet Delay Variation (IPDV) is an IETF RFC 3393 proposal [5][6]:

$$d_{T2} - d_{T1} = d_{\Delta T} \quad (1)$$

where
d_{T1}... delay of a packet sent at time T_1
d_{T2} ... delay of a packet sent at time T_2
$d_{\Delta T}$ ... type - P - one - way - ipdv from source to destinatio n

Delay per hop:

$$d_H = d_t + d_p + d_q \quad (2)$$

where
d_H ... delay per hop
d_t ... transmiss ion delay
d_p ... processing delay
d_q ... queuing delay

End-to-end delay:

$$d_T = \sum_{i=1}^{n} d_{H_i} \quad (3)$$

where
d_T ...end - to - end delay
d_H ...delay per hop
n... number of hops

We assume that (3) has a Gamma distribution function [7]:

$$f(x;k;\theta) = x^{k-1}\frac{e^{-x/\theta}}{\theta^k \Gamma(k)} \quad (4)$$

where $x, k, \theta > 0$

A 64-Byte packet sequence has been generated according to (4) at a 1 Gbps transmission rate (Fig. 1).

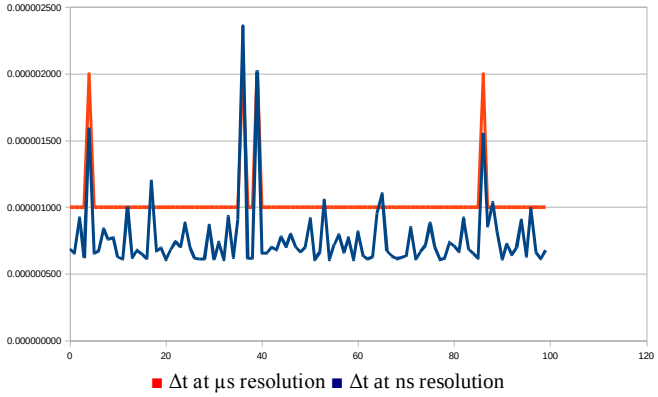


Figure 1. Gamma distributed PDV of 64-Byte frames at 1Gbps transmission rate

It can easily be shown that microsecond time resolution could be insufficient to describe the time domain relation (1)(2) between packet arrivals on Gbit/s or a higher speed network path [8].

II. RELATED WORK

In the last couple of years several research projects have realized the problem of the inefficient time resolution of packet timestamps [9][10][11]; most of their proposals and solutions resulted in hardware-based packet timestamping. For higher performance some of them integrated the entire capturing process into a dedicated hardware device [9][11][12][13]. However, none of them was focused on extending the resolution of software-based packet timestamping.

III. PROBLEM DEFINITION

A. NIC driver architecture

The NIC driver connects the physical layer and the internal packet structures of the operating system. A sophisticated network driver design combines interrupt and polling operation modes using the kernel feature NAPI (New API) [14]: at lower traffic it uses interrupts, while at higher loads it switches to polling mode [15].

Interrupt mode: When a frame arrives at the NIC, it generates an interrupt that calls a specific handler registered by the driver. The handler places the frame into the input packet queue and the kernel processes it thereafter. The handler is given priority over the kernel's processing code as long as frames are arriving at a higher rate (due to a high network load) than the kernel can handle them. High traffic results in a high number of interrupts that could consume hardware resources.

Some NIC drivers can support the passing of multiple frames within an interrupt.

- Polling mode: The kernel queries the driver about the arrival of new frames with a specific frequency. Resource consumption of this method is optimal at high network load.
- Timer-driven interrupts: The NIC asynchronously notifies the kernel about frame reception. The handler processes the frame, which has arrived since the last interrupt.

Since timestamping of the incoming packets is performed by the queue handler, the timestamp value does not necessarily depend on the operation mode of the NIC driver. Nevertheless, in order to determine the dependency of the timestamp value on the operation mode further analysis is required.

B. The OS kernel

We must define the exact code point over the data path from the NIC to the kernel input queue where timestamping is performed. The Linux kernel puts timestamps onto each frame when they are enqueued to the input packet queue. This is the point where the kernel processes the frame (Fig. 1).

Since libpcap relies on kernel timestamps, we had to observe the latest Linux kernel functions, which could acquire 10^{-9} TSP from a high frequency and very accurate clock source.

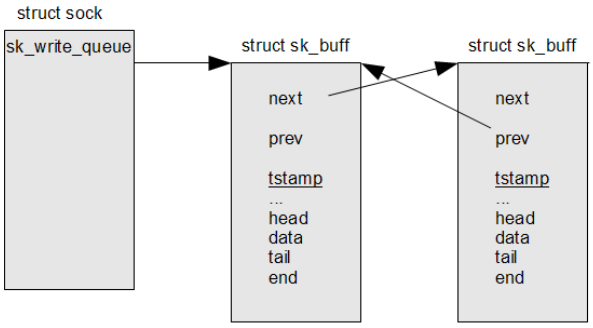


Figure 2. Sk_buff structure of the linux kernel

C. Clock sources

The Linux kernel supports multiple clock sources. Their availability depends on the underlying hardware architecture.

1) ACPI (Advanced Configuration and Power Interface) Power Management Timer

This clock, known as the RTC *(Real Time Clock)*, is usually integrated into the south bridge of the motherboard. Its 3.579545 MHz clock frequency limits its precision to the microsecond domain.

2) HPET (High Precision Event Timer)

This is available on most of today's PC architectures. HPET is a high precision clock source due to its very low jitter, which is within the nanosecond domain. However its clock frequency is about 10 MHz, which is not an appropriate for high resolution timestamping.

3) Jiffies

These are based on timer interrupt and are referred to as the kernel heartbeats. Jiffy frequency can be specified at compile time. Under recent 2.6.x Linux kernels it is set to 1/250 Hz (4 ms resolution) or its maximum 1/1000 Hz (1 ms) by default, which is far from the requirements of proper timestamping. There are plans to remove this timing method and move to tickless systems because of power saving considerations.

4) TSC (Time Stamp Counter)

A 64-bit CPU register that is present on all x86 processors since the Intel Pentium. It counts the number of ticks since boot or reset. The time stamp counter is an excellent high-resolution, low-overhead way of providing timestamps. The novel constant TSC feature ensures that the duration of each clock tick is uniform and supports the use of the TSC as a wall clock timer even if the processor core changes frequency. "This is the architectural behaviour moving forward for all Intel processors." [16]

Constant TSC operates at the CPU's clock speed from which the 10^{-9} second TSP can be easily derived.

D. Libpcap

The last stage of transmission just before getting to the capture application is the libpcap. Timestamp information received by the libpcap depends on the factors discussed in the previous sub-sections. The Linux-specific part of the libpcap is contained in the pcap-linux.c source file. The library captures the packets with the *pcap_read_packet()* function. Timestamping is handled either by the SIOCGSTAMP IOCTL call or by the TPACKETv2 structure.

IV. Implementation of High Resolution Timestamping

Our goal was to reveal and test all of the kernel functions and features that will be essential parts of our project to modify libpcap to a nanosecond-capable capture library. In this section, related source code snippets are presented in such a way that the beginning of deleted and inserted source code lines are marked with the '-' and '+' signs respectively.

A. Implementation

It is feasible to reach nanosecond TSP resolution purely on software-based tapping:

- The *tstamp* member of *sk_buff* structure is capable of nanosecond resolution
- The Linux kernel function *ktime_get_real()* to query the system clock is in nanosecond resolution
- This function is adequate to fill up nanosecond *tstamp* fields in *sk_buff*
- Accordingly user-space applications (such as libpcap-based ones) could display and process 10^{-9} second resolution timestamps

- The Linux kernel supports TSC as a clock source
- For efficient time synchronization dedicated LAN interfaces and a PTP timing protocol could be used within a low latency wired environment

The input queue handler within the 2.6 kernel puts a 64-bit timestamp onto each frame that is enqueued to the input_packet_queue. The Linux kernel API features the *ktime_get_real()* function that enables us to query nanosecond resolution timestamps from the kernel.

For nanosecond time resolution we assume that the kernel's clock source is a constant TSC that operates at ≥1GHz frequency.

The latest Linux kernels (v2.6.27+) introduce the SIOCGSTAMPNS call that returns with the nanosecond precision timestamp of the last incoming packet.

Through this IOCTL call, we indirectly get to the *sock_get_timestampns()* function inside kernel's sock.c. This function relies on the *ktime_get_real()* for timestamp generation and uses the *ktime_to_timespec()* to convert it to *tv_nsec* format, which is a nanosecond capable time variable within the *timespec* data structure.

```
include/linux/sk_buff.h:

struct sk_buff {
/* These two members must be first. */
struct sk_buff        *next;
struct sk_buff        *prev;
struct sock           *sk;
ktime_t               tstamp;
struct net_device     *dev;
}


include/linux/ktime.h:

ktime_t
union ktime {
      s64     tv64;
#if BITS_PER_LONG != 64 &&
!defined(CONFIG_KTIME_SCALAR)
      struct {
# ifdef __BIG_ENDIAN
      s32     sec, nsec;
# else
      s32     nsec, sec;
# endif
      } tv;
#endif
};

typedef union ktime ktime_t;
```

Our first modification is replacing the SIOCGSTAMP IOCTL call with the more recent SIOCGSTAMPNS one:

```
pcap-linux.c, in function pcap_read_packet():

-       if (ioctl(handle->fd, SIOCGSTAMP,
&pcap_header.ts) == -1) {
-             snprintf(handle->errbuf,
PCAP_ERRBUF_SIZE, "SIOCGSTAMP: %s",
pcap_strerror(errno));
+       if (ioctl(handle->fd, SIOCGSTAMPNS,
&pcap_header.ts) == -1) {
+             snprintf(handle->errbuf,
PCAP_ERRBUF_SIZE, "SIOCGSTAMPNS:
%s", pcap_strerror(errno));
              return -1;
       }
```

Libpcap alternatively uses the *tpacket_hdr* structure to query packet description header information.

```
struct tpacket_hdr
{
      unsigned long   tp_status;
      unsigned int    tp_len;
      unsigned int    tp_snaplen;
      unsigned short  tp_mac;
      unsigned short  tp_net;
      unsigned int    tp_sec;
      unsigned int    tp_usec;
};
```

We would like to emphasize the limitation of this structure: content of the *tp_usec* element is always a microsecond precision sub-second time value (TSP). Latest linux kernels (2.6.27+) now feature the enhanced *tpacket_v2* structure:

```
struct tpacket2_hdr
{
      __u32         tp_status;
      __u32         tp_len;
      __u32         tp_snaplen;
      __u16         tp_mac;
      __u16         tp_net;
      __u32         tp_sec;
      __u32         tp_nsec;
      __u16         tp_vlan_tci;
};
```

The novel *tpacket_v2* is able to store nanosecond precision TSP as well as some VLAN information. We managed to maintain and adapt the benefits of *tpacket_v2* structure within the packet capturing process. Our next modification is to retain the nanosecond information provided by the tpacket_v2 structure:

pcap-linux.c, in function *pcap_read_linux_mmap*():

```
                case TPACKET_V2:
                        tp_len    = h.h2->tp_len;
                        tp_mac    = h.h2->tp_mac;
                        tp_snaplen = h.h2->tp_snaplen;
                        tp_sec    = h.h2->tp_sec;
  -                     tp_usec   = h.h2->tp_nsec / 1000;
  +                     tp_usec   = h.h2->tp_nsec;
                        break;
```

Using the default capture buffer size of libpcap (which is 2 Mbytes) disk I/O performance could lead to a serious number of packet drops at a high transmission rate. Unfortunately libpcap does not feature any option for adjusting capture buffer. In order to prepare libpcap for high speed packet procession, its capture buffer had to be increased. We made a series of measurements at 1 Gbps transmission rate so as to determine the optimal size of this buffer for capturing without packet loss. These measurements resulted in a capture buffer of 128 Mbytes, which is an empirical value. The last modification made to libpcap is to increase the default buffer size:

pcap-linux.c, in function *activate_mmap*():

```
        if (handle->opt.buffer_size == 0) {
  -             /* by default request 2M for the ring buffer
*/
  -             handle->opt.buffer_size = 2*1024*1024;
  +             /* request 128M for the ring buffer */
  +             handle->opt.buffer_size =
128*1024*1024;
        }
```

With this modification we enhanced the well-known libpcap (v1.1.x) library so as to be able to put nanosecond precision timestamps onto captured packets without packet loss. This feature relies on some novel features of the latest linux kernels that we effectively integrated into the libpcap library via our modifications. We made libpcap ready to operate with nanosecond precision timestamps; however its effective TST greatly depends on the performance of the underlying hardware.

At this point it is important to note that the nanosecond resolution is largely theoretical. On a dedicated server-class machine with 2 x Intel Xeon dual-core 3GHz (Woodcrest 5160) CPUs and 8GB of RAM, we managed to get a TST of approx. 75 ns. Hence, we would like to emphasize that effective TST and its variance greatly depends on the hardware performance of the host computer, its current system load, various critical kernel parameters and the time data conversion overhead. Even so, in our result TST is close to the time precision requirement of packet capturing on 10 Gbps Ethernet since the inter-arrival time of minimum-sized consecutive frames is about 61ns.

System dependency and variance of TST are rooted in the software-based nature of the solution. Accordingly, an extensive series of comparative tests against hardware-based solutions is required for its validation (see Section IV for details).

B. *Application*

With the nanosecond capable libpcap, a wide range of network data traces can be captured and stored for subsequent analysis. Accordingly, we have made further developments to make the commonly used tcpdump and Wireshark capable of easily processing, displaying and storing these high precision timestamps in the quasi-standard pcap file format (Figs. 3, 4).

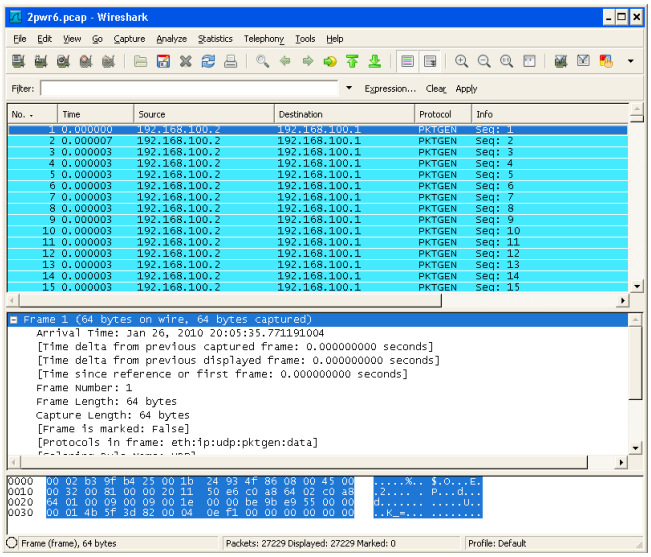


Figure 3. Output screen for microsecond timestamp resolution with the standard libpcap and Wireshark

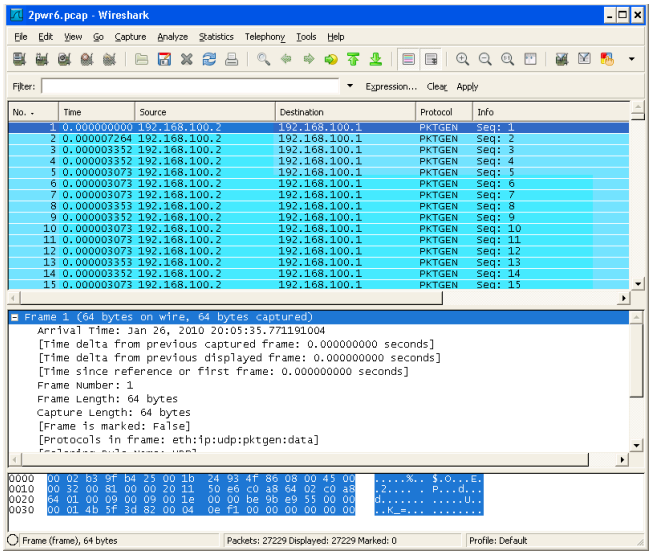


Figure 4. Output screen for nanosecond timestamp resolution with the enhanced libpcap and Wireshark

While tcpdump is not, Wireshark 1.2.x is indeed ready to process pcap trace files that feature nanosecond precision packet timestamps [17]. We had to apply a specific magic number (0xa1b23c4d) that registered for the nanosecond pcap format. Using this number, Wireshark could be made capable of identifying and capturing pcap files with alternative attributes and structures.

In the pcapio.c source file, in function libpcap_write_file_header():

```
-file_hdr.magic = PCAP_MAGIC;
+file_hdr.magic = PCAP_NSEC_MAGIC;
```

The magic number of 0xa1b23c4d stands for a subversion of pcap that includes nanosecond precision timestamps for each packet.

The default time precision of the Wireshark GUI must be set to a nanosecond:

In the gtk/recent.c source file:

```
- recent.gui_time_precision = TS_PREC_AUTO;
+ recent.gui_time_precision
=TS_PREC_FIXED_NSEC;
```

For comparison, two screenshots present the timestamp resolution enhancement. Microsecond scale TSP is the default time resolution for libpcap and Wireshark (Fig. 3), while our modified version is capable of capturing nanoscale TSP as displayed in the “Time” column (Fig. 4).

V. PRECISION EVALUATION OF THE SOFTWARE TIMESTAMPING

A. *The timestamp generation process within the kernel*

Timestamp put onto each packet must be adequately accurate. Hence, it is essential to investigate the process of its generation and to minimize CPU consumption of its sub-processes. Received packets are enqueued by the kernel in the CPU’s incoming packet queue. Packet enqueuing is performed within interrupt context while software timestamps are generated by the getnstimeofday() and ns_to_timespec() functions (Fig. 5).

For representing time domain relation of the successive packets in 10^{-9} second resolution, TST also has to be kept in this time domain. The minimum of this overhead can be predicted by measuring the execution times for getnstimeofday() and ns_to_timespec(). For this measurement we applied a clock source with the lowest and the most uniform overhead, called TSC. The CPU instruction used to read the TSC register value is RDTSC [18] since its execution time takes constant or shows a very low variation on most systems. Time consumption for the aforementioned functions are measured by inserting TSC checkpoints into the kernel code. The read TSC values are corrected with its overhead on the current system as well as taking the system CPU clock frequency in account.

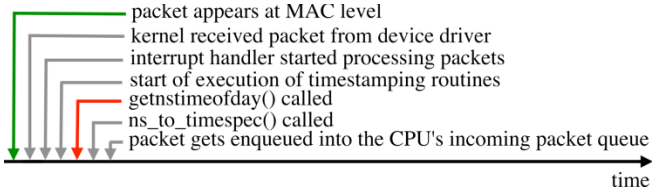


Figure 5. The sub-processes of software-based timestamp generation

The minimum overhead of TST for common CPU frequencies can be derived from mean results of a measurement series made on various hardware architectures (Fig. 6). For a CPU of 3 GHz only some 10 ns is the estimated minimum value of TST. Real execution times show some variation since several processes must share the hardware resources, in this case the CPU itself. In the extreme timestamp generation instructions are preceded or interrupted by the execution of other processes' instructions, which the kernel scheduler decides upon.

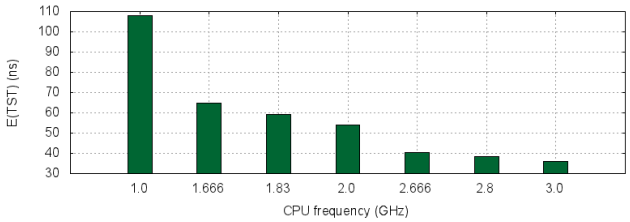


Figure 6. Timestamping overhead on different CPU speed

B. *Timestamping performance of the common kernel clock sources*

In order to compare timestamping capabilities of the mostly available clock sources, we set up a Gigabit Ethernet test network. A dedicated FPGA (Field-programmable Gate Array) packet generator had been set up and a continuous sequence of 72-Byte packets has been generated and captured. Since the PCI-based Netfpga-1G board [19] was applied as GbE NIC the inter-frame gap was adjusted to 1472 bytes due to the data transfer limitation of the PCI bus and the Netfpga device driver. Also note that its driver does not support NAPI, more packets per interrupt mode or other performance enhancing technologies. The inter-arrival times and packet losses were recorded for every clock source. Beside the software timestamp derived from the kernel clock source an additional 8 ns resolution hardware timestamp was inserted by the NetFPGA (Fig. 7).

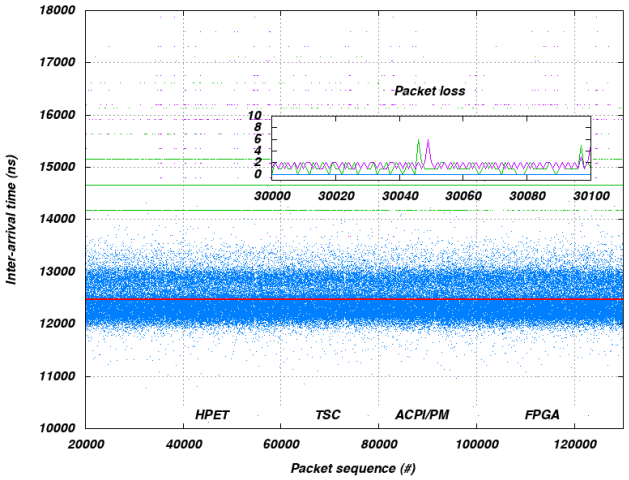


Figure 7. Precision evaluation of the timestamping

On Figure 7, one dot represents a successfully received and timestamped packet where the timestamp value is relative to the arrival of the previous packet.

For this dual timestamping operation mode, we modified the Linux kernel as well as the libpcap library to be able to store both timestamps side by side. Since our FPGA-based packet generator injects a 32-bit serial number into each packet, losses were easy to detect.

In this case beside that the inter-arrival times show large variation using the ACPI-PM and HPET clock sources, serious packet drops were present due to the high overhead of accessing these clock sources.. ACPI-PM based timestamps show the highest overhead. The three nearly solid lines of HPET show that TST variance is lower than that of ACPI-PM. Since the packet sending rate was constant a higher inter-arrival time value indicates a higher execution overhead.

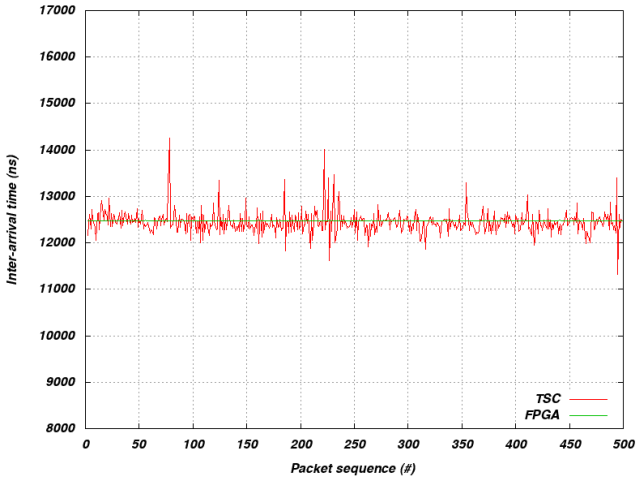


Figure 8. TST performance comparison (TSC and hardware timestamping)

Based on the hardware timestamps we can assume that the time values derived from TSC show a more realistic representation of the inter-arrival times. Moreover, it features the lowest TST overhead among the supported kernel clock sources.

Accordingly, it is reasonable to compare the variance of inter-arrival times gained from the TSC and the NetFPGA (Fig. 8). It is important to note that hardware timestamps show inter-arrival times of the packets in the MAC layer in contrast to the software based timestamps that represent the time moment of enqueuing the received packet.

Nevertheless, there is an obvious difference between the hardware and software timestamps in absolute time since their generation occurs at different points of the data path. They are not related very closely but the comparison shows that the relative inter-arrival times derived from TSC can be fairly close to the hardware-based values. However, even with TSC the TST variation of software timestamps has a significant extent since the execution of the generator functions is triggered by the kernel's scheduler subsystem (Fig. 8). Software timestamps are generated by kernel space functions, thus it is easy to see that kernels with pre-emption enabled are not eligible for high precision timestamping and high performance packet capturing.

C. *Timestamp generation using TSC*

To generate a timestamp based on TSC its value has to be converted, since the register value read by RDTSC is not represented in natural time but in the CPU frequency. Linux kernel has the cyclecounter/timecounter/timecompare framework. It makes possible to use independent cycle counter running on an arbitrary frequency to convert it to natural time. The cyclecounter structure has to be initialized by storing the counter's current value (in this case the register value of TSC) and letting it know its ticking frequency. Since there is no floating point support in the kernel, this frequency conversion is described by a mask, a shift and a mult member. Mask describes the size of the counter (64 bit for TSC). Shift is fixed at 22 (based on an algorithm in arch/mips/kernel/time.c from the Linux kernel) and clocksource_khz2mult() helps to convert CPU frequency into a multiplier. Then the timecounter has to be fixed to a base time (using ktime_get_real() and another clock source such as HPET) and to be stored the counter's current cycle counter value.

Timecounter_cyc2time() function is applied to convert counter value to natural time and timecompare_update() to update offset since last conversion, furthermore to handle possible counter turnaround. Downside is that the usage of these function calls adds significant processing overhead.

Nevertheless, TSC is actually the most adequate clock among the commonly available Linux kernel clock sources. On high performance and carefully tuned systems, its precision is sufficient for generating timestamps in the nanosecond time domain for certain traffic patterns, however for intensive traffic (high packet rate) hardware-based solution has to be applied.

As a future project, a post-processing of TSC data could be implemented to get the potential benefits of offloading register data conversion to time of day format.

VI. CONCLUSION

Version 1.1.1 of libpcap provides a 10^{-6} second native resolution, however pcap format supports a larger 2 x 32-bit timestamp value for each stored packet. On Gigabit Ethernet and faster networks, a timestamp resolution that works in the microsecond domain may not enable the precise reproduction of the time-domain relation between consecutive frames. Therefore overall analysis of the data transmission could lead to a false result.

For packet capturing with libpcap, it is assumed that timestamping is performed when a frame is enqueued to the kernel's input packet queue. Accordingly libpcap must retrieve timestamps from the kernel.

In this paper, we showed our alternative libpcap-based network monitoring solution for Linux systems, which features nanosecond resolution timestamping. Our primary goal was to test and evaluate all of the clock sources and kernel functions and features that are essential parts of our project to turn libpcap into a nanosecond-capable capture library. With the presented modifications and additions to the original codes, we managed to maintain and adapt the benefits of *tpacket_v2* structure within the entire packet capturing process, which resulted in our enhanced libpcap solution. In Section V, the precision of the applied software-based timestamping was analyzed and evaluated. We showed that the variation of the TST derived from two factors: the retrieval overhead of the applied clock source and the kernel's scheduler that commands the execution of running processes.

ACKNOWLEDGMENT

The work is supported by the TÁMOP 4.2.1./B-09/1/KONV-2010-0007 project. The project is implemented through the New Hungary Development Plan, co-financed by the European Social Fund and the European Regional Development Fund.

REFERENCES

[1] Peter Orosz and Tamas Skopko, "Software-based Packet Capturing with High Precision Timestamping for Linux," August 22-27, 2010, 5th International Conference on Systems and Networks Communications, Nice, France

[2] Peter Orosz and Tamas Skopko, Timestamp-resolution problem of traffic capturing on high speed networks, January 28-30, 2010, ICAI international conference, Eger, Hungary

[3] Gianluca Iannaccone, Christophe Diot, Ian Graham, and Nick McKeown, "Monitoring very high speed links," Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, November 01-02, 2001, San Francisco, California, USA

[4] Libpcap, a common open source packet capture library for Unix/Linux systems. [Online]. Available: http://www.tcpdump.org/, 29/07/2011

[5] IETF RFC2679, A one-way delay metric for IPPM. [Online]. Available: http://www.ietf.org/rfc/rfc2679.txt, 29/07/2011

[6] IETF 3393, IP Packet Delay Variation Metric for IPPM. [Online]. Available: http://www.ietf.org/rfc/rfc3393.txt, 29/07/2011

[7] C. J. Bovy, H. T. Mertodimedjo, G. Hooghiemstra, H. Uijterwal, and P. Van Mieghem, "Analysis of End-to-end Delay Measurements in Internet," submitted to PAM 2002

[8] Jörg Micheel, Stephen Donnelly, and Ian Graham, "Precision timestamping of network packets," Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, November 01-02, 2001, San Francisco, California, USA

[9] The DAG project. [Online]. Available: http://www.endace.com, 29/07/2011

[10] Attila Pásztor and Darryl Veitch, "PC based precision timing without GPS," Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, June 15-19, 2002, Marina Del Rey, California, USA

[11] Cace TurboCap network interface card. [Online]. Available: http://www.cacetech.com/products/turbocap.html, 29/07/2011

[12] Nethawk iPro traffic analysis appliance. [Online]. Available: https://www.nethawk.fi/products/nethawk_ipsolutions/ipro/, 29/07/2011

[13] Cascade Shark Appliance. [Online]. Available: http://www.riverbed.com/us/products/cascade/cascade_shark_appliance.php, 29/07/2011

[14] Linux NAPI device driver packet processing framework. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/napi, 29/07/2011

[15] Christian Benvenuti, Understanding Linux Network Internals, O'Reilly, 2006

[16] TSC, Intel 64 and IA-32 Architectures Software Developer's Manual. [Online]. Available: http://developer.intel.com/Assets/PDF/manual/253667.pdf, 29/07/2011

[17] Wireshark Network Protocol Analyser. [Online]. Available: http://www.wireshark.org/, 29/07/2011

[18] Performance monitoring with the RDTSC instruction, [Online]. Available: http://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf, 29/07/2011

[19] NetFPGA project. [Online]. Available: http://www.netfpga.org/, 13/01/2011

Building CPU Stubs to Optimize CPU Bound Systems: An Application of Dynamic Performance Stubs

Peter Trapp, Markus Meyer, Christian Facchi
Institute of Applied Research
University of Applied Sciences
Ingolstadt, Germany
{*trapp, meyerma, facchi*}*@haw-ingolstadt.de*

Helge Janicke, François Siewe
Software Technology Research Laboratory
De Montfort University
Leicester, United Kingdom
{*heljanic, fsiewe*}*@dmu.ac.uk*

Abstract—Dynamic performance stubs **provide a framework for the simulation of the performance behavior of software modules and functions. Hence, they can be used as an extension to software performance engineering methodologies. The methodology of *dynamic performance stubs* can be used for a gain oriented performance improvement. It is also possible to identify "hidden" bottlenecks and to prioritize optimization possibilities. Nowadays, the processing power of CPUs is mainly increased by adding more cores to the architecture. To have benefits from this, new software is mostly designed for parallel processing, especially, in large software projects. As software performance optimizations can be difficult in these environments, new methodologies have to be defined. This paper evaluates a possibility to simulate the functional behavior of software algorithms by the use of the *simulated software functionality*. These can be used by the *dynamic performance stub* framework, e.g., to build a *CPU stub*, to replace the algorithm. Thus, it describes a methodology as well as an implementation and evaluates both in an industrial case study. Moreover, it presents an extension to the *CPU stubs* by applying these stubs to simulate multi-threaded applications. The extension is evaluated by a case study as well. We show show that the functionality of software algorithms can be replaced by *software simulation functions*. This stubbing approach can be used to create *dynamic performance stubs*, such as *CPU stubs*. Additionally, we show that the concept of *CPU stubs* can be applied to multi-threaded applications.**

Keywords-software performance optimization; CPU bound systems; simulated software functionality; stubs; multi-core; multi-threaded

I. INTRODUCTION

CPU stubs [1], which are a subset of *dynamic performance stubs* (DPS) have been introduced in [2]. They can be used for "hidden bottleneck" detection, and a cost-benefit analysis can be performed by demonstrating the level of optimization potential. This leads to more gain-oriented performance optimizations. These benefits are not addressed in other software performance engineering methods (see [3]–[7]). *DPS* extend these methods by simulating various levels of system load. Hence, the problem of a missing cost-benefit analysis can be bypassed. The *DPS* can be used within the software development cycle of large software systems, e.g., in telecommunication systems.

Many system architectures achieve higher throughput by using multiple cores. Hence, the application has to be able to do parallel processing to fully utilize the available capacity of the system. As multi-threaded and parallel processes are difficult to optimize, new methodologies in the area of software performance engineering have to be defined. The methodology of *DPS* can be used to optimize CPU bound processes by using *CPU stubs*.

A. Dynamic Performance Stubs

The idea behind *DPS* is a combination of performance improvements [3]–[7] in already existing modules or functions and the stubbing mechanism from software testing [8], [9]. The performance behavior of the component under study (CUS) will be determined and replaced by a *DPS*. This stub can be used to simulate different performance behaviors, which can be parameterized. Typically, the CUS is the part of the software under test (SUT) that has been identified as a potential performance bottleneck. The optimization expert can use *DPS* to analyze the performance of the SUT. This procedure relates to stubbing a single ("local") software unit, and a local stub has to be built. The *DPS* can also be used to change the behavior of the complete system. A software module has to be created, which interacts "globally" in the sense of influencing the whole system instead of a single software component. This stub will be called a "global stub".

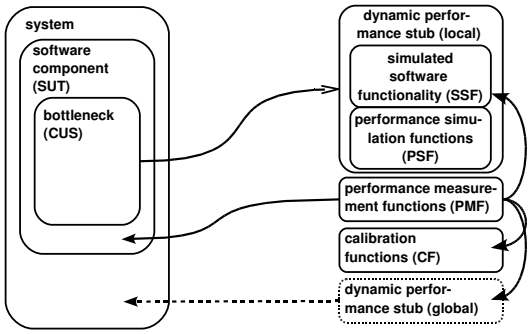


Figure 1. Interactions of "Dynamic Performance Stubs"

Figure 1 sketches the design and the interaction between a real system on the left and the *DPS* on the right side. The unfilled arrowhead indicates a replacement. Filled arrowheads describe the extension of a unit by this feature and the dashed block provides an additional functionality to the *DPS* and will not really replace a software unit. In the context of *DPS*, the system under test (SUT) is a software module or function, which includes a software performance bottleneck.

The framework of the *DPS* consists of the following parts, which is presented in Figure 1:

- Simulated Software Functionality
  The *simulated software functionality* (SSF) is used to simulate the functional behavior of a CUS or a software performance bottleneck. This can be achieved by generating valid output values for dedicated input values without executing the original functionality. Another possibility is to simulate different states of objects inside of the CUS. Hence, the application can be executed without the original functionality as it is realized by the *SSF*.
- Performance Simulation Functions
  Performance simulation functions (PSF) provide the ability to simulate the performance behavior of the replaced CUS. They are divided into four categories, as also in [3], [10]:
  - CPU
  - Memory
  - I/O
  - Network

  As an example, the *CPU PSF* can be used to simulate the processing behavior of the application and hence, the CPU utilization of the process.
- Performance Measurement Functions (PMF)
  To provide a basic set of evaluation possibilities the *performance measurement functions* can be used. They are mainly glue/wrapper functions for the measurement functions already provided by the system.
- Calibration Functions (CF)
  In order to provide trustworthy results, the stubs have to be adjusted to a dedicated system. This can be done using the *calibration functions*.

For more detailed information on *DPS*, the reader is referred to [2].

CPU Stubs: CPU stubs, as a special subset of *DPS*, can be used to handle CPU bound processes. These processes are highly utilizing the CPU so that the CPU is the bottleneck. Therefore, a general approach to parameterize the runtime behavior and CPU usage has been achieved and a possible implementation has been presented in [11]. Additionally, an extension to multi-core and parallel processing applications has been done in [1].

Memory Stubs: Memory stubs are separated into *cache memory-* and *main memory stubs*.

- *Cache Memory Stubs* can be used to simulate the data cache access behavior of software modules or functions to improve suspected memory bottlenecks. The algorithm, a validation as well as an evaluation by means of a proof of concept for cache memory stubs have been published in [12].
- *Main memory stubs* simulate the stack and heap behavior of software modules or functions. They are an extension of the *DPS* framework to simulate the main memory behavior to achieve a cost-benefit oriented optimization. They are defined in [13].

B. Content of the Paper

In Section II, the problem of simulating the results of software algorithms is addressed, by describing a novel approach to the simulation of software functionality using stubs. Starting with evaluating requirements concerning the simulation functions, a methodology is presented. Additionally, the concept of a possible implementation is depicted. Both, the methodology as well as the implementation are evaluated in an industrial case study to optimize a network component of a long term evolution (LTE) telecommunication system. This section extents and completes the approach of *CPU stubs* as published in [1].

In Section III, the paper shows an introduction to the *CPU stubs* as presented in [1]. Here, the *CPU PSF* are presented and a methodology for using *CPU stubs* to optimize CPU bound systems in multi-core or parallel processing environments is given. The introduction as well as the methodology depicts the concept of *CPU stubs* and have been published in [1].

In Section IV, the *CPU stubs* are extended to simulate the performance behavior of multi-threaded applications. This extension is realized by defining objectives and by presenting a novel approach for *multi-threaded CPU PSF*. Additionally, the approach is validated by a case study.

Finally, related work for the *DPS* and for the *SSF* is provided in Section V.

II. Simulated Software Functionality

The *SSF* allows the replacement of an existing software module or function by a stub, in order to do software performance improvement studies. In the context of this paper, the *SSF* is used to replace a software bottleneck (CUS) with a *DPS* being able to simulate different performance scenarios to estimate the benefit of potential performance optimizations.

In the following, the requirements to the *SSF* and the methodology is presented. A implementation of the *SSF* is given. This section is concluded by an industrial case study.

A. Requirements

In order to be able to replace a bottleneck with a *DPS*, it is necessary to recreate the functionality of the software module or function. Hence, the following requirements can be defined and subdivided into: *requirements on the system*, which have to be satisfied by the SUT and *requirements on the SSF*, i.e., how the *SSF* has to behave:

1) **Basic Requirements on the system:**
   a) *Deterministic CUS Behavior*
      The software module or function has to have a deterministic functional behavior. Any execution of the function with the same input values returns the same output values, e.g., deterministic output values depending on the input values. The performance behavior of the function has to be deterministic, too.
   b) *Reproducible test execution*
      The used test environment and test scenarios have to deliver reproducible results. This is a common requirement to any test environment.
   c) *Automated test case execution*
      It is preferable if the test cases can be executed automatically. This property significantly reduces the effort for repeated executions of the tests. Additionally, reproducible test scenarios can also be used for performance measurements.
2) **Requirements on the SSF:**
   a) *Automatic generation of the serialization specification*
      The *serialization specification* is a description, which provides the procedural method to serialize the data types used in the *SSF* library. This *serialization specification* shall be generated automatically, because it removes additional effort for the user of the *SSF* and decreases the amount of possible errors, e.g., writing a wrong *serialization specification*. Hence, a serialization functionality shall be provided as well as an almost automatically *serialization specification* shall be generated. These can be used to automatically store the C++ objects.
   b) *Record and restore C++ data structures*
      It has to be possible to record and restore C++ data structures. Especially, it has to be possible to record and restore classes including non-public members, structures and lists. Moreover, the *SSF* has to be able to work with "NULL"-pointers, e.g., the "NULL"-pointers shall be stored in the trace file and restored during the stubs execution.
   c) *Simulate the functional behavior*
      The *SSF* shall be able to restore the functionality of the CUS. Moreover, it has to be able to restore all recorded C++ data structures into the memory of the SUT. Additionally, it shall be able to create objects if they are not available in the system.
   d) *Simulate the functional behavior with appropriate performance*
      The *SSF* has to be able to restore the functionality in negligible time, which is at least faster than the execution of the original software function. This is necessary to reduce the runtime overhead during the execution. Hence, the performance parameters can be easily adjusted using the *PSF*. This requirement mainly applies if the *SSF* is used in the context of *DPS* performance measurements. In this case, the requirement has to be fulfilled.

Especially, the requirements to the system (Requirements 1a and 1b) as well as the Requirements 2b and 2c are important. Not reaching them renders the *SSF* unusable. Requirement 2d is mainly important in the context of *DPS* as this requirement enables the performance adjustments, which are necessary for the *DPS*' approaches. This requirement may not be that important if the *SSF* approach is used in different scenarios. The Requirement 2a can only be fulfilled partly as stated in Section II-C1. The Requirement 1c is only suggested as it can significantly remove the overhead for applying the *DPS* framework in the performance evaluation study.

Moreover, there are some requirements to the C++ compiler [14]. The compiler shall be deterministic, e.g., the compiler has to produce an identical memory layout of two isomorphic classes. Whereas, this can not be strictly guaranteed, it is very unlikely that a non-deterministic C++ compiler is standard-compliant [14]. The "g++" of the gnu compiler collection (GCC) [15], which is used for the evaluation in this paper, fulfills the requirements.

In the next subsection, the methodology for using the *SSF* is presented.

B. Methodology

The *DPS* methodology, identifies potential performance bottlenecks that are replaced by stubs to facilitate gain-oriented performance improvements. The functionality of the potential bottleneck (CUS) is recorded using the libSSF (see Section II-C). The system's behavior with respect to the CUS is analyzed by replaying the functionality using the *SSF* with varying performance measures.

An overview is presented in Figure 2. It describes the overall process to replace the bottleneck by a *DPS*. There are three different parts, which have to be done. First, a header file is generated, which includes the *serialization specification*. Second, the functional behavior of the bottleneck has to be recorded and stored in a trace file. Finally, the trace file can be used to simulate the functional behavior of the bottleneck. The steps as provided in this methodology are presented as circles including the step number.

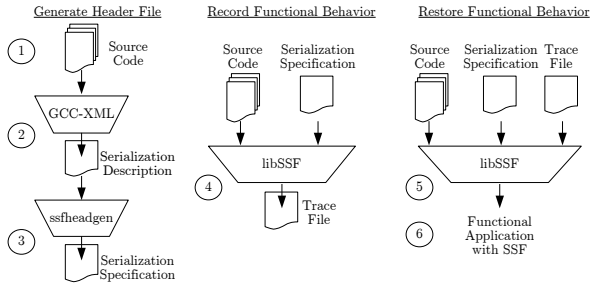


Figure 2. Overview of the *Simulated Software Functionality*

The methodology for the *SSF* approach consists of the following steps:

1) **Identify Serialization Objects:**
The *SSF* is able to store and restore different states of the traced objects. Thus, it can be used to simulate the results of several algorithms. In some cases, it is necessary to use parts of the original functionality to improve the simulation results. Here, the content of the object may be stored and restored before and after executing the original software functions.
2) **Create Serialization Description:**
In this step, the serialization description of the identified objects has to be created. This is simply done using the "GCC-XML" tool set [16].
3) **Create Serialization Specification:**
The *serialization specification* is created. It contains a description to serialize and de-serialize the objects which will be stubbed. The serialization objects (see Step 1) and their description (see Step 2) are processed by the *ssfheadgen* tool, which is a part of the *libSSF* library, to generate a C++ header file that contains the *serialization specification*. This specification has been created automatically for many basic data types, as explained in Section II-C1, but, can also be easily extended by the developer to support object serialization. This header file will be included into the CUS in the next step.
4) **Record the state of the objects:**
In this step, the CUS is adjusted to store the results of the algorithm using the *libSSF*. Furthermore, the test cases that utilize the functionality, which will be stubbed (see Step 1), have to be executed and the state of the results have to be recorded into a trace file.
5) **Create Functional Software Behavior:**
The original functionality, which is a part of the CUS, is replaced by the *SSF*. This is detailed in Section II-C.
6) **Test the instrumented CUS:**
As the stub has been created in Step 5, the functionality of the stub has to be validated. The instrumented CUS is validated against the previously recorded behavior of the CUS that contained the original functionality. As there is the potential of introducing functional errors during the instrumentation, the instrumented CUS is re-validated against the previously recorded execution trace. If the validation passes, the stubs can be used to do the performance study with the *DPS* framework.

This section has shown how a stub can be created using the *SSF*. The following section presents a possible implementation called *libSSF*.

C. Implementation

The implementation of the *SSF* is done in a library called *libSSF*. The library can be included into any C++ source code and allows for the storage of the content of C++ data structures into a binary trace file. Moreover, the *libSSF* can also be used to read from the trace file to reconstruct the C++ data structures. Thus, the functional behavior of the CUS is also recreated. For this reason, the source code of the application will be parsed using the "GCC-XML" tool set [16], which generates an XML description of a C++ program from GCC's internal representation. Based upon this serialization description, *libSSF* generates an internal representation of the objects that will be stubbed. The following functionalities are provided by the *libSSF*. These are the general steps (see Figure 2):

1) Generate Header File
This file includes the *serialization specification*.
2) Record Functional Behavior
This functionality will be used to store the results of the software functionality of the CUS.
3) Restore Functional Behavior
This functionality will be used to simulate the software functionality of the CUS.

Following, the listed items are described in more detail.

1) Generate Header File (Serialization Specification): A tool provided by the *libSSF*, called "ssfheadgen", parses the XML description of the "GCC-XML" tool. It extracts the type information and generates a C++-header file, which includes the internal representation of the objects that will be replaced by the *SSF*.

This header file can be included into the C++ source code of the CUS and contains the *serialization specification* of the objects. Beside for the basic data structures, e.g., basic data types or fixed size arrays, which have to be serialized and de-serialized, the developer has to adjust the header file to his needs. This has to be done manually as it is not always possible to determine the size of data associated with a pointer value.

Whenever possible, the header file already contains comments and suggestions to assist the developer in serializing the object, e.g., for pointers or arrays[1]. Moreover, the header file includes the original names, as used in the CUS source

[1]In these cases a "stop criterion" has to be specified by the developer.

```
template <> void Stubfactory::serializeType(class array_class *ssfSaveObj) {
  void *prt = ssfSaveObj;
  struct ssfSave_array_class *ssfObject = (ssfSave_array_class *) ptr;
  this->serializeArray(ssfObject->ac_i, numberOfElements);
}
```

Listing 1. Example: Serialization of a Fixed Sized Integer Array inside of a C++ class

code, of the replaced objects, so that the developer can reuse these names for convenience.

Listing 1 shows an example of the serialization of an array of integers ("ac_i"). The array is a private member of a class ("struct array_class"). This snippet is used to deserialize as well as to serialize the data values of the object.

The "serializeType"-function from Line 1 will be called indirectly inside of the CUS. The provided parameter specifies a C++ class which shall be serialized and stored. In Line 2, a type cast of the object pointer to a void pointer is done. This is necessary for being able to furthermore cast the pointer to a "struct", which reflects the C++ class. In this case, the private or protected members of the provided class ("ssfSaveObj") can be accessed and, hence, stored. This is done in Line 4, where, the private member, which is a fixed size array in this example, will be copied into the trace file.

This example shows that private and protected members can be serialized. Other serialization functions are available to support the developer.

2) Record Functional Behavior (Binary Format): The C++-header file, which has been generated by "ssfheadgen", is included into the CUS. And, the software tests, which have been done to identify the serialization objects, have to be repeated. Now, the information of the objects are stored in a trace file. This recording of the data structures is done using the function "saveStateOfParam", which is included into the CUS. This function is provided by the *libSSF*. The declaration of the function can be seen in Listing 2.

```
template <class TYPE> void saveStateOfParam(
    const char *name, const char *type, TYPE
    *dataVar);
```

Listing 2. Stores a Data Structure e.g. class

The following three parameters have to be passed to the "saveStateOfParam"-function call:

1) "const char *name": This is the name used to store the object in the trace file, e.g., "conn".
2) "const char *type": This refers to the type and name of the object to be stored, e.g., "class Connection".
3) "TYPE *dataVar": This is a pointer to the data which will be stored, e.g., the value of the conn variable.

The three given examples in the list above can be interpreted as: Store the value of the *conn* variable, which has an object type *"class Connection"* into the trace file using the name *"conn"*. The function uses the parameters and stores the data structure as well as additional information into a binary trace file. The structure of a trace file entry is given and described below:

- Test Run
  This is an internal reference counter starting from zero. The "test run" number can be used to summarize different stored variables into a combined run, e.g., if the value of a variable has to be stored before and after some modification within a single execution of the function.
- Size of Object Name (Byte)
  The size of the object name is given in bytes including a "NULL"-termination character.
- Name of the Object
  The name of the object which has been stored. It is usually the same name as the name of the object within the original source code and can be used for referencing the stored data.
- Size of the Object Type Name (Byte)
  The size of the object type name is given in bytes including a "NULL"-termination character.
- Name of the Object Type
  The name of the object type, e.g., "class Connection", which means the data entry refers to a C++ class named "Connection". Here, object type refers to any C++ data structure and can also be a basic data type such as an integer.
- Additional Information
  The "additional information" (addInfo), which is a field of 8 bits, is used to determine whether a fully initialized object has been stored or if a "NULL"-pointer has been passed. This information is stored in the first bit flag. The remaining bits of this field are unused. Hence, the first bit of "additional information" field is set to "0" if an initialized data structure has been stored. In this case, the following two additional data fields are stored in the trace file for this test run:
  - Size of the Stored Data (Byte)
    This is the size of the serialized data in bytes.
  - Stored Data
    These are the values of the data structure. The data have been serialized in advance and are successively ordered in the trace file.

The information is stored in a binary format for performance

reasons. A decoded as well as semicolon separated example trace entry is given in Listing 3.

In this case, an object "conn" of the "class Connection" type has been stored. The values of the serialized private members are: "1", "2", "1" and "302845744".

```
1;5;conn;17;class Connection;0;
    14;1;2;1;302845744;
```

Listing 3. Example: Decoded and semicolon separated trace file entry

The *libSSF* provides an option to generate a trace file decoder for a dedicated trace file. This has been implemented to provide human-readable traces to the developer.

3) Restore Functional Behavior (Deserialization): The recorded values have to be recreated into the memory of the used C++ data structure. Hence, it is necessary to overwrite the values already stored in the memory of the object. To do this, three different cases have to be considered:

1) The object as well as trace data are available.
   In this case, the existing attributes of the object have to be overwritten as the object is already available in the system.
2) The object does not exist but data are available.
   The object has to be created and initialized using the values of the trace file. Moreover, the pointer to the object has to be returned to the system. This is possible as the delivered memory pointer has to be "NULL". In this case, no memory is associated with the original object. As there is no reference available, dangling pointers can not occur.
3) The object does not exist and no data are available.
   This case happens if an initialized object is not necessary, e.g., if the return value of a search algorithm does not find the item. I.e., the CUS returns a "NULL" value. In this case, the object pointer passed to the "loadStateOfParam"-function of the *libSSF* (see Listing 4) has to be "NULL". Here, no memory will be allocated.

A fourth case is that an initialized object has been passed to the *libSSF* but no data are associated within this test run. In this case, a "NULL" pointer would have been returned by the *libSSF*, which will overwrite the original pointer value of the object. This is not allowed as it would cause a memory leak. Additionally, it is not possible to delete the associated object data as this could lead to a double free error. Hence, the developer has to care about this particular case, e.g., deleting the object and setting its pointer value to "NULL" before the "restore" is function called.

The restore functionality of the *libSSF* is implemented by the "loadStateOfParam"-function call. This function will be used to replace the software functionality of the CUS. The declaration of the function is given in Listing 4.

The parameters passed to the *libSSF* are as follows:

```
template <class TYPE> TYPE* loadStateOfParam(
    string dataVar, TYPE *dst);
```

Listing 4. Restore Functionality of the *libSSF*

- "string dataVar": This is the name of the object, which will be deserialized. Here, the same name as specified as the first parameter of Listing 2 is used, e.g., if "conn" is passed to the "loadStateOfParam"-function, the with conn associated data will be returned.
- "TYPE *dst": This is a pointer to an object which will be overwritten by the values read from the trace file. Hence, it is implemented as a template any type of the object can be deserialized and restored by the *libSSF*, e.g., the "class Connection" with an instance name "conn" can be used.

In the case that an object has to be created within the *libSSF*, the pointer value of the newly allocated memory will be returned to the CUS. Here, the original value of the pointer will be overwritten so that the allocated memory can be deleted inside of the original software.

The provided methodology and implementation will be applied to a real world example which is presented in the following section.

D. Case Study

The *DPS* framework has been used to optimize several algorithms of a long term evolution (LTE [17]) telecommunication system.

This section describes the application of the methodology and the newly developed *SSF* within a performance improvement study. The main contribution of this case study is to show that the software functionality of the CUS can be replaced by the *SSF*. This includes the following steps:

- The *serialization specification* is generated.
- The software functionality of the CUS can be recorded.
- The software functionality of the CUS can be replaced by the *SSF*. In this case, the SUT shall be fully functional for this particular test scenarios.

Last but not least, the case study provides performance measurements to validate that the *libSSF* can be used in the context of the *DPS* framework that will be used to evaluate software performance optimization potentials.

1) Test Environment: The measurements have been done in a test environment. The used platform is based on an Intel Xeon CPU, which is an IA-64 architecture and includes OS and memory.

The application has been built using the available build system of the company. This uses the "g++" of "GCC" (Version 3.4.3) for host test environment evaluations. The "-Os" compiler option has been used, which is basically a "-O2" but without optimization flags that increases the code size.

As the presented measurements have been done in a test environment, the results can only be used for validation purposes of the *SSF* but do not reflect the performance of the telecommunication system.

The requirements to the software and test environment, as specified in Section II-A, for using *DPS* are fully met. These are, in particular, a deterministic CUS as well as an automatic and reproducible test case execution environment.

2) Application of the Methodology: The SUT has a "ConnectionContainer" class which stores several connections of the type "class Connection". The function "get(connID)" returns the connection specified by the connection identification ("connID") which is an object of the "Connection" class. Moreover, it returns "NULL" if the connection does not exist in the "ConnectionContainer". The connection class has four private members as can be seen in Listing 5.

```
1  class Connection
2  {
3     ...
4     private:
5        TL3ConnectionId     m_connectionId;
6        u16                 m_streamId;
7        TUeContextId        m_contextId;
8        TAaSysComSicad      m_uecAddress;
9     ...
10 }
```

Listing 5. Excerpt of the Class "Connection"

Step 1: The "get(connID)" function has been identified as bottleneck and, hence, the "Connection" class has been chosen for serialization.

Steps 2 & 3: In the next step, the members of the "Connection" class are serialized using the "GCC-XML" tool set (Step 2). An example serialization output of the *ssfheadgen* (Step 3) is shown in Listing 6.

```
1 name= 'm_connectionId' id= _4096
2     type= _1501
3 -> name= 'TL3ConnectionId' id= _1501
4         type= _1532
5   -> name= 'u32' id= _1532
6           type= _73
7     -> name= 'unsigned int' id= _73
8             type=
```

Listing 6. Example of Serialized Class Member

Here, only the first member "m_connectionId" is presented. The "GCC-XML" combined with the *ssfheadgen* tool identified the "m_connectionId" over four serialization steps as an unsigned integer.

As of Step 3, the *serialization specification* is written into a C++ header file. The first part of the file contains the serialized object, which is presented in Listing 7. As can be seen, the "Connection" class, which has been converted into a data structure, consists of four "private" members, which are integers.

```
struct ssfSave_Connection{                          1
  unsigned int          m_connectionId;             2
  short unsigned int    m_streamId;                 3
  unsigned int          m_contextId;                4
  unsigned int          m_uecAddress;               5
};                                                  6
```

Listing 7. Serialized "Connection" Object

The second part, which is the serialization code, is also included into the file. An extract is shown in Listing 8 for this case study.

```
template <> void Stubfactory::serializeType(        1
    class Connection *ssfSaveObj) {
  void *ptr = ssfSaveObj;                           2
  struct ssfSave_Connection *ssfObject = (          3
      struct ssfSave_Connection *) ptr;
  this->serializeType(&ssfObject->                  4
      m_connectionId);
  this->serializeType(&ssfObject->m_streamId);      5
  this->serializeType(&ssfObject->m_contextId)      6
      ;
  this->serializeType(&ssfObject->m_uecAddress      7
      );
}                                                   8
```

Listing 8. *Serialization Specification* of the "Connection" Object

Here, the "serializeType"-function in Line 1 is able to serialize a object of the "Connection" class. It calls internally several different "serialize"-functions (Lines 4 - 7), which overload the function from Line 1. The "serialize"-functions from Lines 4 - 7 call internally a "serializeAtom"-function, which is able to store and restore basic data types. The values of the variables are stored in their associated members of the data structure (see Listing 7). The "type casts" in Lines 2 and 3 are necessary to access the private members of the "Connection" class (see Section II-C1).

Step 4: Now as the setup has been finished, the measurements have to be repeated to store the software functionality of the CUS. The chosen test case is a functional test case which evaluates different use case scenarios. We only studied a small subset of the test case for the *libSSF*. In our context the test case includes 40 times calling the stubbed functionality ("get(connID)"-function). The test case includes the following use cases: "create new object", "reuse existing object" and "delete and create new object". A decoded excerpt of the recorded trace file is shown in Listing 9. Lines 4-7 of the listing show the recorded values of the private members of the "Connection" class for the second test run.

Steps 5 & 6: In the last two steps, the stub has to be created using the restore functionality. Moreover, the proper

```
1 testrun:0; sizeObjectName:5; objectName:conn;
      sizeObjectType:17; objectType:class
    Connection; addInfo:1;
2 testrun:1; sizeObjectName:5; objectName:conn;
      sizeObjectType:17; objectType:class
    Connection; addInfo:0;
3   sizeOfData:14;
4   m_connectionId:1;
5   m_streamId:2;
6   m_contextId:1;
7   m_uecAddress:302845744;
```

Listing 9. Excerpt of a Decoded Trace File

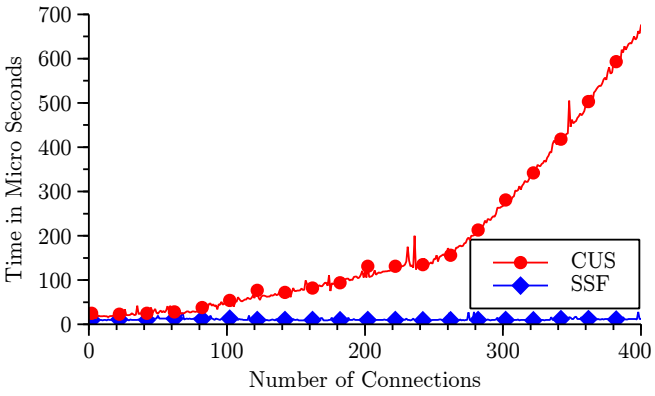


Figure 3. Compare the Times Between Original and Stubbed Software Functionality

working of the stub has to be validated.

The functionality, which will be replaced by the stub, is removed from the CUS and replaced by the restore functionality of the *libSSF* in order to simulate the original functionality. Now, the test case, as used in Step 4, is executed and the results are validated. In the test environment the test case passed. In this case study, the *libSSF* was able to simulate the functional behavior of the CUS.

3) Performance Measurements: The concept of the *SSF* will be used within the *DPS* framework to simulate different performance behaviors of a software bottleneck. Hence, the time to restore the functional behavior of the CUS is critical.

A case study using the *DPS* for optimizing CPU bound processes has been presented in [18]. The focus was to optimize a CPU bottleneck. The case study in this section uses the measurement results of [18], but, interprets the results from a different point of view.

Here, the differences between the execution time of the CUS and the execution time for the *SSF* have been evaluated. In the case study of [18], a previous version of the *libSSF* has been used. However, the results of [18] can still be used for this evaluation as only smaller changes have been done.

The same test environment and software functionality, as described in [18] has been used. A description of the environment and software functionality can also be found in Section II-D1. The chosen test case started with a single database entry and ramped up to searching 400 database entries. Each test case has been done five times and a statistical evaluation was performed by evaluating the minimum, average, maximum and the squared coefficient of variation (SCV, see [19]). The time values have been recorded in cycles using the time stamp counter (TSC, see [20]), which has been read by inline assembler.

In Figure 3, the time behavior for searching an entry in the database (y-axis) depending on the amount of database entries (x-axis) is presented. The lower line (diamond) shows the results for restoring the functional behavior of the search algorithm by using the *libSSF*. The upper line (circle) depicts the original behavior of the CUS.

The new evaluation pointed out the average time for restoring the functional behavior of the "get(connID)" function is 11 µs[2], which is approximately 30800 cycles. The SCV is 0.0135. This factor indicates that it takes approximately always 11 µs without significant variations to simulate the functionality independent of the amount of database entries.

In contrast, the original functionality to identify the connection identification number ("connID") took a minimum of 22µs if only a single entry was in the database and about 675µs if 400 entries have to be searched. The measured results show an exponential increase in time for an increasing database size.

As can be seen, the *SSF* was even in the worst case as twice as fast. Due to this, the identification of the software optimization potential and the improvement of the bottleneck's time behavior were easily realized. Moreover, the methodology of using *CPU stubs* has been applied to the SUT in [18], successfully.

E. Discussion

The *SSF* can be used to store and recreate the functional behavior of software modules or functions of C++ applications. This implemented by the possibility to store and restore the values of C++ data types, e.g., data structures or classes including their private and protected members. Moreover, it is possible to use the *SSF* with applications which use many different programming techniques, such as virtual or abstract classes, inheritance or polymorphism. The functionality is implemented by a library called *libSSF* which can be included into the C++ source of the application.

Advantages: Using the *libSSF* has several advantages for the developer. The main topics are:

- **Store and Restore the Software Functionality**
  The *libSSF* can be used to record and restore the states of traced objects. Hence, it can be used to simulate

[2]The first message has been ignored to avoid side-effects that only occur for the first message ("first message effect").

software functionalities and algorithms, e.g., search-, sorting-, or calculation algorithms. Moreover, existing objects can be modified to the needs of the developer.

- **Mainly Automatic Header Generation**
  The header, which is generated by the *ssfheadgen* tool, can be easily included into the source code of the CUS. Here, only some small modifications have to be done. Moreover, the *ssfheadgen* tool provides suggestions to support the developer by this task. This enables the developer to easily trace and evaluate the content of C++ data types.
- **Reuse Object Names**
  Data types can be stored into and read from the trace file reusing the same names as in the original source code. This significantly reduces the complexity to use the *libSSF* within the CUS.
- **Using Data Types Multiple Times**
  The same variables can be recorded multiple times, even within one single execution of the CUS. Moreover, several different data types can be combined into dedicated runs as well as many different runs can be combined. This provides high flexibility for clustering different runs and data types for a better abstract view on the stubbed components.
- **Readable Values of the Objects**
  The *libSSF* provides the possibility to decode the binary trace files into human readable trace files. Hence, the values of recorded data types can be used for evaluating the outcome of algorithms and, hence, as additional debugging possibility.
- **Only Small Adjustments to the System**
  To simulate the software functionality, only smaller adjustments to the CUS have to be done. For recording, only the library has to be included as well as the necessary function calls have to be added. For restoring, additionally, the original functionality has to be removed, e.g., by uncommenting.

Restrictions: As often, there is a trade off between time and memory usage. If the library is used to restore the functionality of the software it will read the whole trace file into the memory during the initialization. Hence, it uses a lot of memory. Moreover, if a data type has been often recorded, each traced value is preloaded into the memory, e.g., if an integer has been stored ten times the *libSSF* will allocate ten times the size of the integer. This behavior has been chosen as the main focus is on the execution time of the restore functionality. It can be changed with some smaller modifications to the library to only load the data when they are needed. This leads to a longer execution time, of course.

Summary: As can be seen, the methodology of the *SSF* as well as their implementation, realized by the *libSSF*, can be used to record and restore the software functionality. Moreover, the time measurements of the *libSSF* have shown that it can be used in the context of the *DPS* framework.

This is an important contribution to the gain-oriented performance improvement framework *DPS*, as it allows to gauge the system-wide impact of a potential improvement before investing in the actual optimization of the algorithms that underly the functionality that has been simulated by the *DPS*. This informs decision making as to what bottlenecks should be prioritized and to what degree their optimization has a system wide impact.

The requirements on the system, which are 1a, 1b and 1c, as well as to the *SSF* (2b, 2c and 2d) as stated in Section II-A have been fulfilled with the *libSSF*. Finally, the Requirement 2a has been fully fulfilled in this particular case study, but, this can not be applied in a general way as explained in Section II-C1. However, this does not lower the contribution as the *libSSF* supports the possibility to manually adjust the serialization functions. And, hence, provides a broad range for applying the *SSF* to software systems.

III. CPU Stubs

This section introduces *CPU stubs*. They can be used to simulate the CPU performance behavior of a bottleneck. First, the *CPU PSF* are described. Afterwards, a methodology to apply the *CPU stubs* to multi-core and parallel processing is presented.

A. CPU Performance Simulation Functions

The *CPU PSF* consist of two simulation elements: *system influencing-* and *system non-influencing CPU PSF*:

- System influencing
  This functionality simulates the process execution while the process is running. Regarding the process states [21], this can be seen as “Running”. In this case, the CPU has to execute instructions. An implementation can be done by executing a busy loop [11], which executes the NOP instruction.
- System non-influencing
  This functionality simulates the behavior of a process that has been delayed for any reason, e.g., because of a scheduling event or a waiting for I/O. This functionality can be seen as “Blocked” or “Ready” regarding the process states [21] and is simulated by delaying the execution of the process, e.g., using a sleep function call.

By using the *system influencing* and *system non-influencing CPU PSF* any state, regarding the CPU, of a process can be rebuilt.

B. Methodology

In this section, we revisit the methodology of using *CPU stubs* for simulating the execution states running and blocked/ready (see [21]) of individual processes that was presented in [1], [11] and show how these can be used in combination with the *SSF* that was presented in Section II.

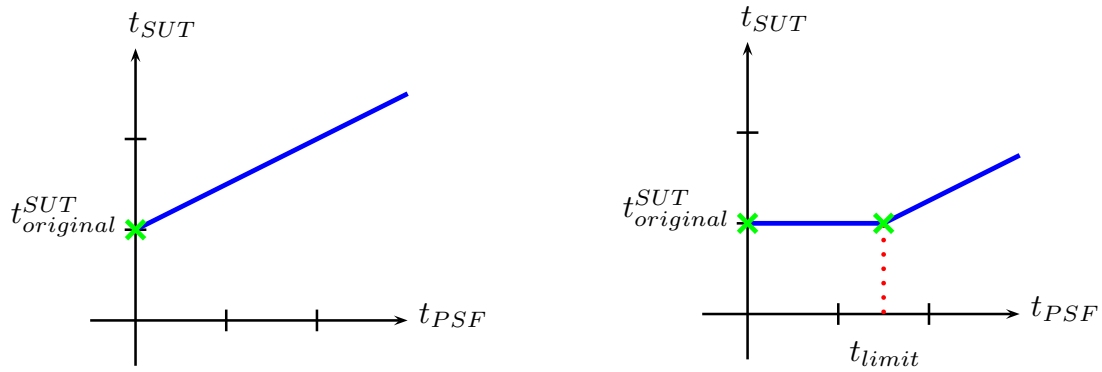
In the preceding section, we described a methodology and an implementation of simulating the software functionality of a CUS in order to determine the potential gain of optimization. Not all bottlenecks can be analyzed this way. With the parallelization of processing tasks by modern architectures and operating systems, concurrency issues and analysis of individual CPU usage becomes increasingly important. For this reason, we adapt our previous approach for the simulation of CPU behavior to a multi-core setting and show how the use of *CPU stubs* can complement the analysis of CUS using *SSF* with respect to concurrency.

1) **Determination of the CPU bottleneck**
The SUT has to be defined and a suspected bottleneck (CUS) has to be identified, which is done by common software performance engineering (SPE) [22], [23], e.g., profiling or tracing. Now, several performance indicators have to be determined:
   - t^{CUS}: Time spent in the bottleneck (CUS).
   - t^{SUT}: Time spent in the software module or function (SUT) from which the CUS is a part.
   - t^{CUS}_{busy}: Time spent in the CUS using the CPU. It includes the user-mode time as well as the system-mode time, see [21].
   - $t^{CUS}_{waiting}$: Time spent in the CUS waiting to be scheduled, see also: process state "Ready" in [21].
   - $t^{CUS}_{blocked}$: Time spent in the CUS waiting for an event, see also: process state "Blocked" in [21].

   The measured values have to be deterministic within several performance test runs.
2) **Validate CPU Bottleneck**
Here, a simple validation of the chosen CUS will be done. The *system influencing CPU PSF* is inserted in front of the CUS and the performance measurements will be repeated increasing the time spent in the *PSF* (t_{PSF}). The measured time of the SUT mainly follows one of the diagrams given in Figure 4.
In Figure 4a the increase of the *system influencing CPU PSF* leads to an arithmetically increasing amount of time spent in the SUT. Therefore, the CUS seems to be a CPU bottleneck. Hence, the next step can processed.
In the other case, Figure 4b shows that an increase in the execution time of the CUS does not increase the time spent in the SUT for $t_{PSF} < t_{limit}$. This points out that the CUS is no bottleneck for the system. Another potential CPU bottleneck has to be identified (Step 1).
This step can be done to remove overhead as it excludes the CUS from being mistaken as a CPU bottleneck easily. This step is optional.
3) **Study the Bottleneck Performance Behavior**
The value $t^{CUS}_{blocked}$, as determined in Step 1, will be used to evaluate the CPU utilization of the CUS.
A value of $t^{CUS}_{blocked} = 0$ means that there are no waiting periods triggered by the CUS while executing. So, the process will not be interrupted by the CPU except there are external events, e.g., scheduling. In this case, the methodology can be used as provided.
A value of $t^{CUS}_{blocked} > 0$ means that the process switches to the "Blocked" state. Here, the trace files recorded in Step 1 have to be studied further, in order to identify successive working and waiting periods of the process. The following steps of this methodology have to be done for every working period starting from the biggest to the smallest working period. The waiting period will be simulated with the *system non-influencing CPU PSF*. The simulated waiting time will normally be constant, if no further reduction caused by optimizations of this time period can be expected.
4) **Flat CPU Stub - Evaluate the Optimization Potential**
Now, a *flat CPU stub* will be used to determine the optimization potential. A *flat CPU stub* is a *DPS*, which only simulates the functional behavior of the CUS using the *SSF*. Hence, it only introduces small overhead in the system and can be used to simulate the ideal time behavior of the CUS. This can be used to analyze the maximum performance gain of the SUT as it is not the same as $t^{CUS} = 0$, especially in multi-core or parallel processing environments. As the final result often depends on several in parallel working threads or processes. Therefore, the following values have to be measured:
   - t^{STUB}_{flat}: Time spent in the *flat CPU stub*.
   - t^{SUT}_{flat}: Time spent in the SUT including the *flat CPU stub*.

   An indicator of the possible optimization amount can be evaluated by calculating:
   - $t^{CUS}_{reduced} = t^{CUS} - t^{STUB}_{flat}$
   - $t^{SUT}_{reduced} = t^{SUT} - t^{SUT}_{flat}$

   $t^{CUS}_{reduced}$ is the time, which has been reduced in the CUS. The $t^{SUT}_{reduced}$ value describes the total possible optimization gain. If the CUS is executed more than once in sequence, the maximum number of iterations per CPU ($iter$) has to be evaluated. Hence, $t^{CUS}_{reduced} * iter$ and $t^{SUT}_{reduced}$ has to be compared. It is possible to use the factor $iter$ because the same CUS is executed, so each iteration takes the same amount of time. The values t^{CUS} and t^{SUT} are taken from Step 1. Now, the calculated values can be compared and the following cases can be evaluated:
   - $t^{CUS}_{reduced} = t^{SUT}_{reduced}$: This means that the SUT directly depends on the CUS. Hence, there are no system dependencies, i.e., "hidden bottlenecks". Additionally, no "over optimization", as described



(a) CUS seems to be a bottleneck

(b) CUS is no bottleneck

Figure 4. Validate CUS as a Bottleneck

in [24], can be done. The more time optimized in the CUS the better it is. In this case, the next step of this methodology is Step 7, i.e, optimize as much as possible. However, in case of an expected hardware bottleneck, Step 5 can be done. This behavior is typically for batch or procedural processing in single core environments.

- $t^{CUS}_{reduced} > t^{SUT}_{reduced}$: In this case, the possible optimization amount is less than the time spent in the CUS. Thus, there are system dependencies, which have to be studied further and we can move on to the next step. This behavior can mainly be seen in multi-core and parallel processing systems. As there might be parallel threads or processes, which additionally delays the execution after the actual bottleneck has been reduced. This is particularly the case if a change over in the critical path has happened (see [25]).

The case $t^{CUS}_{reduced} < t^{SUT}_{reduced}$ does not has to be considered. This would mean that the speed up of the execution time in the SUT is more than has been reduced in the CUS. Hence, the execution time of $t^{SUT} - t^{CUS}$ would has been decreased, but, the software within this part of the SUT has not been changed.

As it is only an indicator, the time $t^{SUT}_{reduced}$ delivers no information about the amount of optimization, which has to be done in the CUS, especially for $t^{CUS}_{reduced} > t^{SUT}_{reduced}$.

5) **Idle CPU Stub - Evaluate System Dependencies**
Here, the *flat CPU stub* will be extended using the *system non-influencing CPU PSF*. This is called an *idle CPU stub*. The total simulated time is the total processing time of the CUS (t^{CUS}_{busy}). Hence, the following equation holds $t^{CUS}_{busy} = t^{STUB}_{idle}$. Where, t^{STUB}_{idle} is the time spent in the *idle CPU stub*. Now, the performance measurements will be redone and the t^{SUT}_{idle} value, which is the total execution time of the SUT including the *idle CPU stub*, shall be recorded. Dependencies between an *idle CPU stub* and the system can be evaluated using the values: t^{SUT}_{idle} and t^{SUT}. Thus, the total execution time of the original SUT will be compared to the execution time of the SUT using the *idle CPU stub*. The following cases can be separated:

- $t^{SUT}_{idle} = t^{SUT}$:
  This means that the total execution time of the SUT has not changed due to the usage of the *idle CPU stub*. Whereas, the *idle CPU stub* only uses the CPU at the very first beginning and than hands the CPU over to the system. However, the total execution time of the SUT has not been changed. Hence, the conclusion that no other process is blocked by the CPU can be done. Therefore, adding CPUs to the system does not provide a significant performance improvement. Nevertheless, as of Step 2, the CUS is the bottleneck.
- $t^{SUT}_{idle} < t^{SUT}$:
  Here, the total execution time of the SUT decreases by using an *idle CPU stub*. Therefore, further processes are at least partially available in the "Ready" queue. In this case, these processes can be executed earlier. Therefore, the total execution time decreases. An optimization of the CUS as well as an additional CPU decreases the total execution time.

The case that $t^{SUT}_{idle} > t^{SUT}$ does not have to be considered as it means that reducing the amount of instructions would lead to a longer execution time. This is not possible in typical CPU bound systems.

This step evaluates dependencies between running processes in the system and the CUS. Moreover, information about the influence of adding CPUs to the

system can be achieved. However, the measurements do not provide any information whether a faster CPU will increase the total execution time. Albeit expected that a faster CPU will increase the total execution time. As the process is CPU bound, the amount of instructions determines the total execution time. Using a faster CPU means that each cycles and, hence, an instruction, is executed faster.

6) **Busy CPU Stub - Cost Estimation**
The *flat CPU stub* will be extended with the *system influencing CPU PSF*. Now, the performance measurements will be repeated and the time spent in the *system influencing CPU PSF* (t_{PSF}) will be varied from zero to the total execution time of the CUS (t_{busy}^{CUS}). Typically, the time spent in the PSF will be increased by 10% of the total execution time for each iteration. This can also be redone if a particalur time slice, e.g., between 20% and 30%, identified a change over in the critical path as explained in this step. The following values have to be measured:

- t_{busy}^{STUB}: Time spent in the *busy CPU stub*.
- t_{busy}^{SUT}: Time spent in the SUT including the *busy CPU stub*.

Using these results, two different types of bottlenecks can be distinguished:

- Total Bottleneck:
  In this case, the measured values of the execution time from the SUT is linearly increasing. Thus, an optimization of the CUS will always result in an improvement of the execution speed and, therefore, decrease the latency. This result should have been already achieved in Step 4.
- Limited Bottleneck:
  If the processing of the SUT depends on other functions respectively on their results, the graph might look similarly as given in Figure 5. The graph is split in two parts. In the first part, $t_{PSF} \leq t_{limit}$, the time of the SUT is constant at a minimum value (t_{min}^{SUT}). Within this area, the chosen CUS is no bottleneck to the system as an increasing in the amount of processing (t_{PSF}) does not lead to an increased execution time (t_{SUT}). At t_{limit} the behavior of the CUS chances to a CPU bottleneck. As can be seen in the figure, the time spent in the SUT increases along the time spent in the *system influencing CPU PSF* (t_{PSF}). This evaluation shows that an optimization of the bottleneck can only decrease the latency in the SUT to a given value (t_{min}^{SUT}).

This information can be used to identify "hidden" bottlenecks, e.g., a "hidden" bottleneck appears at t_{limit} of Figure 5. This limit is basically the maximum,

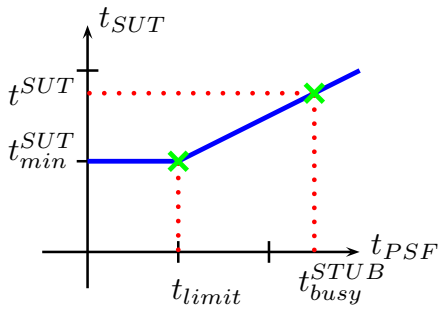


Figure 5. Limited Bottleneck

which can be achieved by an optimization of the CUS. Hence, it can be compared to a changeover in the critical path (see [25]). Additionally, the information can be used for a cost-benefit analysis. Thus, a gain-oriented improvement can be done.

7) **Optimization of the Software**
Now, the software module or function has to be optimized. Hence, the results from the cost-benefit analysis can be used for a software improvement related to the optimum between cost and effort. Finally, the performance of the software component has to be measured again. A new bottleneck has to be identified (first step) if the results show that the performance targets are not achieved yet.

A CPU bound process can be optimized by using the described methodology. This is especially true for multi-core systems and multi-threaded applications where the bottleneck is single-threaded. Here, a changeover in the critical path after an optimization can be identified before doing the optimization itself.

C. Summary

In this section, we have presented the gain-oriented performance optimization methodology that can be used to optimize CPU bound bottlenecks. Increasingly, single-threaded systems are parallelized and software functions that constitute bottlenecks in the system use multiple threads that can be executed over a number of CPUs.

The following section provides an extension of the *CPU stubs* to simulate the performance behavior of multi-threaded bottlenecks.

IV. Multi-Threaded CPU PSF

As shown, using *CPU stubs* provide many possibilities to simulate the performance behavior of a system. The combination of *SSF* (see Section II) and *PSF* (see Section III) enable the execution of several performance simulations in order to determine the system's performance gain that can be achieved by its optimization.

To rebuild the system's functional behavior, the *SSF* was introduced in Section II. It is possible to generate the

required functionality half-automated. However, the amount of work that has to be done in order to be able to execute the desired simulations can still be rather high. In some cases, e.g., the generation of files, it can be more difficult to rebuild the functional behavior of a small piece of code than of larger software modules. In the "file" example above, it would be easier to use the created file as a whole within the *SSF*.

To be able to do this, the performance behavior of more complex software modules has to be rebuilt by the *CPU PSF*. For that the *CPU PSF* are extended to simulate the performance behavior of components under study that are even multi-threaded in this section.

A. Objectives

This section presents the main reasons for simulating a system's multi-threaded performance behavior using *CPU PSF*:

- In some cases, it is necessary to rebuild the performance behavior for more complex and even multi-threaded software modules. As shown above, the construction of the *simulated software functions* can be quite difficult for small functions. For that the possibility to simulate multi-threaded modules within the system and, thus, simplify and fasten the use of *CPU stubs* can extend the *CPU stubs*' field of application.
- In addition to that the methodology presented in III-B should be used in complex environments where the identification of performance bottlenecks can really be difficult. Here, *CPU stubs* can show their benefits when investigating the impact of a multi-threaded component onto the system's performance.

As presented, it is necessary to extend the available *CPU stubs* from [1] to be able to simulate the detailed performance behavior of more complex software modules, e.g., modules, which are multi-threaded. This extension is described in the following subsections. Next, the approach is evaluated.

B. Approach

To rebuild the performance behavior of complex, multi-threaded software modules, a defined interface is introduced to enable code generation. The data structured this way can be used to generate code that can be executed for the simulation. This interface uses four different actions to describe the behavior of a multi-threaded application. They can be separated into actions that are "specific for the *CPU stub's* performance simulation" and into actions that "model the creation and termination of threads" within the system.

- Describe the threads' performance behavior
  - **RUN** The *RUN* action is used to describe situations where CPU is used by the thread. This action is initialized by the keyword 'run' and followed by a number presenting the duration of this action in microseconds.
  - **SLEEP** This action is used to simulate the time while the thread does not use the CPU. This occurs when the process is blocked due to system calls or user interaction. The keyword for the *SLEEP* action is 'sleep' that is also followed by the time in microseconds.
    The blocked time, caused in the thread by waiting for another thread to terminate, e.g., waiting for a *JOIN*, is not simulated using the *SLEEP* action. If the addressed thread is still running and did not terminate until the call of the *JOIN* action, the calling thread is automatically blocked by the system. As this is the desired behavior for the simulation, it will not be simulated with a *SLEEP* action.
    Moreover, the blocked times are sometimes very short, e.g., a few microseconds. Thus, they can not be simulated accurate enough. Therefore, these short blocked time slices are not modeled via *SLEEP* actions but included into the *RUN* actions.
- Describe the threads' creation and interaction
  - **CREATE** In order to simulate multi-threaded software modules, it is necessary to create new threads. This is done by using the *CREATE* action. Its structure is 'create NUMBER', where NUMBER is used to identify the thread that has to be created. This is needed to be able to start the correct *PSF* for this thread.
  - **JOIN** To synchronize the created threads, the *JOIN* action is introduced. It consists of the keyword 'join' and the NUMBER representing the thread that has to be joined. As described in the *SLEEP* action, a call of *JOIN* blocks the calling thread until the addressed thread terminates. For that reason, a call of the *JOIN* action can result in an amount of time where the thread is blocked, even if this is not specially modeled within the interface.

For each of the threads that are rebuilt, an own file to describe its performance behavior has to be build in our environment. For that the code generation will produce one *PSF* for each thread. Those reference each other by using *CREATE* and *JOIN* actions.

The next section presents one appropriate implementation of the *multi-threaded CPU PSF*. Measurements are applied to obtain the data needed for the described interface.

C. Implementation

In order to rebuild the original performance behavior of a multi-threaded CUS and, therefore, to provide the data for the previously described interface, some measurements have to be done, to gather the needed amounts of execution time

and the information about the splitting and joining threads within the CUS.

1) Measurements: The Linux Trace Toolkit (LTTng) [26] is applied to collect all the information by using the available kernel markers. Based on scheduling events, the processes' execution time can be calculated. The markers for process creation and termination are used to determine the various threads that are spawned during the execution of the CUS and their respective timestamps.

2) Performance Simulation Actions: The *PSF* for the *RUN* and *SLEEP* actions are build as shown in [11]. The *RUN* action is generated as a *system influencing CPU PSF*, whereas, the *SLEEP* action is simulated by a *system non-influencing CPU PSF*. With the evaluation of the LTTng traces, the combination of busy and idle times can be simulated according to the original behavior.

3) Thread Behavior Actions: The actions *CREATE* and *JOIN* are used to describe the behavior of the threads that are simulated. In this approach, this is done by using POSIX threads [27]. When creating a new thread, its corresponding *PSF* is started. As described in the interface's structure, the *JOIN* action is used to synchronize the threads. This enables the *CPU stubs* to recreate the same thread behavior as the original software component does.

With the shown possibility to describe the performance behavior of a multi-threaded CUS, a case study can be performed to evaluate the approach's usability and the fact that multi-threaded performance behaviors can be rebuilt by *CPU PSF*.

D. Case Study

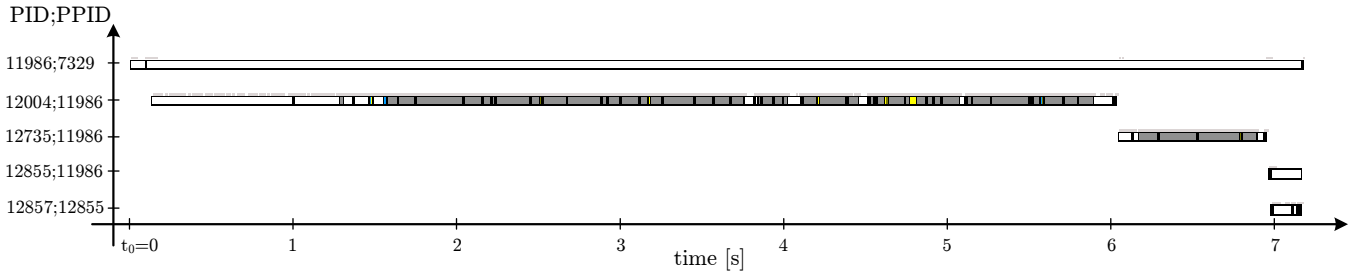
In this case study, the presented approach to rebuild complex performance behaviors of multi-threaded applications using *CPU stubs* should be validated. For that a compiling process of the GNU compiler (GCC version 4.6.0) is simulated.

1) Test Execution: The application runs on an Arch Linux operating system (kernel version 2.6.30.9) patched with the LTTng framework (version 0.160). The hardware is based on an Intel Centrino Core 2 Duo CPU with 2.26 GHz and has 4GB RAM. The calibration of the *CPU PSF* has been realized as described in [11].

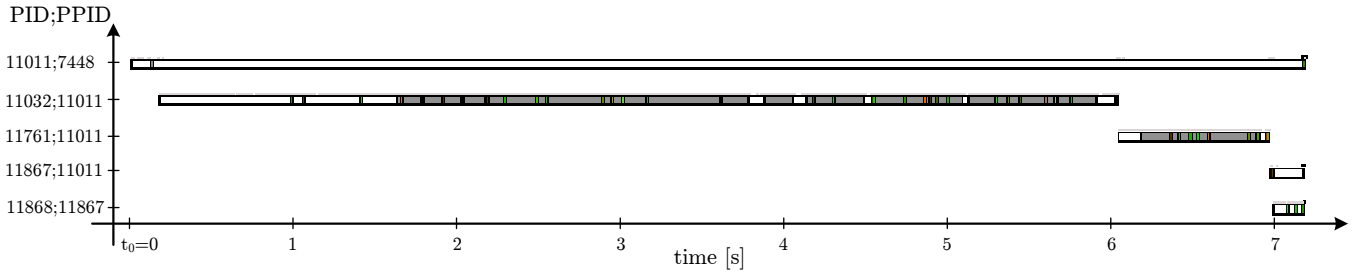
The test case chosen in this case study is the compile process to create a binary file using the GNU compiler (GCC version 4.6.0). In order to perform this case study a C-file containing an implementation of a quicksort algorithm is compiled. The compiled program is only of small size and does not have any further influence on this case study. This test case has been chosen as the GCC uses several threads depending on each other to compile the application.

2) Execution: This case study is performed in four major steps; record the data, generate the performance behavior, simulate the performance behavior and validation. These steps describe the process of creating the *CPU PSF*. The *PSF* can be used within the methodology, as shown in Section III-B, to simulate the performance behavior. E.g., the *CPU PSF* of a single thread can easily be adjusted to create an *idle CPU stubs*, which is Step 5 in the methodology for using *CPU stubs*.

1) **Record the Data**
   As a first step, the original performance data of the GCC call is recorded using the LTTng framework. This step corresponds to Step 1 of the methodology shown in Section III-B.
   Figure 6a shows the original performance behavior of the used GCC call. The x-axis presents the time in seconds. To get comparable results, the begin of the execution is set to $t_0 = 0$. The y-axis depicts the PID and the PPID of the created threads. The drawn bars present the performance behaviors of the single threads. When the CPU is used by the thread, the bar is gray, whereas, a white bar is used for the times where the threads are blocked or waiting for I/O. The other bars (black and colored) present other events that occurred during the execution of the system, such as paging and scheduling. As described in Section IV-B the time slices of those events are too short to be simulated accurate enough and, thus, are included into the *RUN* action.
   Regarding the PPID of each thread, Figure 6a also shows the threads' respective child threads and the order they are built. It can be seen that the thread with the PID 11986 (called Thread 11986) creates the Threads 12004, 12735 and 12855. Thread 12855 itself creates Thread 12857 before it joins back to its parent (Thread 11986).
2) **Generate the Performance Behavior**
   The data measured via LTTng has been transformed to fit to the described structure. As shown in Figure 6a the different process states are identified and rebuilt by performance behavior actions. The gray bars build *RUN* actions, whereas, the white bars are rebuilt by *SLEEP* actions. Additionally, the events that occurred when creating and synchronizing the threads are described by *CREATE* and *JOIN* actions.
   Listing 10 shows an excerpt of the interface's structure for the first created thread (Thread 11986) (see Figure 6a). Line 1 shows a call of the *SLEEP* action, which consumes 12780 microseconds. In Line 2 a call of the *RUN* action is carried out. The creation of threads is realized by *CREATE* actions as shown in Line 3. By the call of the *JOIN* action, e.g., in Line 7, the execution of the simulation blocks until the corresponding thread joined back. In this example, the numbers used for the creation and synchronization of the threads are the PID of the original threads. Beside the fact that this could be any number, it has been



(a) Original Performance Behavior



(b) Simulated Performance Behavior

Figure 6. Comparison of Original and Simulated Performance Behaviors of the GNU complier (GCC version 4.6.0)

chosen to be the PID of the thread for a simplified debugging in more complex environments.

```
1  sleep 12780
2  run 54
3  create 12004
4  run 3
5  sleep 11911
6  run 27
7  join 12004
8  run 74
9  sleep 9495
10 ...
```

Listing 10. Transformed Measured Data Used to Generate Simulation Code

For each of the five threads that run during the execution of the test, the code is generated as a combination of *system influencing CPU PSF*, *system non-influencing CPU PSF* and POSIX threads. Those pieces of code can be combined and used as the performance simulation code.

3) **Simulate and Validate the Performance Behavior of the *CPU stub***
After the *CPU PSF* have been built, the simulation is executed. To be able to compare the results, the measurements within the simulation were also done using the LTTng framework.
Figure 6b shows the simulated performance behavior of the system recorded by the LTTng framework. The x-axis presents the time in seconds and the y-axis depicts the PID and PPID of the executed threads. As in Figure 6a, the gray bars present the busy times and the white lines are used for the blocked times. In Figure 6b, it can be seen that the scheduling events (black and colored) are still triggered and thus are also recorded while the simulation is running. But events for paging that occurred during the original execution do not appear within the simulation. This is due to the *PSF*, as the *CPU stubs* shall only consume the CPU and shall not influence further elements of the system, e.g., the memory. Hence, it is the desired behavior that no page faults occur.

4) **Validation**
Figure 6 shows the comparison of the original (Figure 6a) and the simulated (Figure 6b) performance behavior of the GCC call. The evaluation of both graphs depicts that the performance of the GNU compiler call was rebuilt almost exact by the simulation. The running and blocked states of the single threads can be simulated accurately. The blocked times that occur during this simulation are a combination of the used *SLEEP* actions and the blocked time that occured due to the *JOIN* actions.
Minor inaccuracies of the simulation originate from the page faults that are not rebuilt by the simulation and the non-deterministic occurrence of the scheduling events. However, the performance behavior of the threads fits the values measured within the original execution of the system with high accuracy.

This simulation demonstrates that the *CPU stubs* can also be used to simulate the performance behavior of big software modules that may even be multi-threaded. This approach

can also lead to a simplified construction of the software functionality if the rebuild of a complex functional behavior is easier compared to the one of small modules, e.g., when generating whole files, as done by using the GNU compiler.

E. Summary

In this section, it has been shown that it is possible to rebuild the performance behavior of complex and multi-threaded software modules. An interface that can be used to describe the performance behavior of the system is introduced. Using this interface also simplifies the generation of code to simulate different performance optimization levels within the system by changing the performance parameters of the running and waiting times within the input data. Furthermore, the interface can be used in the previously described methodology and can lead to a faster and easier usage of *CPU stubs*. This uniform interface also increases the portability of *DPS* as it can be easily extended by code generators for further programming languages. Being able to recreate a complex performance behavior offers more possibilities when trying to detect the threads whose performance is critical for the system's performance.

The following section discusses the related work in the various areas that are affected by this paper.

V. Related Work

There are three major areas, which have to be considered for related work: *DPS*, *CPU stubs* and *SSF*.

Dynamic Performance Stubs: There are two different research scopes in simulation of the performance behavior. In [28] it is explained how performance of inner loops can be modeled at the instruction level and which effect they have on the memory/cache performance. Although the possibility of modeling software modules exists, the high degree of granularity of this approach reduces the usability for stubbing whole software functions/modules.

In [29] the usage of smart stubs for software analysis of functions and modules which are partly not available yet is described. Hence, the stubs simulate the budget regarding storage and time resources, which have been estimated, for the to-be-implemented software parts. Also, mainly a management point of view for the non-existing software will be taken.

The *DPS* in our approach will be used for stubbing already implemented and measured software parts in order to find the bounds of the performance improvement within that part. This procedure helps to identify the real gain of the performance improvement without really improving it and, additionally, shows the next bottleneck. So the cost-benefit analysis for improvement activities can be achieved in a more realistic way, because a proper simulated result is better than a simple estimation.

CPU stubs: The CPU executes the instructions of the applications. The scheduler exists to decide which process the CPU has to execute next (see [21], [30], [31]). It maintains several queues about the states of all processes: running, ready or blocked. If no process is either in the running or ready state the idle process is executed by the CPU.

From a process perspective, the process can be either executed by the CPU or is suspended. *CPU stubs* are targeting to simulate the time (CPU) behavior of a bottleneck. Hence, *CPU stubs* have to be able to switch the state. As only the scheduler decides whether a process is in the running or ready state, it is not possible to enforce a process to be in the running state in non-real time systems acting from user space side. Only the possibility to increase the chance to be scheduled soon into the running state exists, e.g., using priorities.

Hence, the *CPU stubs* have to simulate the remaining states "running" and "ready/blocked". The running state can be realized by a "do-nothing loop" and a state change can be initiated by delaying the process execution, e.g., using a sleep function.

In [32], [33] a problem with simulating a dedicated amount of time with do-nothing loops is described. Despite the problems seen, there are big differences in the approaches. The procedure is targeting the area of bulk-synchronous parallel jobs, which are realized as do-nothing loops. The focus is to optimally utilize each of the included processors. So the processes always try to run, ignoring the amount of time needed for the operating system per processor. As soon as the operating system has something to do, the userspace application will be scheduled out and the total execution time will be delayed.

Our *system influencing CPU PSF*, however, will be calibrated in an otherwise idle system with enough time for the OS. As experimentally proved in [11], in our environment the execution time of a process can be simulated with a do-nothing loop, predictably in contrast to [32], [33]. Additionally, because of the fact that such a loop has a defined number of instructions these loops can be used to simulate the time behavior of processes.

Simulated Software Functionality: A key functionality of the *DPS* is to record and to recreate functional behavior using the *SSF*. The recording requires the serialization of internal data-structures into a format from which they can be recovered at a later point. This functionality has been predominantly implemented in distributed systems where objects and code are marshaled for exchange between peers.

Most closely related to our serialization approach is the work in [14], [34] in which a "MPI Serializer" has been introduced. The target of this project is the efficiently and automated marshaling of C++ data structures. The tool generates automatically marshaling and unmarshaling code for the message passing interface (MPI), which is often used

as communication interface in high performance computing (HPC). The "MPI Serializer" is is based on the C++ serialization possibility of the "GCC-XML" project [16], which uses the gcc abstract semantic graph (ASG) scheme [35] to determine the *serialization specification*.

To some extent, our approach is similar to [14], [34] as both projects needs to serialize C++ data structures. However, it differs in many details. E.g., it has been decided to store and restore the functional behavior of software modules, which will be replaced by a stub. This can be used to remove a software bottleneck. In contrast, the focus in [14], [34] is to provide marshaling code for the message parsing interface.

However, both projects are based on the abstract semantic graph scheme provided by the "GCC-XML" project.

In [36], a lightweight fact extractor is presented. It utilizes XML tools, i.e., XPATH and XSLT, to extract static information from the C++ source code files. The approach is to transfer the source code into "srcML", which is a XML representation of the file. The fact extractor is mainly used to parse and search the source code. This technique is often used for reverse engineering, maintenance, testing or even in general development of software systems. This approach is based on "CPPX" [37], which is an open source C++ fact extractor. The fact base, which is generated by "CPPX", can be used as input for software development tools, such as integrated development environments (IDE). It enhances these tools' functionalities, for example by source code visualization, object recovery, restructuring and refactoring.

As in [14], [34], the approach of [36] highly differs from our approach, as it is not supposed to store and recreate the functional behavior of software modules. [36] mainly delivers a XML presentation of the extracted facts of the source code.

VI. Conclusion and Future Work

This paper evaluates our novel approach to the replay of functional behavior of software algorithms by the *SSF*. This functionality is used by *CPU stubs*, which are a subset of the *DPS* framework. These *CPU stubs* consist of the *CPU PSF*, which have been extended to simulate multi-threaded applications.

In order to achieve these results, two distinct functionalities have been combined: *SSF* and *CPU PSF*.

It has been shown that the functionality of software can be replaced by the *SSF* almost automatically. A methodology to use this functionality is given and a possible implementation is provided. This is concluded by an industrial case study.

Moreover, *CPU stubs* can be used to simulate the performance behavior of complex software modules. This can simplify the creation of *software simulation functions* and can be used to determine the impact of multi-threaded components onto the system's performance. The usability of the introduced interface, to describe the multi-threaded performance behavior, has been shown in a case study.

The following aspects concerning the *SSF* and *CPU stubs* will be addressed in the future work:

- *SSF*
  - Evaluate the memory influence of the *libSSF* and identify improvement possibilities to reduce this overhead.
  - The *libSSF* will be extended to easily trace more different data structures. Especially, it should be improved to stub "STL lists"and "STL vectors".
  - By now, the *libSSF* is based on some workarounds. Further possibilities to access private and protected members of classes will be evaluated, e.g., by using friend classes. Moreover, a redesign will be done.
- *CPU Stubs*
  - Consideration of further events within the interface, to rebuild the communication behavior of the threads more accurate.
  - Definition and evaluation of a methodology on the creation of *multi-threaded CPU PSF*.

Additionally, the methodology of *CPU stubs* will be evaluated by complex and multi-threaded case studies.

We have shown that *CPU stubs* can be used to simulate the functional as well as performance behavior of a CUS. This can be used to evaluate performance optimization potential depending on the system. Hence, it is possible to identify "hidden" bottlenecks as well as further improvement possibilities. Moreover, a gain-oriented performance analysis can be achieved.

Acknowledgment

This research was supported by a long term evolution system developing company. The authors would like to thank the LTE group for the excellent support and contributions to this research project. For careful reading and providing valuable comments on draft versions of this paper we would like to thank Sebastian Röglinger and the reviewers.

References

[1] P. Trapp, M. Meyer, and C. Facchi, "Using CPU Stubs to Optimize Parallel Processing Tasks: An Application of Dynamic Performance Stubs," in *ICSEA '10: Proceedings of the International Conference on Software Engineering Advances*. IEEE Computer Society, 2010.

[2] P. Trapp and C. Facchi, "Performance Improvement Using Dynamic Performance Stubs," Fachhochschule Ingolstadt, Tech. Rep. 14, Aug. 2007.

[3] R. Jain, *The art of computer systems performance analysis*. Wiley and sons, Inc., 1991.

[4] P. J. Fortier and H. E. Michel, *Computer Systems Performance Evaluation and Prediction*. Burlington: Digital Press, 2003, vol. 1.

[5] D. J. Lilja, *Measuring Computer Performance: A Practioner's Guide*. New York: Cambridge University Press, 2000.

[6] N. H. Gunther, *The Practical Performance Analyst*. McGraw-Hill Education, 1998.

[7] J. J. Marciniak, *Encyclopedia of Software Engineering*, 2nd ed. John Wiley & Sons Inc, 2002.

[8] A. Bertolino and E. Marchetti, *Software Engineering: The Development Process - A Brief Essay on Software Testing*, 3rd ed. John Wiley & Sons, Inc., 2005, vol. 1, ch. 7, pp. 393–411.

[9] I. Sommerville, *Software Engineering*, 6th ed. Pearson Studium, 2001.

[10] J. Hughes, "Performance Engineering throughout the System Life Cycle," SES Inc., Tech. Rep., 1998.

[11] P. Trapp and C. Facchi, "How to Handle CPU Bound Systems: A Specialization of Dynamic Performance Stubs to CPU Stubs," in *CMG '08: International Conference Proceedings*, 2008, pp. 343 – 353.

[12] P. Trapp, C. Facchi, and S. Bittl, "The Concept of Memory Stubs as a Specialization of Dynamic Performance Stubs to Simulate Memory Access Behavior," in *CMG '09: International Conference Proceedings*. Computer Measurement Group, 2009.

[13] P. Trapp and C. Facchi, "Main Memory Stubs to Simulate Heap and Stack Memory Behavior," in *CMG '10: International Conference Proceedings*. Computer Measurement Group, 2010.

[14] W. Tansey and E. Tilevich, "Efficient automated marshaling of C++ data structures for MPI applications," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, apr. 2008, pp. 1–12.

[15] "GCC, the GNU Compiler Collection," online: http://gcc.gnu.org/, June 2011, [July 5, 2007].

[16] GCC-XML, "GCC-XML, the XML output extension to GCC!" online: http://www.gccxml.org/, 2010, [July 5, 2011].

[17] F. Khan, *LTE for 4G Mobile Broadband: Air Interface Technologies and Performance*, 1st ed. Cambridge University Press, 2009.

[18] P. Trapp, C. Facchi, and M. Meyer, "Echtzeitverhalten durch die Verwendung von CPU Stubs: Eine Erweiterung von Dynamic Performance Stubs," in *Workshop "Echtzeit 2009 - Software-intensive verteilte Echtzeitsysteme", Informatik Aktuell*. Springer Verlag, 2009, pp. 119–128.

[19] R. Srinivasan and O. Lubeck, "MonteSim: A Monte Carlo Performance Model for In-order Microarchitectures," *ACM SIGARCH Computer Architectur News*, vol. 33, no. 5, pp. 75–80, Dec. 2005.

[20] Y. Etsion and D. Feitelson, "Time Stamp Counters Library - Measurements with Nano Seconds Resolution," The Hebrew University of Jerusalem, Tech. Rep. 2000-36, 2000.

[21] A. S. Tanenbaum, *Modern Operating Systems*, 2nd ed. Prentice-Hall, Inc., 2001.

[22] C. U. Smith, "Formal Methods for Performance Evaluation," in *Introduction to Software Performance Engineering: Origins and Outstanding Problems*, 2007, pp. 395 – 428.

[23] C. U. Smith and L. G. Williams, "Best Practices for Software Performance Engineering," in *Int. CMG Conference*, 2003, pp. 83–92.

[24] S. McConnell, *Code Complete: A Practical Handbook of Software Construction*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2004.

[25] S. Chapman, "Finding the Critical Path - A Simple Approach," in *CMG '09: International Conference Proceedings*. Computer Measurement Group, 2009.

[26] "Linux Trace Toolkit Next Generation," online: http://www.lttng.org, 2011, [July 5, 2011].

[27] "The Open Group Base Specifications Issue 6 IEEE Std 1003.1," online: http://pubs.opengroup.org/onlinepubs/009695399/basedefs/pthread.h.html, 2004, [July 5, 2011].

[28] G. Marin and J. Mellor-Crummey, "Application Insight Through Performance Modeling," in *26th IEEE International Performance Computing and Communications Conference (IPCCC'07)*, New Orleans, Apr. 2007.

[29] D. J. Reifer, "The Smart Stub as a Software Management Tool," *SIGSOFT Softw. Eng. Notes*, vol. 1, no. 2, 1976.

[30] D. P. Bovet and M. Cesati, *Understanding the LINUX KERNEL*, 3rd ed. O'Reilly, 2005.

[31] A. Fugmann, "Scheduling Algorithms for Linux," Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2002.

[32] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick, "System Noise, OS Clock Ticks, and Fine-Grained Parallel Applications," in *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*. New York, NY, USA: ACM Press, 2005, pp. 303–312.

[33] D. Tsafrir, "The Context-Switch Overhead Inflicted by Hardware Interrupts (and the Enigma of Do-Nothing Loops)," in *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*. New York, NY, USA: ACM Press, 2007, p. 4.

[34] W. Tansey, "Automated Adaptive Software Maintenance: A Methodology and Its Applications," Master's thesis, Virginia Tech, May 2008.

[35] N. A. Kraft, B. A. Malloy, and J. F. Power, "A tool chain for reverse engineering c++ applications," *Sci. Comput. Program.*, vol. 69, no. 1-3, pp. 3–13, 2007.

[36] M. Collard, H. Kagdi, and J. Maletic, "An XML-based Lightweight C++ Fact Extractor," in *Program Comprehension, 2003. 11th IEEE International Workshop on*, May 2003, pp. 134 – 143.

[37] CPPX, "CPPX - Open Source C++ Fact Extractor," online: http://www.swag.uwaterloo.ca/cppx, july 5, 2011.

Intelligent Look-Ahead Scheduling for Structural Steel Fabrication Projects

Reza Azimi
Dept. of Civil & Environmental Engineering
University of Alberta
Edmonton, Canada
razimi@ualberta.ca

SangHyun Lee
Dept. of Civil & Environmental Engineering
University of Michigan
Ann Arbor, USA
shdpm@umich.edu

Simaan M. AbouRizk
Dept. of Civil & Environmental Engineering
University of Alberta
Edmonton, Canada
abourizk@ualberta.ca

Abstract— **Look-ahead Scheduling can be a difficult task, especially for non-repetitive, irregular work packages and activities such as occur in structural steel fabrication, where a variety of equipment, material and skilled work is required to manufacture unique steel pieces. Although punctual look-ahead scheduling based on the most recent system analysis and project data can significantly improve productivity and project control, this technique has not been extensively used in the construction industry. This paper presents an intelligent and integrated simulation-based framework in which real-time as-built data are captured and along with intelligently generated as-planned data are fed into the simulation model for look-ahead scheduling. A distributed simulation system based on the High Level Architecture is proposed to enhance the performance of the system. The ability of the proposed system to incorporate real-time actual data along with different scenarios that represent the dynamic work environment and external factors opens new doors to improve the accuracy of look-ahead scheduling in the construction industry. To exhibit the feasibility of the proposed framework, a prototype system is developed and deployed in a steel fabrication company.**

Keywords: Construction Simulation; High Level Architecture (HLA); project monitoring and control; real-time data capture, Artificial Neural Networks.

I. INTRODUCTION

This paper amplifies the work originally presented in [1]. Currently, industrial steel structures are popular for constructing a variety of buildings, from heavy industrial buildings and petrochemical refineries to sheds, shelters, and roofs. Reduced construction times, efficiency, and cost-effectiveness can be considered as the major benefits of using industrial steel elements. Detailing, procurement, steel fabrication, shipment, and site erection are the five major phases in a typical industrial steel construction project [2]. Steel fabrication, one of the complex phases, refers to the production of steel pieces through a series of operations, including detailing, fitting, welding, and surface processing in a confined environment called a fabrication shop [3, 4]. Better control over quality, effective labor utilization, and reduction of waste are some advantages of manufacturing steel elements in a fabrication shop. Material handling and inspection activities occur frequently during the fabrication process. There are a large variety of steel pieces produced, in terms of both dimensions and processing requirements. Steel fabrication activities require an assortment of equipment, material and labor disciplines in order to produce steel pieces.

The complexity and variety of products, and the large number of potential resources, activities, interactions, constraints and uncertainties, make the planning and control of ongoing and forthcoming steel fabrication projects a complicated task. Project planning involves several activities, including master scheduling and short-term or look-ahead scheduling, which are two key elements in successful delivery of the projects. Master scheduling refers to the overall view of the projects and general fabrication strategies. Such scheduling may be used for several reasons: forecasting demand, long-term coordination (e.g., regarding material requirements and staffing level) and rough budgeting. However, master scheduling suffers from a lack of information about actual durations and cannot be properly detailed far into the future. Conversely, a look-ahead schedule is a detailed plan for work packages to be completed in a relatively short time frame. Look-ahead scheduling helps project managers focus on the work packages that should be done at some time in the future and the corrective actions in the present that will lead to finishing those work packages on time, within budget and to a specified quality. These detailed schedules should be developed and updated in a timely manner based on the actual project performance data and the conditions in the construction environment, to precisely represent the tasks that have been done and the tasks that remain [5]. Such a detailed schedule is a solid foundation in terms of performance analysis and taking effective corrective actions.

This paper proposes a steel fabrication shop modeling approach for efficient look-ahead scheduling of steel fabrication projects in the Construction Synthetic Environment (COSYE) based on the High Level Architecture (HLA) infrastructure. In this approach, real-time, high-quality actual project data are captured and fed into a simulation model along with the uncertainties and the factors influencing the productivity of the fabrication shop. In this way, reliable updated look-ahead schedules are generated by the simulation model and "guesstimates" are rarely needed. Current practice and research carried out regarding look-ahead scheduling is discussed, and the conceptual framework of the integrated real-time simulation-based scheduling system is then described. The feasibility of the proposed framework is demonstrated by developing a prototype system that was used for a case study in a steel

fabrication shop. The advantages and limitations of the proposed system are also discussed in this paper.

II. Background and Literature Review

The basis of Look-ahead Scheduling (LAS) is similar to regular scheduling. Usually, activities or work packages are regular and predictable. When that is not the case, work packages need to be appropriately classified and work packages that are considered similar may differ in terms of process duration and required resources. Similar to regular scheduling, standard data should be generated using time studies or expert judgment. Once these standards are established, work packages can be scheduled, allowing foremen and project managers to forecast and control projects over comparatively short time intervals. LAS usually refers to a foreman's schedule, settled and tracked by foremen on a short-term basis. The foremen determine which work packages would be processed by their crew during the next few days, and the project managers monitor the accuracy of the schedule. This monitoring leads to identifying factors that affect the production rate. Once these factors are identified, the project managers can address the implicit causes and routinely enhance the accuracy of the LAS. LAS helps improve productivity by eliminating or reducing time spent that is not adding value to projects. It also helps ensure all the required resources and material are ready for ongoing projects at any time needed. Since LAS sets realistic and obtainable goals for a short time span, as a psychological effect, the workers tend to get the work done as soon as possible.

Smith [6] argues that LAS has been successfully implemented in different domains such as material handling, quality assurance, manufacturing, maintenance, engineering, and assembly operations. LAS has been largely used for mass production systems where immediate follow-up and corrective actions are a must in the case of deviations [7]. As an example, combined with pairwise comparison LAS was utilized for scheduling random operations in job-shops [8]

In spite of the fact that very little information on LAS is provided in the literature, effective LAS are crucial to the successful completion of construction projects [5]. Daneshgari and Moore [9] state that on average up to 70% of construction job schedules experience changes. They observed four projects over a four-month period, ranked the impact of the unscheduled activities on the lost productivity and concluded that implementation of LAS is a great tool to improve the productivity. Similarly, Hadavi and Krizek [10] concluded that short-term scheduling results in higher productivity compared to long-term scheduling. Studying decision support systems in manufacturing operations and determining the types of data required to plan and control effectively, Schmahl [11] concluded that LAS can be used to support continuous improvement efforts in production operations. Scheduling problems could be solved for either the next hour or the next few weeks by LAS [12]. Guidelines regarding developing work packages for effective utilization of LAS as well as the situations where LAS is more likely to succeed have been discussed by Ramireza-Valdivia et al. [7]. Finally, emphasizing the key role of LAS in enhancing production control, Ballard [13] proposed strategies for improving LAS.

In construction, many operations are repetitive and involve uncertainty and resource constraints. This motivated researchers to deploy discrete-event simulation for LAS problems [14]. Simulation is a mathematical-logical model representing a real-world evolving system. Users can use the simulation model to analyze and forecast the performance of a system considering different scenarios. Actual data captured from ongoing projects along with the uncertainties of the construction environment can be fed into the simulation model to "tune it up" for generating better results [15, 16]. A proper updating process for simulation input modeling based on high quality data is necessary to achieve simulation accuracy. The emergence of new technologies has enhanced data acquisition systems by providing automated high-quality real-time data. For example, Radio Frequency Identification (RFID) has been used to track real-time locations of steel pieces in a steel fabrication shop [2] and monitor steel works in high-rise buildings [17]. The same data acquisition system based on RFID technology developed by the authors [2] is also used in the prototype system of this research to closely supervise the operations that lead to obtaining the benefits of LAS.

Current studies focus on real-time data acquisition and improving simulation input modeling, which involves finding statistical distributions of the model input parameters, such as the durations of different activities [14]. For instance, Song et al. [14] used Global Positioning System (GPS) technology to capture required data, such as truck hauling time, and update a simulation input model for LAS of asphalt hauling and paving projects. These simulation systems commonly have two characteristics: first, they usually model repetitive operations, and second, the final output of these operations is one or (rarely) a few limited products. This paper deals with simulating a steel fabrication shop, in which the operations are repetitive while each product (i.e., steel piece) is typically unique. This uniqueness means the time required for processing each steel piece varies. Estimating processing duration is dependent on productivity; the degree of precision depends on the nature of the work and is influenced by several factors. The relationship between these factors and the processing duration/productivity cannot be demonstrated in an accurate and clear manner, increasing the difficulty of estimating the steel piece processing activity durations which are used for updating the simulation input model. A well-structured Artificial Neural Network (ANN) model, which has an optimal structure regarding layers and nodes, is capable of learning from data sets and reliably approximates any complicated relationships between dependent and independent variables [18]. ANN models can also handle moderate amounts of noise, which is common in the historical data, and can generate knowledge from defective or noisy data [19]. ANNs have been widely used for modeling productivity in construction; for example, concrete construction productivity [20], formwork production rates [21] and pipe spool fabrication productivity [22]. ANNs have

also been exploited in this research to intelligently generate process durations for each steel piece to be manufactured, considering influencing factors. A framework is proposed to integrate a real-time actual data collection system for steel fabrication projects with an intelligent input data generation system and with simulation models, which utilizes the as-built data for updating input models and improving the simulation results for LAS.

III. LOOK-AHEAD SCHEDULING USING SIMULATION TECHNIQUE

In steel fabrication projects, LAS involves a number of uncertain factors and constraints. No project can be started earlier than a given date due to the limitations regarding the availability of required material, space and equipment. Each project should be delivered by a certain date depending on client demand and the conditions of the erection site. The scheduler should also take into account the limits on other resources such as skilled workers, cranes and active stations in the steel fabrication shop.

A typical steel fabrication project may contain a few hundred steel pieces, which makes using traditional techniques such as the Critical Path Method (CPM) a time-consuming and tedious exercise. Moreover, in the case of any deviation from the baseline or changes in resource availability, adjusting the schedule would be very difficult. Computer simulation is a powerful technique to efficiently react to system changes and generate updated schedules. Simulation is used here as the underlying technique to model the fabrication shop and resources and activities required to process steel pieces. To be effective, the simulation model should be updated based on the most recent system changes. Then, the impacts of these changes need to be observed by the model for modifying the LAS. To address this, this paper presents an intelligent and integrated simulation system based on the framework previously developed by the authors for automated and integrated project monitoring and control [2]. Several components of the framework established using the HLA infrastructure have been modified and the whole system is enhanced by adding intelligence for reliable look-ahead scheduling purposes. An as-planned database, as-built data acquisition, discrete event simulation, a steel fabrication process knowledge base, a calendar, and an intelligent adjuster are the major components of the proposed system (Fig. 1). HLA provides a reliable infrastructure for efficient integration of all the components of the LAS system. Interoperability, reusability, flexibility, and system speedup [2] are some advantages of utilizing HLA as the backbone for the proposed LAS system.

To have an effective LAS system, the following steps must be taken. First, the process of the steel fabrication should be well investigated and a simulation model based on that is established. For each project a baseline schedule is defined by a scheduler. During the execution course of the project, real-time data is collected from the fabrication shop and utilized to update the simulation model so that it reflects the dynamic nature of the project environment. Once the initial state of the simulation model is set, it can be used for experimenting with different scenarios and an updated LAS can be generated based on the simulation results.

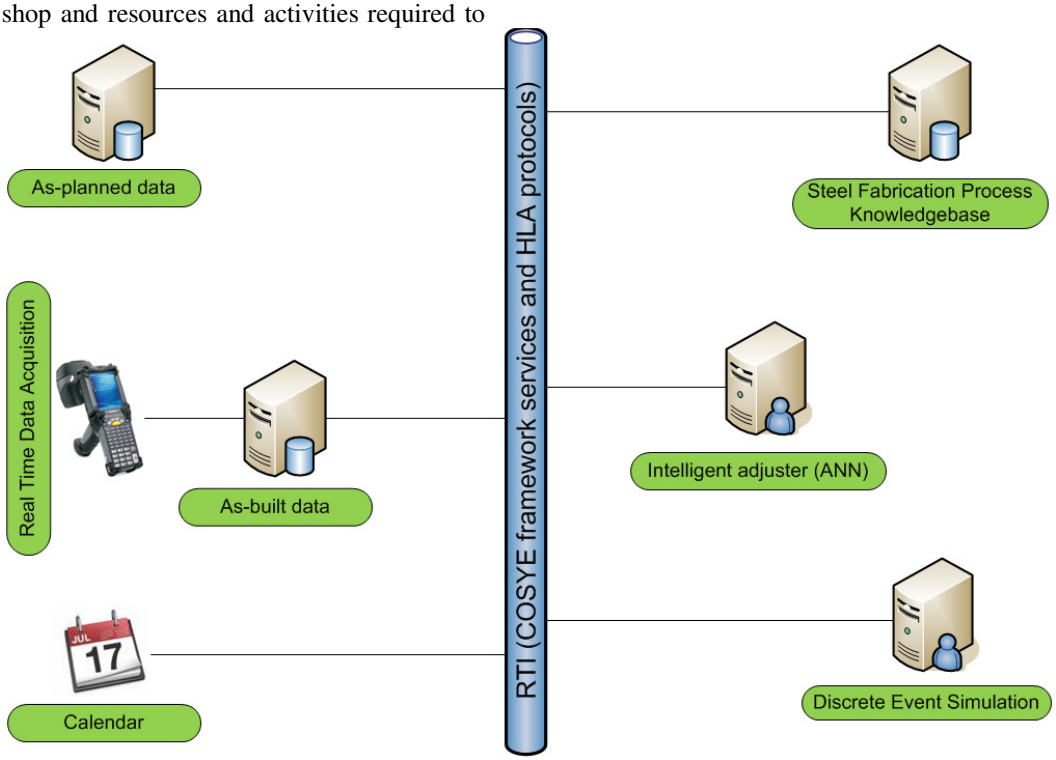


Figure 1. Integrated LAS system- a modified version of the model originally proposed by the authors[2]

A brief review of the steel fabrication process as well as the components of the proposed system is presented in the following sections to provide the reader with an overview of the research topic.

A. *Steel Fabrication Process*

A typical steel project consists of a number of steel pieces, such as beams, columns, or trusses, with different dimensions and specifications. Fabrication of steel pieces starts with the detailing area. In this area, the components of a steel piece are cut and/or punched according to the engineering design. Cut components are transferred to the fitting area to be fitted. Fitters bring the components together using tack welds to form the steel piece. Once inspected, the fitted piece is sent to the welding area where the welders weld the piece according to the provided specifications. Another inspection happens once the welding is done. If required, the piece is sent to the painting area, otherwise the piece is ready to be shipped to the erection site (Fig. 2).

Each area in the fabrication shop is composed of several stations which makes it possible to have several pieces processed in each area simultaneously. Steel pieces are transferred by rail carts or cranes depending on the situation. If a piece must be processed in a working area but there are not enough resources available to process that piece, it is piled in a certain storage area and waits until the required resources are available.

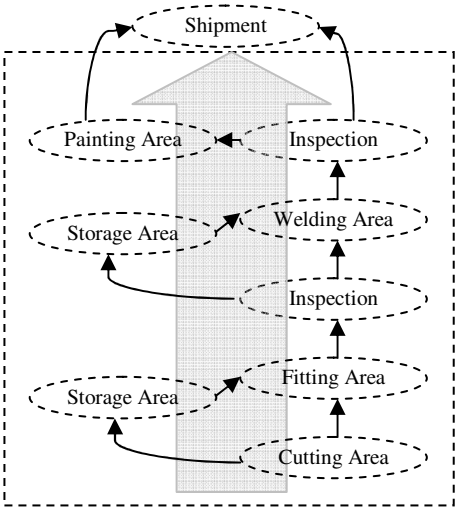


Figure 2. Steel fabrication process

Rework will be necessary if a piece is rejected by an inspector.

B. *Components of the Proposed System*

The backbone of the proposed system is the High Level Architecture (HLA) which has been discussed in more detail in the next section (i.e., System Implementation).

Some components of the proposed system such as Discrete Event Simulation, the calendar, and real-time data capturing have already been developed and detailed information about these components can be found in [2]. A brief explanation of those components is also provided below.

B1. Real-Time Data Capturing

As in [2], Radio Frequency Identification (RFID) technology has also been used in this research to collect real-time data. RFID tags are put on steel pieces and the locations of the tagged pieces are tracked using portable RFID tag readers. The location tracking also captures the time a piece enters each area and the time it leaves that area. These data are then interpreted to provide project performance data, project progress, and activity duration in the steel fabrication process knowledge base component. Inter-process communication between the tag readers and the as-built database occurs via Transmission Control Protocol over Internet Protocol (TCP/IP).

B2. Steel Fabrication Process Knowledge Base

The real-time data acquisition system generates raw data. Useful performance data, such as man-hours spent on fitting or welding each piece, project percent complete, production rate and so on need to be extracted from these raw data to be used for updating LAS. Interpretation of these raw data can be automated by correlating the location of a steel piece with fabrication events and activities based on the experts' knowledge regarding the process of fabrication. A comparison between the location of a steel piece and pre-defined areas discloses meaningful information about the fabrication operation. For example, the entrance of the steel piece into the fitting area is recorded as a start-fitting event and exiting the fitting area is considered as an event called end-fitting. Once done, it is time to extract activity information from the event data. Generally speaking, each activity is begun by an event and is finished by another event. So, for the previous example, the duration of fitting activity for each steel piece can be calculated considering the start- and end-fitting events. To be effective, the gained knowledge should be managed properly in terms of capture, arrangement and retrieval of knowledge. To enforce that, knowledge bases are frequently used by the practitioners and researchers. As an example a process knowledge base for asphalt hauling is developed in [14]. A sample hierarchical structure of the required knowledge for structural steel fabrication data interpretation is presented in Table I.

TABLE I. SAMPLE ELEMENTS OF THE STEEL FABRICATION PROCESS KNOWLEDGE BASE

Area	Action	Event	Activity
Fitting Area	Enter	Start fitting	Fitting steel piece
Fitting Area	Leave	End fitting	Waiting (storage area)
Welding Area	Enter	Start welding	Welding steel piece
Welding Area	Leave	End welding	
…	…	….	

Different techniques can also be used to calculate other performance data. For example, the progress of each piece is determined by translating piece location to percent complete using the rule of credit based on expert knowledge [2]. For instance, the experts may consider piece fitting as 40%

progress in the fabrication phase. Project percent complete can thus be calculated by summing up the weight of each piece times its progress, all divided by the weight of the project.

B3. Discrete Event Simulation

This component is a modified and improved version of the discrete event simulation (DES) model of steel fabrication shops developed earlier [2]. Modeling the fabrication shop and prediction of the behavior of the fabrication shop is not based on mathematical models and analytical solutions. Discrete event simulation was utilized for this purpose due to the fact that simple closed form analytical solutions are not available for the modeling steel fabrication process. However, the DES federate has the capability to incorporate any available mathematical or analytical solutions.

To be effective, the DES component of the proposed LAS system should be updated in a timely manner to reflect changes in the fabrication shop's environment. The proposed DES is a self-adjustive component that exploits the captured actual real-time data to update its input models. The initial state of the simulation model is set based on the most recent data. The actual data can also be used to generate updated distributions that are substituted with the earlier input model. The intelligent adjuster is a component that takes care of this responsibility. It automates the input-model update procedure with no slow and error-prone human involvement.

Scheduling is performed in DES model based on the earliest-due-date (EDD) dispatching rule, such that the project with the earliest due date is selected and processed first. The scheduling engine of the DES model enables automated project schedule updates to be generated and stored in an MS Access database. The DES model is also capable of performing earned value analysis, and cost and schedule performance indices can be calculated for all the steel pieces.

B4. Intelligent Adjuster

Updating input models can be carried out by external prediction models [23]. The proposed intelligent adjuster is an autonomous Artificial Neural Network (ANN) component that is trained with the actual data available in the aforementioned steel fabrication process knowledge base to generate updated distributions, e.g., regarding duration of different activities required to process each piece of steel for the simulation input model. While it is difficult and sometimes not feasible for estimators to consider in their estimations all the influencing factors for a huge number of steel pieces with different dimensions and specifications, artificial intelligence has the capability to overcome this issue and forecast and update the distributions to be used in the steel fabrication simulation model. In this way, the accuracy of the simulation model is enhanced by explicitly modeling uncertainty variables and their impacts on the performance of the fabrication shop and ongoing projects.

B5. Calendar

Schedules are highly influenced by day shift hours, night shift hours, overtime hours, and holidays. The user defines these parameters within the calendar federate which sets related initial values for the DES model [2].

IV. SYSTEM IMPLEMENTATION

A) Infrastructure

A prototype system has been developed for look-ahead scheduling of steel fabrication projects. The proposed system implements the Construction Synthetic Environment (COSYE) software environment [24], an HLA-based simulation environment developed at the University of Alberta. HLA is a reliable infrastructure for integrating different components of a simulation model, called federates, into a single distributed simulation model, referred to as a federation [25]. HLA promotes interoperability between simulations and aids the reuse of models in different domains, which leads to reduced time, cost and efforts to create a synthetic environment for a new purpose [25]. Development of simulation models of different construction applications significantly benefits from these features of HLA because these simulation models usually share a number of common components [2]. HLA can be characterized by three main components [26]: HLA rules, the HLA interface specification, and the Object Model Template (OMT). The OMT provides a common framework for data exchange between different federates. The run-time infrastructure (RTI) is a piece of software that complies with the HLA specifications and provides services such as synchronization, communication, and data exchange between federates.

COSYE is composed of an RTI, an environment that is optimized for development of federations in different construction domains, and a suite of generic modeling elements. During run time, COSYE provides necessary communication, information exchange, and data sharing protocols using an RTI that assures synchronization, coordination and consistency between different federates.

In this prototype system, the primary project LAS (baseline) is prepared by a scheduler and is stored in a MS Access database. Captured real-time actual data are also stored in a MS Access database. The base model of the DES component (federate) was developed within the Simphony.NET simulation environment [27] for modeling steel fabrication operations. Figure 3 depicts the interface for the DES federate [28].

B) Factors affecting the intelligent adjuster's forecasting

The process knowledge base, demonstrated in Table 1, is used to extract fabrication activity information from the actual data and feed the intelligent adjuster component.

Fitting and welding operations are the critical operations in the structural steel fabrication and usually take more than 80% of the available resources on average. In the proposed system the intelligent adjuster is composed of two Artificial

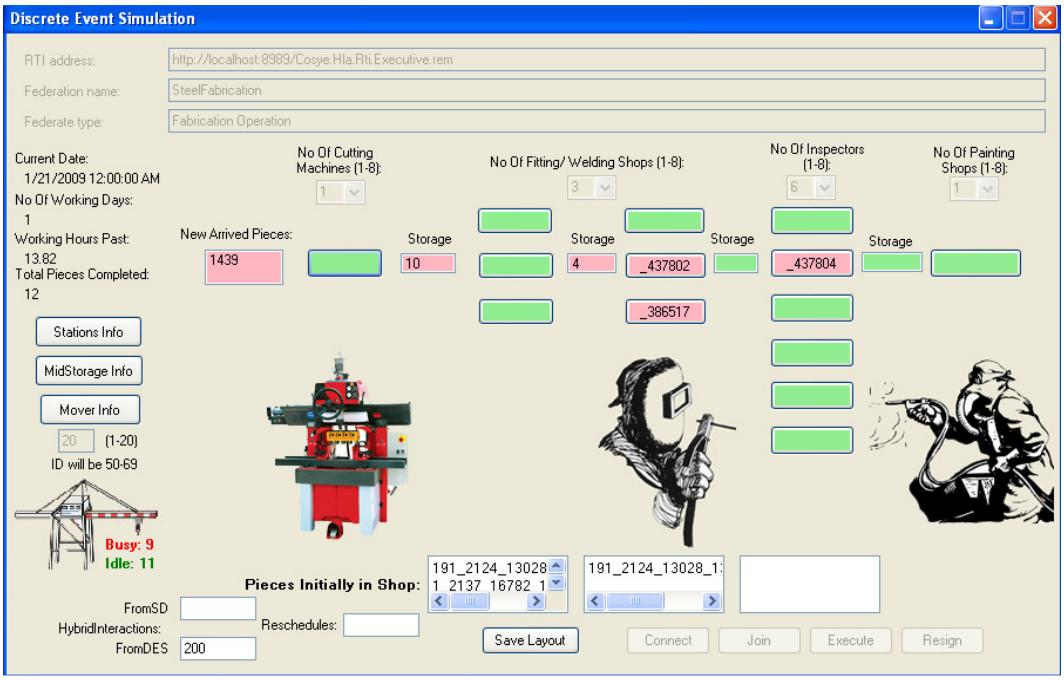
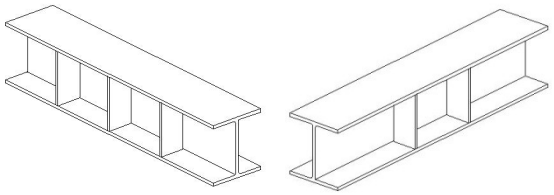


Figure 3. The DES federate [28]

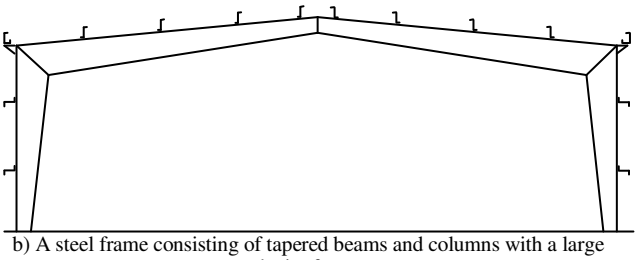
Neural Network (ANN) models and predicts the steel fitting and welding productivity/durations based on the complexity of each steel element and other influencing factors. Influencing factors can be divided into two major categories: the steel piece itself and the fabrication shop environment. Song et al. [29] proposed four piece-oriented influencing factors, such as number of fittings, number of cutouts, piece length, piece weight, and two influencing factors regarding the fabrication shop environment for the fitting operation, such as worker rank and work shift. There are two concerns regarding the proposed piece oriented factors. Firstly, although it is clear for beams and columns, piece length is a vague concept for other steel pieces such as frames and stairs. As an example, for a square steel frame, one person may consider the side length as the piece length while another person may use the diagonal of the square as the piece length. Piece dimension is an important factor because piece movement and piece flipping in each operation are highly affected by this property of steel pieces. However, piece weight commonly has a close and positive correlation with piece dimension (i.e., the bigger the piece, the greater its weight). This means that by considering piece weight as an influencing factor the piece dimensions are implicitly addressed. Secondly, the number of fittings and cutouts are two influencing factors in the fitting operation but not necessarily for the welding operation. Two different approaches can be taken considering the influencing factors. One approach is to define the factors for each specific operation, while the other approach is to define general influencing factors that can be used for different operations in steel fabrication. Within this research the second approach has been taken because of its universality and usage in the whole fabrication process. Therefore, the influencing factors considered in this research that are related to the characteristics of the steel piece include piece complexity – replacing the number of fittings and cutouts – and piece weight; the ones that are related to the shop environment are the rank of the workers and the working shift in which the pieces are manufactured.

Having said that, the inputs of the intelligent adjuster are the weight of the steel element (piece/assembly), the shift in which it has been fabricated, the rank of the worker who has processed that steel element (the higher the rank the more experienced the worker) and piece complexity. Piece complexity is represented by a parameter called “complexity factor.” “Complexity factor” refers to the number of the components in each steel piece. For instance, the steel beam shown in figure 4a is composed of an I shaped beam and 5 stiffeners. Thus, there are 6 components forming that steel piece which results in a complexity factor equal to 6.

Steel pieces differ, sometimes significantly, regarding their complexity factors. In other words, while some steel pieces are fairly simple (e.g. Fig. 4a), some pieces can be considerably more complicated (e.g., Fig. 4b). The complexity factor can also be calculated for each division with the same concept, i.e., total number of steel components divided by the number of steel pieces forming the division. A template was also developed to automatically capture complexity factors and weight of steel pieces from 3D models (Fig. 5) of steel projects. These 3D models use Building Information Modeling (BIM) which is a common way to construct a building virtually before building it in the real world, bringing the structures from concept to reality [30]. Automatically capturing information from 3D models with minimum human involvement guarantees high-quality data with great speed for data analysis purposes.



a) Two side views of a steel beam with 5 stiffeners.



b) A steel frame consisting of tapered beams and columns with a large complexity factor.

Figure 4. Sample of structural steel pieces with different complexities

A sample of data captured automatically from a 3D model by the developed template is represented in Table II.

The intensity of complexity is an indicator that determines how complicated a steel element is (i.e., piece complexity) and has a direct relationship with the complexity factor. It is one of the input variables utilized for training the intelligent adjuster and is defined as in Table III.

TABLE II. SAMPLE OF DATA CAPTURED BY THE DEVELOPED TEMPLATE FROM 3D MODELS

Piece ID	C.F.*	I.C.**	Weight(kg)
50A1	12	4	582
50A2	17	5	555
50A5	10	3	529
50A7	17	5	539
50A4	10	3	529
50A8	17	5	542
50A6	11	3	519
50A15	19	5	293
50A16	9	3	280
50A20	9	3	132
50A18	7	2	137

* Complexity Factor
**Intensity of Complexity

TABLE III. DEFINITION OF THE INTENSITY OF COMPLEXITY

I.C.	Definition	Explanation
1	Not complicated	1 ≤ C.F. < 4
2	moderately complicated	4 ≤ C.F. < 8
3	complicated	8 ≤ C.F. < 12
4	very complicated	12 ≤ C.F. < 16
5	extremely complicated	16 ≤ C.F.

C) Training the intelligent adjuster

The intelligent adjuster uses two back-propagation networks, each with 4 input nodes, one hidden layer, and one output node at the output layer.

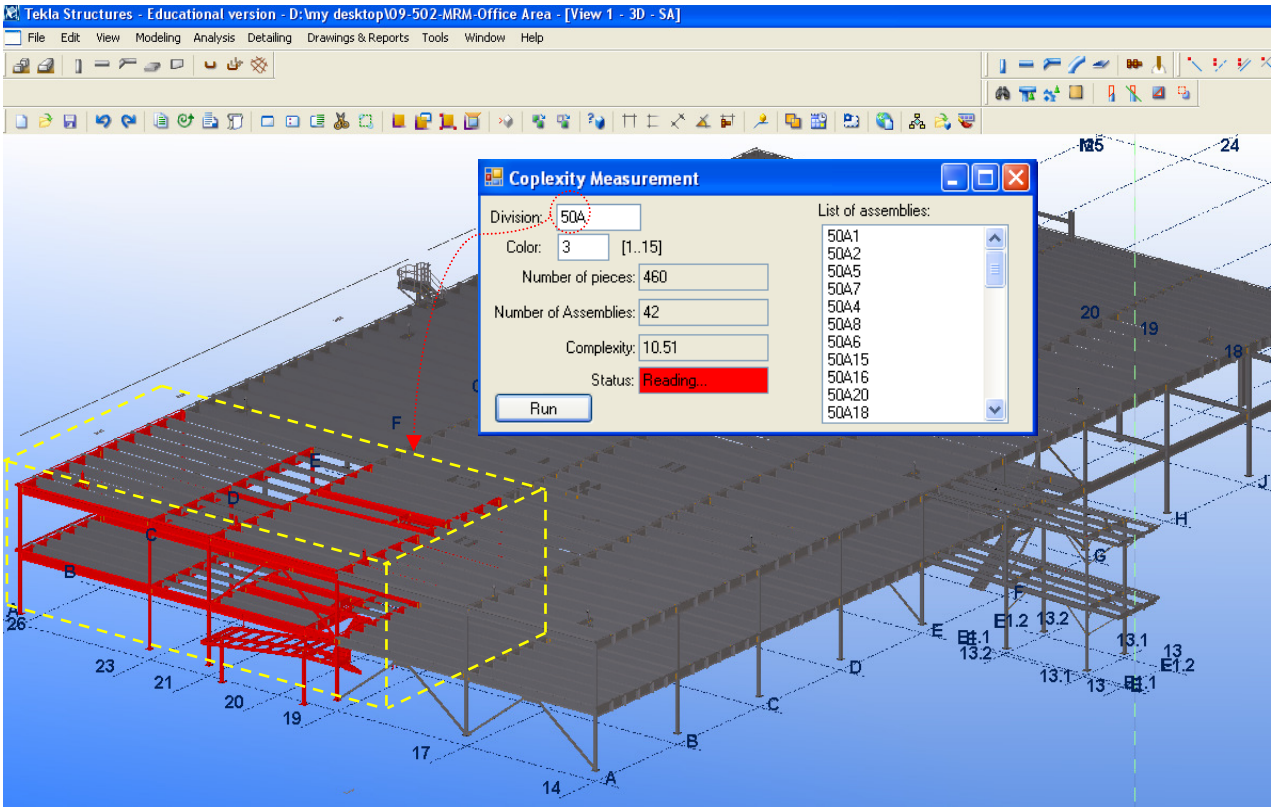


Figure 5. Developed template for capturing complexity factor from 3D models

The number of hidden neurons for the networks is calculated with the following equation [31]:

$$H_n=0.5\times(I+O) + \sqrt{P} \quad (1)$$

Where:
H_n: Number of hidden neurons
I: Number of inputs
O: Number of outputs
P: Number of patterns

"Neuroshell 2" [31] was used to train the networks. 114 fitting data points and 61 welding data points were captured by the real-time RFID data capturing system during a two-week time/case study used for training and validating the ANN models. For the fitting operation, 92 data points were randomly selected for training and 22 data points were used for testing. The learning rate and momentum of the fitting ANN model were set to 0.1. The initial weight of the links within the ANN model was set to 0.3 and numeric range of the linear scaling function used for the input layer was [-1,1]. For the welding operation, the number of training and testing data points were 49 and 12 respectively and the initial settings for learning rate, momentum etc. were similar to the settings of the fitting ANN model. The labor-driven nature of the steel fabrication process may lead to variance in productivity or activity duration even for similar steel pieces processed with laborers with the same rank and in the same shift. In this research one assumption is that such a variance is trivial; if that is not the case some techniques such as data filtration or using averages of the data can help in compensating for variances in the data. Sample actual data regarding welding operation that were used for training the intelligent adjuster are presented in Table IV.

TABLE IV. HISTORICAL DATA USED FOR TRAINING THE INTELLIGENT ADJUSTER

Weight (kg)	Shift (Day:1-Night:2)	Rank	I.C.	Duration (Minutes)
519	1	2	3	109
87	1	1	4	163
122	1	2	2	55
919	1	1	5	197
549	2	2	3	160
111	1	1	3	90
973	2	2	4	218
1250	1	3	4	121
…	…	…	…	…

The training results regarding duration of fitting and welding operations (in minutes) are summarized in Table V. There are several factors that may affect the maximum absolute error in terms of duration prediction presented in Table V. For instance, once a fitted or welded piece is rejected by an inspector, it requires rework, and the activity duration is extended and is greater than a situation in which rework is not required. Missing components of a steel piece or unclear or impractical drawings are other examples that extend the normal fitting/welding durations. With that said, and considering the wide duration ranges in fitting and welding data sets (i.e., from 10 to 290 minutes for the fitting data set and from 20 to 352 minutes for the welding data set), the trained networks are considered proportionately accurate in forecasting the fitting and welding durations with an acceptable margin of error.

TABLE V. INTELLIGENT ADJUSTER TRAINING RESULTS

Item	Fitting	Welding
Patterns processed:	114	61
R^2:	0.89	0.87
Mean absolute error:	13.46	26.14
Min. absolute error:	0.02	0.11
Max. absolute error:	73.20	53.00
Correlation coefficient r:	0.95	0.94

Two indicators, including R Squared (R^2) and the correlation coefficient (r), usually used for interpreting the neural network models, are also presented in Table V. As indicated in Table V, the R^2 values for the fitting and welding operations are 0.89 and 0.87 respectively. According to the network training results presented in Table V, the correlation coefficients for the fitting and welding operations are 0.95 and 0.94 respectively, which implies a strong positive relationship between the model outputs and the actual outputs for theses operations.

Figure 6 shows the trained network predictions against the actual welding duration values for the welding data points.

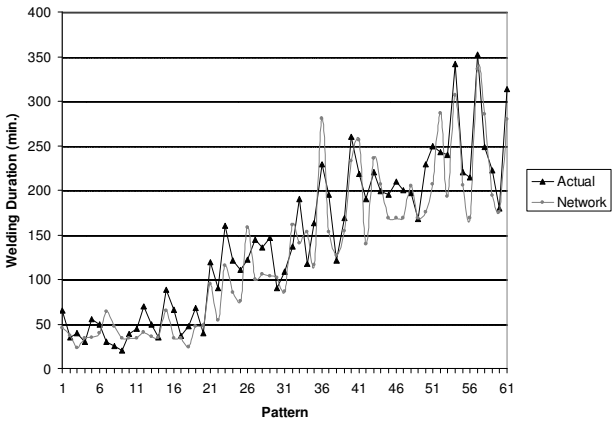


Figure 6. Actual vs. forecasted welding duration for data points

The trained ANN models were compiled as VB.NET code and used for developing the intelligent adjuster federate. This federate is capable of generating the distributions and other scheduling variables, such as productivity rate, based on the attributes (e.g. weight, complexity intensity, etc.) of each steel piece. An example of the generated scheduling variables is shown in Figure 7. This figure is a snapshot of an interface developed to provide managers with an intuitive report on the forecasted trend of productivity rates for fitting and welding activities within a 3-day time interval ending at

the date selected by the user. Such information can be a heads-up for impending deviations from desired values. The generated distributions then will be fed into the DES federate to improve its results. If required, the user can modify the influencing factors and experiment with different scenarios with the simulation model to find the best corrective actions to converge on the project goals.

V. CASE STUDY

A case study, the construction of an administrative office, was carried out to verify the feasibility and accuracy of the implemented simulation-based LAS system for steel fabrication projects. Because of the size of the building, the whole project is divided into 57 "divisions." Each division itself includes several hundred steel pieces, including beams, columns, stairs and frames. The baseline schedules for all the divisions were prepared by the scheduler and stored in the baseline database. The data acquisition system was set up in the fabrication shop. The major elements of the data acquisition system are RFID tags, a tag reader, and a router connected to a computer. Data capturing began with the launching of the project in the shop (Fig. 8). The DES federate models four activities – cutting, fitting, welding, and painting. The resources required for this model include active stations (which correlate with the number of operators) in different areas, cranes and rail carts, inspectors, and storage areas.

Usually, several divisions are processed at the same time in a fabrication shop and certain stations in each area are assigned to each division. The data capturing was carried out for all the steel elements that were being processed in the fabrication shop. In this way, scheduling information for the project level as well as the performance information (such as production rate at different areas) at the shop level was determined.

The actual data collected during the case study were used to examine the accuracy of the simulation model. However, during the case study several outstanding discrepancies were captured. First, some fitting and welding stations shut down due to the fact that a number of fitters and welders were laid off during the case study. Second, the actual process time in each station was longer than expected in many cases. Third, the production rate predicted by the simulation model was significantly greater than what happened in reality. These discrepancies detract from the utility of the LAS and mean that the simulation model needs to be properly modified and updated in some aspects to address these deviations.

Updating the simulation model is easy with regard to changing available resources (e.g., the number of active stations), but with variables such as process time and production rate, it is less straightforward. Variances in the process time and production rate can be caused by faults in estimation, external factors that were not considered in the simulation model, or both. While the faults in process time estimation are usually covered by the actual data, considering the effects of the external factors in the simulation model is a difficult task. For example, during the course of the case study, a number of workers received

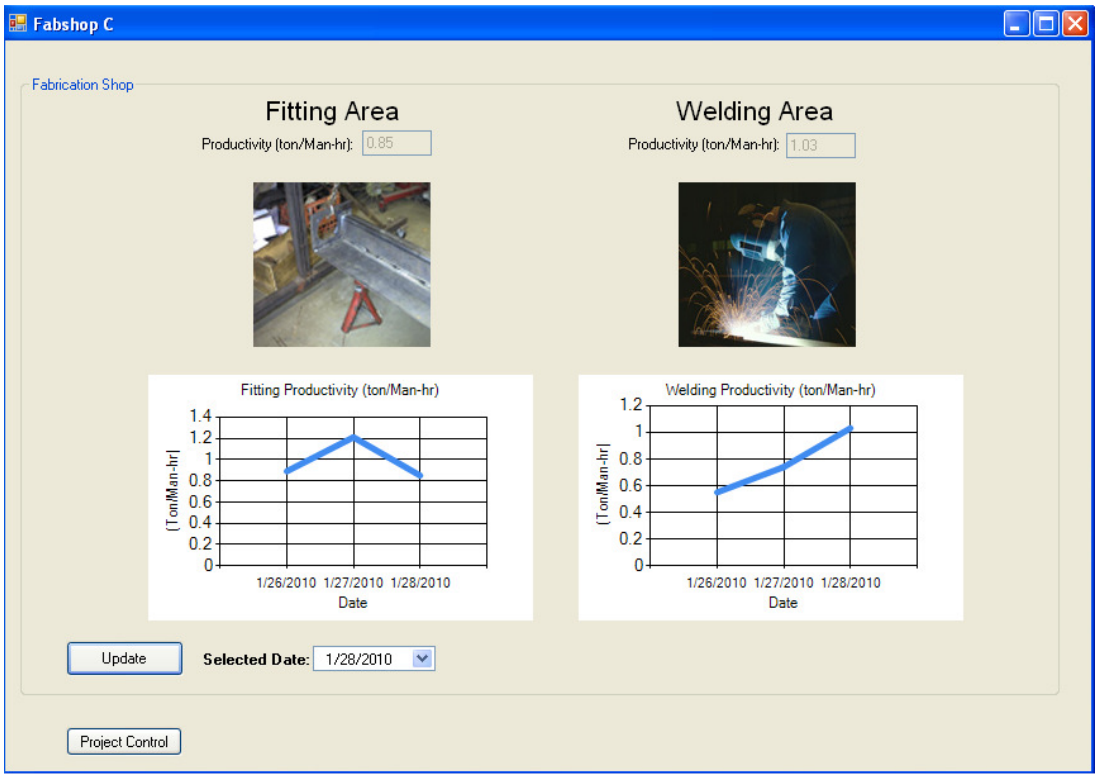


Figure 7. Fitting and Welding production forecasted by the intelligent adjuster component for the fabrication shop.

Figure 8. Collecting actual data using RFID technology

termination notices due to the recession conditions. After being informed that their employment will cease by the next month, the productivity of these workers was far lower than what was estimated. The arrival of a new project with a higher priority that had not been considered in the master plan was another issue that meaningfully affected the available resources (e.g., raw material), shop performance and LAS for ongoing projects during this case study. A thorough tracking of the variances regarding schedules and budget (i.e., assigned man-hours) resulted in determining seven major factors causing variances in the studied fabrication shop (Fig. 9). Once these factors are tracked, they need to be addressed to improve the performance of the ongoing projects and the fabrication shop itself. Project managers and foremen try to make improvements in different areas that cause deviations and variances over the course of the fabrication, but it is still a major concern for them to have reliable LAS for different projects which take into account current situation in the fabrication shop. This is the benefit of the LAS system which has been developed, which enables users to have updated LAS automatically generated based on the most recent actual data. An updated LAS is a sound foundation for decision making and system analysis. Sample results of the simulation model, based on 30 simulation runs, regarding LAS based on the actual data are shown in Table VI. The simulation results in Table VI represent four divisions including 321 steel pieces that originally were planned to be fabricated in the specified time interval (i.e., from January 18 – 27, 2010). Scheduled start and finish dates come from the project

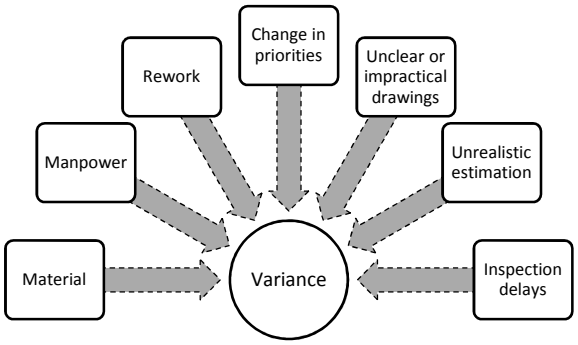


Figure 9. Major factors causing cost/schedule variance in the case study

baseline defined by a shop manager and two foremen, each with more than 15 years of experience. The calculated start and finish dates are the simulation output, generated based on the current shop conditions and the activity duration distributions forecasted by the intelligent adjuster for the pieces planned to be manufactured within the stipulated period of time. A comparison between the actual finish dates and the estimated finish dates generated by the people involved in scheduling, and what the developed intelligent system generated, results in determining estimation errors. Estimation error analysis reveals that the intelligent system could generate more reliable managerial information and LAS. As an example, for division 52A, human error regarding the finish date was 11 days, while the intelligent system had an error of 4 days. The average absolute estimation errors for the 4 divisions shown in Table VI was 9 days in the case of human judgment, while the intelligent system's estimation had an absolute error of 1.75 days on average. This may be attributable to the fact that the intelligent system generates schedules based on the recent conditions of the dynamic environment of the fabrication shop, perceived influencing factors and their combined interactions, while human beings are seldom able to consider all of these parameters in scheduling. It should be noted that a shortage of actual historical data that could be used for training the intelligent adjuster limited the accuracy of the developed system. Even though the performance of the developed LAS system is quite promising, having more actual historical data can enhance the intelligent adjuster forecasts and subsequently improve the accuracy of the simulation results.

TABLE VI. SAMPLE RESULTS OF THE SIMULATION FEDERATE

Division ID	No. of pieces	Scheduled Start Date	Scheduled Finish date	Calculated Start date	Calculated Finish Date	Actual Finish Date	Error in Estimating Finish Date (days)	
							Human Estimator	Intelligent LAS system
52A	64	18/01/2010	22/01/2010	18/01/2010	29/01/2010	1/2/2010	11	4
51A	34	19/01/2010	25/01/2010	19/01/2010	29/01/2010	29/01/2010	4	0
5A	103	19/01/2010	27/01/2010	20/01/2010	4/2/2010	5/2/2010	9	1
50A	120	21/01/2010	27/01/2010	25/01/2010	10/2/2010	8/2/2010	12	-2

CONCLUSION

Look-ahead scheduling of steel fabrication projects that considers projects' constraints as well as the fabrication shop's constraints is very complicated. This paper implements an intelligent and integrated simulation-based LAS framework for an actual case study in a steel fabrication shop. The system that was developed utilizes RFID technology to capture as-built data. As-built data along with the as-planned data are fed into the system; raw actual data are translated to meaningful data, and an intelligent component generates/forecasts essential scheduling variables based on the actual and historical data for each steel piece, considering several piece-oriented and environmental influencing factors, and updates the simulation model to allow it to produce reliable look-ahead schedules. The proposed system is expanded by employing High Level Architecture (HLA). In this way, the model is split into several components that are linked together in a well-structured format.

Unlike traditional scheduling methods that are static and time-consuming to update, the capability of the proposed system to dynamically and intelligently incorporate the most recent project data and changes in the fabrication shop environment can improve the accuracy of LAS and reduce input modeling burdens on end users.

REFERENCES

[1] Azimi, R., Lee, S., and AbouRizk, S.M., "Integrated simulation-based Look-Ahead Scheduling for steel fabrication projects", The Second International Conference on Advances in System Simulation (SIMUL 2010), Nice, France, 2010, pp. 129-133.

[2] Azimi, R., Lee, S., AbouRizk, S., Alvanchi, A., "A framework for automated and integrated project monitoring and control system for steel fabrication projects," Automation in Construction, 2011, 20(1), pp. 88-97.

[3] Azimi, R., Lee, S., and AbouRizk, S.M., "Automated project control system for steel projects," Joint International Conference on Construction Engineering and Management and Construction Project Management (ICCEM-ICCPM), Jeju, Korea, 2009, pp. 479-486.

[4] Azimi, R., Lee, S., & AbouRizk, S. M., "Resource performance indicators in controlling industrial steel projects", ASCE Construction Research Congress, May 2010, Banff, Canada, pp. 21-30.

[5] Hinze, J. W., Construction planning and scheduling, 3rd ed., Pearson, NJ., 2008.

[6] Smith M., Manufacturing Controls: How the Manager Can Improve Profitability, Wiley, Chichester, 1981.

[7] Ramírez-Valdivia M.T., Christian P.H., Frederick P., Aksoy B., and Zimmers Jr. E.W., "Development of a software tool to improve performance of packaging operations through short interval scheduling," Packaging Technology and Science, 2003, 16(5):pp. 179-89.

[8] Golenko-Ginzburg, D., Gonik, A., "Using look ahead techniques in job-shop scheduling with random operations," International Journal of Production Economics, 1997, 50 (1), pp. 13-22.

[9] Daneshgari P. and Moore H., "The secret to short-interval scheduling," Electrical Construction and Maintenance, 2009, 108(2).

[10] Hadavi, A., and Krizek, R.J., "Short-Term Goal Setting for Construction," Journal of Construction Engineering and Management, 1993, 119 (3), pp. 622-630.

[11] Schmahl, K.E., "Variation in success of implementation of a decision support/finite scheduling system," Production and Inventory Management Journal, 1996, 37(1):28-35.

[12] Zweben M. and Fox M., Intelligent Scheduling. Morgan Kaufmann: Bethlehem, PA, 1994.

[13] Ballard, H.G., "Lookahead planning: the missing link in production control," Technical Report No. 97-3, in Proceedings of the 5th Annual Conference of the International Group for Lean Construction, Griffith University, Gold Coast, Australia, 1997, pp. 13–25.

[14] Song L., Cooper C., and Lee S., "Real-time simulation for look-ahead scheduling of heavy construction projects," Construction Research Congress, 2009, pp. 1318-1327.

[15] Chung, T.H., Mohamed, Y., and AbouRizk, S.M., "Bayesian updating application into simulation in the North Edmonton sanitary trunk tunnel project," Journal of Construction Engineering and Management, 2006, 132(8), pp. 882–894.

[16] Lu, M., Dai, F., and Chen, W., "Real-time decision support for planning concrete plant operations enabled by integrating vehicle tracking technology, simulation, and optimization," Canadian J. of Civil Eng., 2007, 34(8), pp. 912–922.

[17] Chin, S., Yoon, S., Choi, C., and Cho, C., "RFID+4D CAD for progress management of structural steel works in high-rise buildings," Journal of Computing in Civil Engineering, 2008, 22 (2): pp. 74-89.

[18] Bishop, C. M., Neural networks for pattern recognition, Oxford University Press, New York, NY., 1995.

[19] Swingler, K., Applying neural networks: a practical guide, Morgan Kaufmann Publishers, Inc., San Francisco, CA., 1996.

[20] Sonmez R., and Rowings J.E., "Construction labor productivity modeling with neural networks," Journal of Construction Engineering and Management, 1998, 124(6), pp. 498-504.

[21] AbouRizk, S., Knowles, P., Hermann, U. R.,. "Estimating labor production rates for industrial construction activities," Journal of Construction Engineering and Management, 2001, 127(6), pp. 502-11.

[22] Lu, M., Abourizk, S., Hermann, U. H., "Estimating labor productivity using probability inference neural network," Journal of Computing in Civil Engineering, 2000, 14(4):241-8.

[23] AbouRizk, S.M., and Sawhney, A., "Subjective and interactive duration estimation," Canadian Journal of Civil Engineering, 1993, 20(3), 457–470.

[24] AbouRizk S.M. and Hague S., "An overview of the COSYE environment for construction simulation," Proceedings of the winter simulation conference; Austin, Texas, USA, 2009.

[25] Kuhl, F., Weatherly, R., and Dahmann, J., Creating computer simulation systems: An introduction to the high level architecture, Prentice Hall, NJ, USA, 2000.

[26] The Institute of Electrical and Electronics Engineers, Inc., IEEE 1516 for Modeling and Simulation (M&S) High Level Architecture (HLA) –Framework and Rules, IEEE Standard, 2000.

[27] AbouRizk, S.M. and Hajjar, D., "A framework for applying simulation in construction," Canadian Journal of Civil Engineering, 1998, 25 (3), pp. 604-617.

[28] Alvanchi A., Azimi R., Lee S. and AbouRizk S. "Virtual model of structural steel construction using COSYE Framework." 10th International Conference on Construction Applications of Virtual Reality 2010, Sendai, Miyagi, Japan, 2010, pp. 283-290.

[29] Song, L., AL-Battaineh, H., and AbouRizk, S. M., "Modeling uncertainty with an integrated construction simulation system." Canadian Journal of Civil Engineering., 2005, Vol. 32(3), 533-542.

[30] Brown, J.L., "Wisconsin bets on BIM," Civil Engineering, 2009, 79(9), pp. 40-42.

[31] <http://www.wardsystems.com/manuals/neuroshell2>, Accessed on 12/20/2010.

Towards Experience Management for Very Small Entities

Vincent Ribaud, Philippe Saliou
Département d'Informatique
Université de Bretagne Occidentale, UEB
20 avenue Le Gorgeu, CS 93837
29238 Brest Cedex, France
{Vincent.Ribaud, Philippe.Saliou}@univ-brest.fr

Claude Y. Laporte
Département de génie logiciel et des TI
École de technologie supérieure
1100 rue Notre-Dame Ouest
Montréal (Québec), Canada, H3C 1K3
Claude Y. Laporte@etsmtl.ca

***Abstract*—The ISO/IEC 29110 standard: Lifecycle profiles for Very Small Entities, provides several Process Reference Models applicable to the vast majority of very small entities (defined by the ISO as "an entity (enterprise, organization, department or project) having up to 25 people") which do not develop critical software and share typical situational factors. An ISO/IEC 29110 pilot project has been established between the Software Engineering group at Brest University and a 14-employee company with the aim of establishing an engineering discipline for a new Web-based project. As the project proceeded, it became apparent that setting up the ISO/IEC 29110 standard has to be performed in two steps: 1) provide self-training materials to the VSE employees on this new standard; and 2) support good practices with a simple Experience Management system which is compatible with the ISO/IEC 29110 standard. This paper reports the lessons learned about training from the pilot project, and addresses the research issues associated with the Experience Management system.**

Keywords: software engineering processes, experience management, very small entities, ISO standards.

I. INTRODUCTION

This paper is an extended and enhanced version of a paper presented at the ICSEA 2010 conference [1].

Very Small Entities (VSEs) are recognized as extremely important to the software economy, producing stand-alone or integrated software components for large software systems. While the use of Software Engineering standards may promote recognized and valuable engineering practices, these standards were not designed with the needs and expertise of VSEs in mind, and do not fit the characteristics of VSEs. They are consequently difficult to apply in these settings [2]. The term 'Very Small Entity' (VSE) was defined by ISO/IEC JTC1/SC7 Working Group 24 (WG24) as "*an entity (enterprise, organization, department or project) having up to 25 people*." This definition has subsequently been adopted by the ISO in their response to the specific needs of VSEs: the ISO/IEC 29110 standard, Lifecycle profiles for Very Small Entities [3]. The standard defines a group of Standardized Profiles. Profiles are subsets of appropriate elements of standards which are relevant to the VSE context; for example, processes and products of the main software engineering standards. The ISO/IEC 29110-4-1 Basic Profile [4] applies specifically to a VSE involved in the development of a single software application by a single project team with no special risk involved and no particular situational factors at play.

This paper reports some of the conclusions reached as a result of a pilot project the authors conducted with a 14-person VSE that builds and sells counting systems for tracking visits to public and private sites. Only 3 of the employees are software developers, however, and so the VSE asked for assistance with software processes – mainly managing requirements and establishing a disciplined test process. ISO/IEC 29110 was naturally chosen as the reference framework, and the aim of the pilot project was to set up, within the VSE, the part of the Basic Profile related to requirements.

A VSE claiming compliance with the ISO/IEC 29110-4-1 Basic Profile will implement and use all the profile elements, as identified in Clause 7 of the profile specification [3]. The profile elements concerning requirements are: Project Plan Execution (PM.2), and Project Assessment and Control (PM.3), producing the Change Request work product; and Software Requirements Analysis (SI.2), producing a work product Change Request and Requirement Specification.

These profile elements state what has to be done, but provide very little guidance on how to do it. For the latter, Deployment Packages (DP) are expected to be particularly helpful. A DP is a set of artifacts developed to facilitate the implementation of a set of practices of the ISO/IEC 29110 standard. We introduced the ISO/IEC 29110 materials related to requirements to the VSE, and began to coach a novice engineer on the use of these materials for managing requirements. As the pilot project proceeded, it became apparent that the ISO 29110 set of documents (including DPs) was not up to the task of sustaining this VSE in its engineering activities. We maintain in this paper that implementing standardized software engineering activities in a VSE requires specific operational materials and mechanisms. What we are proposing is to provide VSE employees with a Self-Training Package intended to help the engineer carry out these activities.

Because software engineers in a VSE use SE processes and produce SE products continuously in different projects, we expected that an Experience Management (EM) system tailored for a VSE would provide a way to relate and integrate those project experiences and be a significant help. We also maintain in this paper that an EM system for a VSE should be constructed on a framework suitable for that entity,

but derived from a standardized Process Reference Model (presented in part in Section III.C) taken from the ISO/IEC 29110 Basic Profile [4].

EM solutions to organizing knowledge can be supported by experience factories (EF) [5]. "*EM includes methods, techniques, and tools for identifying, collecting, documenting, packaging, storing, generalizing, reusing, adapting, and evaluating experience; and for the development, improvement, and execution of all knowledge-related processes*" [6]. EF is defined as "*an infrastructure designed to support experience management*" and "*supports the collection, preprocessing, and dissemination of experiences*" [6]. This paper outlines a simple knowledge management system intended to gather, link, and reuse knowledge about SE activities. Requirement Analysis and its associated work products will be used as an example.

Professional competency management focuses on the development of a professional attitude and skills. These components are usually addressed in a 'practicum' or in 'clinical work', and the concept of reflection, inspired by D. Schön's work [7], is central to this competency development. The knowledge management system was designed based on two main guiding principles: the extraction of knowledge of existing SE standards – providing the system with a bootstrap, and the building of new knowledge by the software engineers themselves – a process required to maintain accurate and 'living' knowledge.

The next section provides an overview of the ISO/IEC 29110 standard, EM and EF, and related work. The standard is discussed in section III, and a case study introduced focused on requirement analysis and test activities. In section IV, we present our work on EM for a VSE and discuss some facts related to the case study. We conclude the paper by briefly presenting a few perspectives.

II. REQUIREMENTS AND RELATED WORK

In this section, we present the ISO/IEC 29110 initiative, discuss about the Experience Factory, report on Argyris and Schön's theories of action, and overview D. Schön's reflection-on-action work. Related work is also discussed.

A. SE Standards for Very Small Entities

1) ISO terminology

A Base Standard is an approved International Standard or ITU-T Recommendation [8]. An International Standardized Profile (ISP) is a harmonized document on which there is international agreement, and which describes one or more profiles [8]. A Profile is a base standard or set of base standards and/or ISs, including, where applicable, the selected classes, conforming subsets, options, and parameters of those base standards, or ISPs, required to accomplish a particular function [8]. A Technical Report (TR) is developed like a standard, but its purpose is simply to provide technical information, rather than requirements on implementation. Also, TRs are available free of charge.

2) ISO initiative

SE standards and methods often neglect the needs and problems of small and medium-sized enterprises (SMEs), which constitute a major part of the software industry. In 2005, the ISO recognized the needs and problems of VSEs and established a Working Group (WG24) mandated to develop a set of standards and technical reports suitable for these entities. The resulting ISO/IEC 29110 standard constitutes a set of guidelines for use by VSEs. Those guidelines are based on subsets of appropriate elements of standards, referred to as VSE profiles [3], relevant to the VSE context; for example, processes and outcomes of ISO/IEC 12207 [9], and products of ISO/IEC 15289 [10].

The Generic Profile Group targets VSEs that do not develop critical software and that share typical situational factors. It is composed of 4 profiles: Entry, Basic, Intermediate, and Advanced. As mentioned in the introduction, the Basic Profile [4] applies to VSEs involved in the development of a single software application by a single project team with no special risk involved or situational factors at play. By design, it excludes many of the ISO/IEC 12207 processes.

The standard is composed of five parts. As specified in [3], Part 1 targets VSEs, assessors, standards producers, tool vendors, and methodology vendors. Part 3 targets assessors and VSEs, and Parts 2 and 4 target standards producers, tool vendors, and methodology vendors. Parts 2 and 4 are not intended for VSEs. Part 5 targets VSEs. If a new profile is needed, only Parts 4 and 5 can be developed without impacting existing documents, and they would become Part 4-x and Part 5-x-y respectively, through the ISO/IEC standardization process.

The simplest path for a VSE is to start with Part 5-1-2: Management and engineering guide: Generic profile group: Basic profile. Using the Guide, a VSE can benefit in the following ways [11]:

- An agreed set of project requirements and expected products is delivered to the customer;
- A disciplined management process, which provides project visibility and proposes corrective actions for project problems and deviations, is performed;
- A systematic software implementation process, which satisfies customer needs and ensures quality products, is followed.

3) Deployment Packages

Once ISO/IEC TR 29110-5-1-2 has been downloaded, at no cost, from the ISO website, a VSE may consider that the help provided in it is insufficient to guide the implementation. Deployment Packages (DPs), by contrast, can be expected to provide significant help, a DP being defined as "*a set of artifacts developed to facilitate the implementation of a set of practices, of the selected framework, in a VSE*" [12]. The elements of a typical DP are: process description (activities, inputs, outputs, and roles), a guide, a template, a checklist, an example, presentation material, references and mapping to standards and models, and a list of tools [12]. The mapping is given only as information to show that a DP has explicit links to standards, such as ISO/IEC 12207, or to models, such as the CMMI®. So, by deploying and implementing the package, a VSE can visualize the concrete steps required to achieve or demonstrate coverage. Packages are designed so that a VSE

can implement their content without having to implement the complete framework at the same time.

4) Pilot projects

Pilot projects are an important means for reducing risks and learning more about the organizational and technical issues associated with the deployment of SE practices. A successful pilot project is also an effective means for encouraging the adoption of new practices by members of a VSE [12]. DPs are intended to apply the ISO/IEC 29110 standard in a VSE. Tailoring software processes to a VSE constitutes a kind of process improvement. A pilot project may also be an initial implementation of a DP, which provides WG24 with feedback from the improvement proposals before the DP is adopted as a standard.

B. Experience Management

1) Knowldege and Experience

The main asset of a software company is its intellectual capital, and knowledge management (KM) aims to capitalize on that capital. Schneider has explained how knowledge can be encoded in different representations and stored in ontologies, and that the instances of an ontology make up a knowledge base which can be searched and used for reference purposes [13, p. 135]. Business issues of KM are related to decreasing time and cost while increasing quality, and to making better decisions [14]. But KM implementation requires investment, and is probably out of reach for a VSE. Experiences constitute a subset of knowledge, and the reuse of experiences is a variant of KM. Experience management (EM) is a lightweight KM approach, a possible implementation of which is presented in the next section.

2) Experience Factory

EM is aimed at improving project performance by leveraging experiences from previous projects. In order to achieve experience reuse, Basili et al. [15] have proposed an organizational framework that separates project-specific activities from reuse packaging activities, with process models to support each activity.

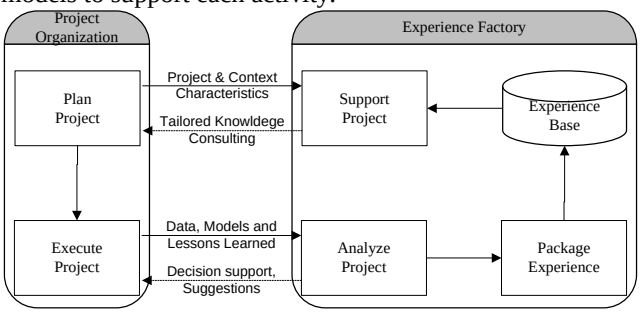


Figure 1. Experience factory (adapted from Ras et al. [6]).

The framework, represented in Figure 1, defines two separate organizations: a project organization, intended to deliver the system required by the customer, and an Experience Factory (EF), the role of which is to monitor and analyze project developments, to develop and package experience for reuse in the form of knowledge, processes, tools, and products, and to supply them to the project organization upon request [15]. "*The EF employs several methods to package the experience, including designing measures of various software process and product characteristics and then build[ing] models of these characteristics that describe their behavior in different contexts*" [16, p. 30]. A dedicated sub organization is required for learning, packaging, and storing experience.

Separating the project from the experience organization, physically or logically, may relieve project teams of the tasks required by EM, but is not, in our opinion, applicable in VSEs, or even in software SMEs (up to 250 employees). We agree with [17] that software SMEs need a more lightweight means of creating these knowledge bases with minimal overhead. Wiki-based repositories are often used as knowledge repositories, because the wiki concept easily integrates users into the knowledge-sharing process in SMEs [18]. Lightweight tools are useful, but knowledge transfer processes have to be built because the goal of EF is to build knowledge by learning from experience. We draft a learning theory in the next section.

C. Argyris and Schön's Theories

1) Theory of Action

According to Argyris and Schön, people design and guide their behavior by using theories of action. They suggest that there are two kinds of theory of action: a theory consistent with what people say, and a theory consistent with what people do. "*Espoused theories of action are the theories that people report are governing their actions. Theories-in-use are the theories of action that actually govern their actions*" [19, p. 7].

Argyris and Schön used three constructions to explain theories-in-use (see Figure 2 for a more comprehensive explanation). *Governing variables* are values that a person is trying to keep within a preferred range (e.g. a manageable workload). *Action strategies* are strategies used to maintain the governing variables within the accepted limits (e.g. refusal to accept extra work). These strategies will have *consequences* which are either intended (e.g. the amount of work does not increase too much) or unintended (e.g. the amount of work is decreasing too drastically).

When there is a mismatch between intended consequences and outcomes, the situation has to be corrected. Argyris states: "*An organization may be said to learn to the extent that it identifies and corrects errors*" [19, p. 4]. They suggest that the first response to this mismatch is to select another action strategy that will still satisfy the governing variables (e.g. accept extra work, but delay providing the result). Such a process of changing the action strategy only, and not the governing variables themselves is called *single-loop learning*. Another possibility is to examine and modify the governing variables (e.g. accept too great a workload in order to reach a new position). In this case, both the governing variables and the action strategy have to be modified, and this is called *double-loop learning*.

Argyris and Schön argue that, although espoused theories vary widely, theories-in-use do not. They labeled the most prevalent theory-in-use 'Model I'. "*Model I theories-in-use are theories of top-down, unilateral control of others for the actors to win, not to lose, and to control the environment in*

which they exist to be effective" [19, p. 7]. They argue that, with such a theory-in-use, problem solving works for issues that do not require that the underlying assumptions of Model I be questioned (*single-loop learning*). Model II theories-in-use make it possible for people "*to have problem-solving skills that question the governing values of their theory-in-use*" [19, p. 7] (*double-loop learning*).

Models of theory-in-use

Model I and Model II theories-in-use consider three elements: (1) *governing variables*, which are values that actors seek to maintain [1], each of which can be thought of as a continuum with a preferred range (e.g. not too concerned, but not too indifferent either); (2) *action strategies*, which are sequences of moves and plans adopted by actors in particular situations to satisfy governing variables [1] to keep those variables within the preferred range (e.g. undertaking physical exercise to reduce stress); and (3) the *consequences* that follow as a result of action, which can be intended – those that the actor believes will satisfy the governing variables (e.g. feeling better after engaging in a sport), or unintended, both types designed to be dependent on the theories-in-use of the recipients, as well as those of the actors.

Single and double-loop learning

When the consequences of an action strategy are what the actor wanted, then that person's theory-in-use is confirmed. If there is a mismatch between intention and outcome, the consequences are unintended. Argyris defines learning as the detection and correction of error. The first response to error is to search for another action strategy. "*Single-loop learning occurs when errors are corrected without altering the underlying governing variables*" [2, p. 206]. An alternative is to question the governing variables themselves, to subject them to critical scrutiny (e.g. to openly investigate the anxiety, rather than trying to suppress it). "*Double-loop learning occurs when errors are corrected by changing the governing variables and then the actions*" [2, p. 206]. Argyris and Schön argue that many people espouse double-loop learning, but are unaware of it, much less able to produce it.

Model I and Model II

Briefly, Model I is composed of four governing variables: (1) achieve the purpose as defined by the actor; (2) win, not lose; (3) suppress negative feelings; and (4) emphasize rationality [1]. The primary behavioral strategies are to control the relevant environment and tasks unilaterally, and to protect oneself and others unilaterally. Thus, the most widely used action strategy is unilateral control over others. Characteristic ways of implementing this strategy are: to make non illustrated attributions and evaluations (e.g. "Your work is poor."); to advocate courses of action in ways that discourage inquiry (e.g. "Surprise me, but don't take risks."); and to treat one's own views as obviously correct, leaving potentially embarrassing facts unstated [1]. The consequences are likely to be defensiveness, misunderstanding, and self-fulfilling processes [2]. Model I leads to low-level learning, and double-loop learning tends not to occur. Argyris and Schön aim to move people from a Model I theory-in-use to a Model II theory-in-use that fosters double-loop learning.

The governing variables of Model II include: (1) valid information; (2) free and informed choice; and (3) commitment: vigilant monitoring of the implementation choice to detect and correct errors [2]. The behavioral strategies involve sharing control with those who have the competence to do so and who participate in designing or implementing the action [1]. As in Model I, prominent behaviors are advocated, evaluated, and attributed. Unlike Model I behaviors, Model II behaviors stem from action strategies where attributions and evaluations are illustrated with observable data, and the surfacing of conflicting views is encouraged so that they can be publicly tested. The consequences include minimally defensive interpersonal and group relationships, great freedom of choice, and a high level of risk-taking. Defensive routines are minimized and genuine learning is facilitated [1, 2].

References

[1] C. Argyris, R. Putnam, and D. McLain Smith, "Action Science: Concepts, Methods, and Skills for Research and Intervention," San Francisco: Jossey-Bass, 1985, pp. 4, 80.

[2] C. Argyris, "Double-Loop Learning, Teaching and Research," Learning & Education, vol. 1 (2), Dec. 2002, pp. 206-219

Figure 2. Theory of Action, according to Chris Argyris and Donald Schön.

2) The reflective practitioner

Schön's "reflective practitioner" perspective [7, p. 20] guides creative professionals to reflect about their creations during (*reflection-in-action*) and after (*reflection-on-action*), thereby completing the creative process. A specialist is a professional practitioner who is used to dealing with certain types of situations again and again. Practitioners build up a collection of ideas, examples, situations, and actions which Schön calls a "repertoire". "*A practitioner's repertoire includes the whole of his experience insofar as it is accessible to him for understanding and action*" [7, p. 138].

A practitioner develops a repertoire of expectations, images, and techniques. As long as her/his practice continues to present the same types of cases, s/he becomes less and less susceptible to surprise [5, p. 60]. But, when a new situation is stimulating enough, the reflective practitioner is surprised. Schön argues that these experienced professionals deal with the 'messiness' of practice not only by consulting the research knowledge base, but by engaging in what he calls "reflection-in-action" [20], which is sometimes described as 'thinking on our feet'.

In many EF implementations, effort is put into analyzing and packaging experiences from raw experiences. But further effort is required to change the way that the whole organization performs its work. "*An organization adopting the EF approach must believe that exploiting prior experience is the best way to solve problems and ensure that the development process incorporates seeking and using this experience*" [16, p. 30]. A parallel can be drawn between an organization using an EF and a reflective practitioner. Raw and packaged experiences play the role of the practitioner's repertoire. As long as his/her practice remains stable, the practitioner relies on her/his tacit knowing-in-action, which is built on previous experiences directly. An organization building new knowledge while analyzing and packaging raw experiences is similar to Schön's reflection-on-action. "*Practitioners do reflect on their knowing-in-practice […], they think back on a project they have undertaken […], and they explore the understandings they have brought to their handling of the case. They may do this in a mood of ideal speculation, or in a deliberate effort to prepare themselves for future cases*" [7, p. 61]. An organization using the EF assumes that the EF and the Project Organization are integrated. "*The activities by which the Experience Factory extracts experience and then provides it to projects are well integrated into the activities by which the Project Organization performs its function*" [16, p. 31]. It assumes that the Project Organization makes no special effort to reuse packaged experiences.

But a practitioner may also reflect on a practice while s/he is performing it (Schön's reflection-in-action). In this case, the possible objects of this reflection are varied. "*He may reflect on the tacit norms and appreciations which underlie a judgment, or on the strategies and theories implicit in a pattern of behavior. He may reflect […] on the way in which he has framed the problem he is trying to solve*" [7, p. 62]. Of course, anyone can encounter a situation where a rule drawn from previous experience cannot be applied, in which case he/she has to be engaged in what

Schön calls a "practitioner's reflective conversation" with the materials related to the situation. Occasions will also arise when none of the packaged experiences will help the Project Organization using EF. Although Schön did not explicitly establish links between the reflection concepts and the nature of organizational learning presented in Figure 2, we consider reflection-in-action as a kind of double-loop learning, and we assume that developing a reflective practice will favor that type of learning.

D. Application to Very Small Projects

The set of ISO/IEC 29110 documents establishes what has to be done in a software project, and will be presented in section III.A. Little help is provided to explain the procedure, however, although pilot projects are carried out expressly intended to put this standard into practice which can be considered as an experience packaging activity. Section III.B and section III.C provide an insightful presentation of a pilot project on requirements.

The main deliverables of a completed SME project are stored by the project manager in an Experience Repository, according to a fixed storage scheme. At best, the Experience Repository of a VSE contains only raw experiences. Data, models, and deliverables collected on previous projects are stored as is, without any structure. The initial benefit for a VSE of using the ISO/IEC 29110 standard is that a common Process Reference Model (PRM) will be shared between projects, as the structure of the PRM may help to structure the Experience Repository. Since the most common knowledge pattern transfer is the copy-paste model, a shared structure will favor reuse of raw experiences. Section IV.A presents an Experience Repository for raw data, and section IV.B.1 presents the copy-paste activities that are using it.

We believe that there can be a considerable gap between the structure of an engineer's repertoire (and hence the project organization that he or she may use) and the structure of the EF. Extracted knowledge facilitates experience reuse and learning. In Figure 1, adapted from [6], arrows from left (Project Organization) to right (Experience Factory) indicate knowledge extraction. Knowledge transfer is a double-loop learning activity. Lessons learned from the pilot project indicate that it is a difficult issue for engineers to cope with, especially novice engineers. Our challenge was to find a way to encourage reflection-in-action and develop double-loop learning. Sections IV.3.2 and IV.3.3 present the practical solution that we provided to the VSE.

Ras et al. [21] address this problem with an approach called 'learning space generation', which enriches experience packages with additional information from specifications provided either by the instructor or by the student. The learning space is presented by means of Wiki pages within a specialized Wiki based on the Software Organization Platform (SOP). Our approach does the opposite. Rather than providing engineers with access to the experience packages, we essentially provide task description exemplars and product exemplars created in small projects.

III. A STANDARDIZED PROCESS PROFILE FOR VSES

In this section, we present the context of the pilot project, the expectations of the VSE, and the application of the ISO/IEC 29110 standard to this project.

A. Implementation of Standardized Processes

At the core of the ISO/IEC 29110 standard is a Management and engineering guide (ISO/IEC 29110-5) [11], which focuses on Project Management and Software Implementation, and an Assessment Guide (ISO/IEC TR 29110-3) [22]. ISO/IEC 29110-5 provides a practical guide to the ISO/IEC 29110-4-1 standard [4], identified as a Basic Profile of the Generic profile group. For instance, the starting point for ISO/IEC 29110 use for requirements is the SI.2 Software Requirements Analysis activity, its list of tasks (SI.2.1 to SI.2.7), and the associated roles.

Deployment Packages (DP) provide VSEs with assistance in adopting standards through a DP Repository http://profs.logti.etsmtl.ca/claporte/English/VSE/index.html. For instance, DP Software Requirement Analysis [23] simplifies task decomposition and provides a step-by-step method for each task.

B. Pilot Project

1) Requirements

Fenton et al. state in [24]: "*For 25 years, software researchers have proposed improving software development and maintenance with new practices whose effectiveness is rarely, if ever, backed up by hard evidence.*" They suggest several ways to address this problem, in particular careful design and measurement experiments, such as pilot projects.

2) Context of the VSE

A VSE with a staff of 14 (3 of them software engineers) requested the help of our SE group in the spring of 2009. The VSE designs, builds, develops, and sells counting systems designed to collect and analyze data on visits to public or private sites. Initially intended for counting pedestrians, this VSE's products now cover bikes, horses, and cars. Counting systems are based on stand-alone counter boxes (including sensors, a power supply, data storage, and data exchange) and a software chain capable of collecting, analyzing, presenting, and reporting counting data. In the previous software chain, the set of data was downloaded from counters by infrared link or GSM, stored on personal computers, and then transmitted via a file transfer utility.

3) The new software project

Because of its clients' requirements and the products supplied by the competition, the VSE began a complete reconstruction of its software chain in order to transform it into a Web-based system, called Eco-Visio, intended to host the data of fleets of counting systems for each client, and capable of processing statistics and generating analytical reports on counting. At the end of June 2009, the VSE hired a graduate of Brest University, who had done his final internship at the VSE. At the same time, we visited the VSE and initiated a pilot project with the intention of transferring a part of the ISO/IEC 29110 standard to the specific context of the VSE. Project stakeholders decided to focus on two SE

activities: 1) the establishment of a practical technique for gathering and managing requirements; and 2) improvement of the system's reliability with a disciplined test process.

The new software project, completed at the end of March 2010, was released as the first version of the new Eco-Visio Web-based system.

C. Basic Profile

1) Basic Profile processes

The Generic Profile Group [4] is a collection of four profiles (Entry, Basic Intermediate, Advanced), providing a progressive approach to satisfying the needs of a vast majority of VSEs that do not develop critical software and share characteristic situational features. The Basic Profile applies to a VSE that is involved in software development of a single application by a single project team involving no special risk or situational factors. The objective of the project is to fulfill an external or internal contract. The internal contract between the project team and their client need not be explicit.

The Basic Profile is made up of two processes: Project Management (PM) and Software Implementation (SI). A process is defined as "*a set of interrelated or interacting activities which transforms inputs into outputs*" [9]. Table I provides the process/activity breakdown, and presents tasks related to requirements and tests (which are the focus of the pilot project cited above).

TABLE I. BASIC PROFILE PROCESS BREAKDOWN

Process	Activities	Pilot project-related tasks
PM Project Management	PM.1 Project Planning	PM.1.1
	PM.2 Project Plan Execution	PM.1.13, PM.1.14
	PM.3 Project Assessment and Control	PM.2.2 and PM.2.4
	PM.4 Project Closure	PM.3.5
SI Software Implementation	SI.1 SW Implementation Initiation	-
	SI.2 SW Requirements Analysis	SI.2.2, SI.2.3, SI.2.4
	SI.3 SW Architectural and Detailed Design	SI.3.5, SI.3.6
	SI.4 SW Construction	SI.4.4
	SI.5 SW Integration and Tests	SI.5.4
	SI.6 Product Delivery	-

ISO/IEC TR 29110-5-1-2 [11] is intended to guide the Basic Profile implementation of PM and SI processes described in ISO/IEC 29110-4-1 [4]. These processes integrate practices based on the selection of ISO/IEC 12207 SW life cycle processes and ISO/IEC 15289 information product (documentation) standards elements. DPs will facilitate the implementation of these processes.

2) Basic Profile products

Clause 9 of ISO/IEC 29110-4-1 [4] establishes the normative list of Basic Profile work product and deliverable specifications. There are 23 work products, which can be the input, output, or internal products of processes, activities, or tasks.

3) Process assessment

ISO TR 29110-3 [22] is an Assessment Guide applicable to all VSE profiles. It is compatible with ISO/IEC 15504-2 and ISO/IEC 15504-3. The assessment has two purposes: 1) to evaluate process capability based on a two-dimensional assessment model (from the ISO/IEC 15504:2006 standard [25]); and 2) to determine whether or not an organization achieves the targeted VSE Profile based on the process capabilities evaluated [22]. A VSE-specific Process Assessment Model (PAM) can be derived by selecting only a set of assessment indicators from ISO/IEC 15504-5: "*an Exemplar PAM"* We selected the assessment indicators relevant to the corresponding process outcomes, as defined in ISO/IEC 29110-4-1.

4) Performing the ISO/IEC 29110 Requirements Analysis

ISO/IEC 29110-4-1 provides a set of cohesive tasks for each activity. Also established here are the VSE needs and suggested competencies. For instance, it sets out the SI.O2 objective: "*Software requirements are defined, analyzed for correctness and testability, approved by the Customer, baselined and communicated. Changes to them are evaluated for cost, schedule and technical impact previously to be processed*" [4, p. 8].

ISO/IEC TR 29110-5-1-2 details the tasks to be performed for each PM and SI process activity: role, description of the task, and input and output products. For instance, it defines tasks SI.2.1 to SI.2.7, detailed in Table II, and their associated output products: *Requirements Specification*, *Verification Results*, *Change Request*, *Validation Results*, and *Software User Documentation*.

The Software Requirements Analysis DP [23] simplifies task decomposition: requirement identification, requirement refinement and analysis, requirement verification and validation, and requirement change management. A step-by-step method is described for each of these four tasks. The DP also provides a Software Requirement Specification template. Training materials and an Excel-based traceability tool can be downloaded from the publicly accessible WG24 website.

The pilot project was intended to provide coaching for the implementation of the Software Requirements Analysis DP. One VSE novice engineer studied the DP and was given a short training course, using the training material associated with this DP. Despite all this helpful material, the VSE engineer was not able to start the Software Requirements Analysis activity, suffering from 'blank page' syndrome. The authors could not provide strong support to the VSE, and we have had to reorientate the pilot project.

5) Problem analysis

Based on feedback that the novice engineer was not able to perform the SI.2 Software Requirements Analysis activity, the authors set about to analyze the problem.

As we stated in section III.B, action theory studies what an actor does in a given situation in order to achieve objectives. Argyris and Schön [26] made a distinction between *espoused* theories, which are those that an individual claims to follow, and *theories-in-use*, which are those that can be inferred from action. Espoused theory and

theory-in-use may be contradictory, and the agent may or may not be aware of any inconsistency. By definition, however, the agent is aware of espoused theory, and theories-in-use can be made explicit by reflection-on-action [27].

Software companies use SE and software quality standards as the foundation of their quality assurance process and quality management system. Since these companies claim to follow and respect standards, we may think that these standards constitute a part of the espoused theories of software engineers, especially Process Assessment and Process Reference Models. In the software field, we observe that a software engineer may have a work behavior – her/his theories-in-use – which often runs counter to the organizations' processes, practices, and procedures that she/he is supposed to follow and talk about, i.e. espoused theories.

What happened to that young engineer? Through the standard documentation and DPs, he received a great deal of information on espoused theory. However, as his repertoire of experience (and VSE Experience Repository) was all but empty, he could not act in accordance with any theory-in-use.

An Experience Repository may act as a product and project memory. It records footprints of the organization's theories-in-use and provides support for learning from past experience. Thus, managing experience in a repository may provide VSE engineers with a simple form of knowledge management. But, as we will see in the next section, an Experience Repository requires additional processes in order to support knowledge transfer.

IV. EXPERIENCE MANAGEMENT FOR THE VSE

Chan and Chao present a research survey conducted among 68 SMEs which have implemented Knowledge Management (KM) initiatives [28]. SMEs are significantly bigger than the targeted VSEs, but the lessons learned in this survey also apply to VSEs. Effective KM is influenced by two types of KM capability: infrastructure and process, which have to be deployed. This section presents a simple Content Management System-based infrastructure to manage experience and some activities that may be part of EM processes.

A. An Experience Repository

1) Related work

A significant part of EM in a software company should be about software documentation reuse (code reuse is outside the scope of this paper). So, the primary inputs of our system are documentation deliverables: plans, requirements, design specifications, data schemas, test cases, and so forth. Publishing and content management systems (CMS) are generally used as the basis for a documentation management infrastructure. But several authors have criticized the rigidity of the editorial control required by a CMS [29] and the need to balance structure/constraint and flexibility [30]. Some promote the use of Wikis and RDFs (Resource Description Frameworks) to resolve these issues [31].

Wikis are probably a suitable tool for facilitating collaborative design and development, and may be viewed as part of the project repository (see Figure 1), but requirements for an EM infrastructure are different. Rech et al. identified several challenges related to knowledge transfer and management processes for SMEs in the software sector: recording, reusing, locating, and sharing information [18]. The authors evaluated a small software enterprise and a micro software enterprise with reuse policies in place. They point out that the engineers have little confidence in knowledge transfer, because only a few reusable documents have been created. They also note that the workflow for reusing knowledge is slow and typically demotivating, because multiple sources have to be searched manually and documents belonging together weren't grouped together or linked [18]. As we will see in the next section, a CMS-based system with a simple and fixed structure may resolve most of these issues.

2) Experience Repository infrastructure

According to Peter Senge [32], what he calls "personal mastery models" and "mental models" are two of the five disciplines that distinguish a learning organization from more traditional organizations. The questions to be answered are: how do experts learn compared to novice practitioners, and how do their mental models differ? "*People with a high level of personal mastery live in a continual learning mode*" [32, p. 142]. Mental models are "*deeply ingrained assumptions, generalizations, or even pictures and images that influence how we understand the world and how we take action*" [32, p. 8]. They are similar to Schön's professional's repertoire. "*From a cognitive point of view, there is a quantitative difference between expert and novice knowledge bases and also a qualitative difference, e.g. the way in which knowledge is organized. Novices lack background knowledge and are not able to connect their experience to their knowledge base. The organization of knowledge at the experience provider's and at the experience consumer's makes the transfer of knowledge between different levels of expertise extremely difficult*" [33]. Part of the problem can be avoided if experts and novices share a common repertoire structure. We use the ISO/IEC 29110 Basic Profile Process Breakdown (see Table I) as the shared structure of the Experience Repository. We discuss later how learning processes should be developed in order to support knowledge transfer.

Managing an Experience Repository for a small project can be greatly facilitated if the structure is kept as simple as possible, which means we should also avoid amassing too many artifacts. Our proposal is that, whenever a project is completed, the project closure activity create its own space in the CMS and use the Process/Activity decomposition of ISO/IEC 29110-4-1 [4, Clause 7] as the structure for that space. Then, only the main deliverables of the project, as defined in ISO/IEC TR 29110-5-1-2 [11, Clause 4], will be stored, and in the right place in the structure.

Table II shows the structure and content of the Experience Repository for some representative activities of each process. To further illustrate our work, we added the activity-related tasks. The left-hand column provides links to

the ISO/IEC 12207 activities profiled in the parts of ISO/IEC 29110 mentioned. We added (in italics) a proposal (published in [34]) for the management of deliverables related to support activities.

The infrastructure is not intended to be a project repository (the left part of Figure 1) hosting project deliverables in different versions. The infrastructure forms part of an Experience Repository intended to record the final state of the project and to provide further projects with exemplars of deliverables. The Alfresco content platform (http://www.alfresco.com/products/wcm) is used as a Web content management suite, mainly for providing an upload-download system organized into a hierarchy of space, with the possibility of a fine-grained control of users' rights over spaces. As mentioned above, the space hierarchy structure is, for each project, the Process/Activity decomposition of ISO/IEC 29110-4-1 [4]. Each space hosts a variety of work products of ISO/IEC TR 29110-5-1-2 [11]. Examples of these products are given in Table II, column 4.

TABLE II. STRUCTURE AND CONTENT OF THE EXPERIENCE REPOSITORY

12207	Activity	Tasks	Output products
Project Management Process			
6.3.1.3.3, 6.3.2.3.1, 6.3.2.3.2	PM.2 Project Plan Execution	• PM.2.1 Review Project Plan • PM.2.2 Change request analysis • PM.2.3 External revision meeting • PM.2.4 Internal revision meeting	Project Plan Change Request Meeting Record
…			
Software Implementation Process			
6.4.1.3.1, 6.4.1.3.2, 6.4.1.3.3, 6.4.1.3.4, 6.4.1.3.5, 7.1.2.3.1	SI.2 SW Require-ments Analysis	• SI.2.2 Document requirements • SI.2.3 and 2.4 V & V requirements	Requirements Specifications V&V Results
7.1.3.3.1, 7.1.4.3.1	SI.3 SW architectural and detailed design	• SI.3.3 Document software design • SI.3.4 Software design verification	Software Design Traceability Record Verification Results
…			
Management and Implementation Support Process			
6.2.1.3.1, 6.2.1.3.3	*Method and tool support*	• *Process establishment* • *Process improvement* • *Tool support*	*Process implementation recommendations* *Tool usage guide*

B. KM Support Processes

According to [35], knowledge can be created through dedicated acquisition, conversion, application, and protection of knowledge assets. In the survey of 68 SMEs by Chan and Chao [28], most of the respondents stated that they encounter knowledge capture problems related to time, place, and people. But Conradi maintains that the hard part is not the "upward externalizing direction", but the "downward, internalizing flow" [36]. The problems that arise are very similar to those encountered in software engineering reuse. The literature agrees that understanding is a high cost factor for reuse. Dusink and Van Katwijk state: "*Essential for a higher degree of reuse is the reusing engineer's understanding of the reusable artifacts, the process, and the actions to be taken*" [37]. Ras states that general reuse education and technology training are the two principal solutions proposed to enhance reuse with respect to understanding [33].

The standard method for transferring knowledge from experts to novices is the copy-paste model, and the VSE asked for a similar pattern, which is for the Experience Repository to work based on this model, and for us to seed the Experience Repository by providing them with suitable examples, such as a Software Requirements Specification, that they can reproduce as closely as possible. The VSE asked for immediate working solutions and did not want to invest in understanding the experiences stored. However, we decided to provide the VSE with two levels of Experience Management Process: a copy-paste level, which is presented in this section, and a continuous understanding level, as discussed in section IV.C.

1) Copy-paste activities

TABLE III. SI.2 SOFTWARE REQUIREMENTS ANALYSIS -- TASKS AND ROLES. THE ASTERISK MEANS 'IF APPROPRIATE'.

Task List	Role
SI.2.1 Assign tasks to the Work Team members in accordance with their role, based on the current *Project Plan*.	Technical Leader, Work Team
SI.2.2 Document or update the *Requirements Specification*.	ANalyst, CUStomer
SI.2.3 Verify the *Requirements Specification*.	AN
SI.2.4 Validate the *Requirements Specification*	CUS, AN
SI.2.5 Document the preliminary version of the *Software User Documentation* or update the present manual.*	AN
SI.2.6 Verify the *Software User Documentation*	AN
SI.2.7 Incorporate the *Requirements Specification*, and *Software User Documentation** to the *Software Configuration* in the baseline.	TL

The copy-paste process is designed to be as simple as possible. Clauses 4.2.8 and 4.3.8 of ISO/IEC TR 29110-5-1-2 [11] propose task decomposition of the PM and SI processes for each activity (Table III presents the decomposition for SI.2 Software Requirements Analysis), together with inputs and outputs of each task. So, we can establish the workflow for each of the 23 work products (cf. §III.C.2). For instance, Figure 2 presents the workflow of Work Product 11, *Requirements Specification*.

Figure 3. WP11 *Requirements Specification* workflow

It may happen that a work product workflow spans several activities of the same process, such as WP17 *Software User Documentation*, which covers SI.2 to SI.5,

and even PM and SI processes, such as WP8 *Project Plan*, which covers PM.1 to SI.6.

The VSE needs a simple model to locate, store, and retrieve work products, according to the Process Reference Model used. Our proposal is to locate a work product inside the CMS space associated with the last activity that outputs the final version of this work product. So, WP11 *Requirements Specification* will be located in the 'SI.2 SW Requirements Analysis' space, WP17 *Software User Documentation* will be located in the 'SI.5 SW Integration and Tests' space, and WP8 *Project Plan* will be located in the 'SI.6 Product Delivery' space.

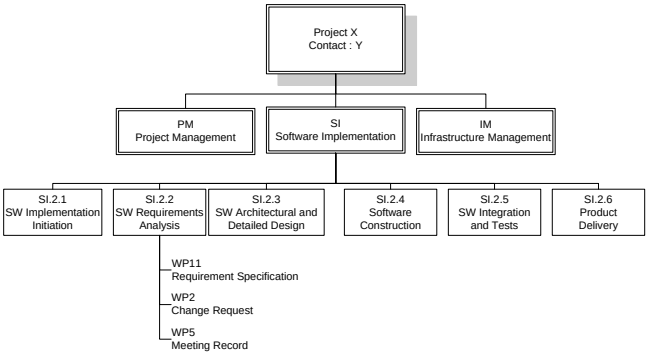


Figure 4. Structure and content of a project in the Experience Repository.

The task of storing work products in an Experience Repository space associated with the project will be allocated to the PM.4 Project Closure activity, which is the responsibility of the Project Manager (PM) role. With this simple copy-paste EM process, the PM role stores artifacts, and (ideally) every VSE employee can access and copy the artifacts of his/her choice. An extract from the structure and content of a project space is given in Figure 4.

2) Learning software engineering processes

A common assumption in software process improvement is that "*the quality of a software product is largely governed by the quality of the process used to develop and maintain it*" [38, p. 8]. Based on this assumption, Conradi states: "*This often means that relevant work practices (processes) must be systematically documented as formal routines, often as standard process models. These routines must then be communicated to the developers, customized and adopted by them and later revised based on experience and overall strategies*" [39, p. 268].

The ISO/IEC 29110 standard provides a Process Reference Model and DPs aimed at guiding the implementation of this model. We were not confident in the ability of novice engineers to understand the work practices and associated DPs documented in the ISO/IEC 29110 standard. So, we scheduled a training week on ISO/IEC 29110 Software Requirements Analysis in December 2009. Ten novice engineers (including the VSE engineer) attended the session, which comprised a course on requirements and a case study using the Software Requirement Analysis DP [23]. This DP is summarized in Figure 5.

Task 1. Requirements identification. The objective is to clearly define the scope of the project and identify key requirements of the system. Steps are: (i) Collect information about the application domain; (ii) Identify project scope; (iii) Identify and capture requirements; (iv) Structure and prioritize requirements.

Task 2. Requirements refinement and analysis. The objective is to detail and analyze all the requirements identified. Steps are: (i) Detail requirements; (ii) Produce a prototype.

Task 3. Requirements verification & validation. The objective is to verify requirements and obtain validation from the customer or his representative. Steps are: (i) Clarify fuzzy requirements (verification); (ii) Review SRS (Software Requirements Specification); and (iii) Validate requirements.

Task 4. Requirements change management. The objective is to manage requirements change in line with a process agreed upon with the customer. Steps are: (i) Track changes to requirements; (ii) Analyze the impact of changes; (iii) Identify changes that are beyond the project scope; (iv) and Prioritize changes.

Figure 5. Step-by-step path proposed by the DP Requirement Analysis.

The session began with an introductory lecture on requirements, but trainees were quickly plunged into action with the preparation of a peer review on a requirements analysis guide. This guide was issued by a major ISO 9001 software company (at which both authors had been employed for about 10 years). The SW Requirements Specification (SRS) Document was issued by the DOD-STD-2167A software development standards [40]. This guide is intended to facilitate the writing of the SRS. Peer review of this guide provided trainees with their initial exposure to standardized requirements management.

During the second phase of the session, trainees had to contribute to writing a similar guide, based only on the ISO/IEC 29110 standard. The authors provided trainees with a preliminary version of the guide, written in a top-down manner, starting with the ISO/IEC 12207 standard processes devoted to requirements (6.4.1 Stakeholder Requirements Definition, 7.1.2 SW Requirements Analysis) and finishing with the ISO/IEC 29110 Basic Profile SI.2 Software Requirements Analysis activity. Trainees had to incorporate both the Software Requirement Analysis DP and its step-by-step approach into the guide.

Finally, trainees had to apply the enhanced guide to a 'real' SRS and update this SRS to comply with the guide requirements. That SRS is for eCompas – an existing system developed by the second author and former graduate students.

C. Understanding Experience

1) Learning from experience

Despite the path traced in the standard during the training session), some young engineers (and this is true of the VSE engineer in particular) reported being unable to find their way through managing the requirements.

As mentioned above, the ISO/IEC 29110 standard attempts to document the best practices as formal routines. Several authors have studied the gap between the rationalistic, linear model of software engineering and the reality for most small software organizations. Conradi and Dyba carried out a study in the context of a national software process improvement program in Norway for SMEs to assess the attitude to formalized knowledge and experience sources. They found that "*developers are rather skeptical at using written routines, while quality and technical managers are taking this for granted*" [39]. Dyba [41] states that "*a specific challenge involves balancing the refinement of the existing skill base with the experimentation of new ideas to find alternatives that improve on old ideas.*" But our hypothesis is more straightforward: for many novice engineers, the copy-paste model is not sufficient as a knowledge transfer pattern, because they have no previous experience to help them understand the formalized knowledge. The training session provided novice engineers with products resulting from past experience and with the assistance of teachers. We built the content session using a normative curriculum of a professional school, as attributed to Edgar Schein in [42]: "*First teach them the relevant basic science, then teach them the relevant applied science, then give them a practicum in which to practice applying that science to the problems of everyday life.*" The normative curriculum reflects an objectivist view of professional education, often portrayed as the opposite of a constructivist view. "*Objectivist conceptions of learning assume that knowledge can be transferred from teachers or transmitted by technologies and acquired by learners. […] Constructivist conceptions of learning, on the other hand, assume that knowledge is individually constructed and socially co constructed by learners based on their interpretations of experiences in the world*" [43, p. 217]. We do not oppose the notions of objectivism and constructivism. Rather, we believe that they offer different points of view which may be combined to favor learning. An important step in the learning cycle is the activity of reflection. If we provide learners with details of past experience acquired by other people, we have to find a way to help learners reflect on that experience. By reflecting on the experience acquired (by her/himself or others), learners integrate the lessons learned from that experience into their own knowledge structures.

2) Reflection-on-action and reflection-in-action

To meet the challenges of their work, practitioners rely on their repertoire of experience, along with a certain ingenuity acquired during that practice, rather than on knowledge-oriented curricula or formulae learned during their basic education. D. Schön describes this repertoire as follows: "*The practitioner has built up a repertoire of ideas, examples, situations and actions. […] When a practitioner makes sense of a situation he perceives to be unique, he sees it as something already present in his repertoire. To see this site as that one is not to subsume the first under a familiar category or rule. It is […] to see the unfamiliar, unique situation as both similar to and different from the familiar one, without at first being able to say similar or different with respect to what. The familiar situation functions as a precedent, or a metaphor, or […] an exemplar for the unfamiliar one*" [7, p. 138].

In order to help VSE employees understand the VSE Experience Repository, and consequently add to their own repertoire, we have designed practices that may help software engineers become 'reflective' practitioners. These practices are generally borrowed from two streams: industrial – process improvement and product assessment – and Schön's theory of reflection-on-action and reflection-in-action. For instance, bootstrapping an engineer's repertoire for a given activity in SE (e.g. requirements analysis or design) may require an approach based on tailoring an activity before the activity itself is performed. This approach has been presented through the specific case of the design in [44].

Such an approach is generally implemented in two steps: 1) tailoring the activity to acquire a minimal structure of the repertoire through a deductive approach (by writing a guide, for instance); and 2) initializing the repertoire through an inductive approach, with the use of retroengineering, for instance. This approach is a pragmatic answer to the lack of support and training that may be experienced in small projects, where the main effort is concentrated on project management and software development tasks.

3) Self-Training Packages

As reflective practices are performed by the learners themselves, very few interactions with a coach are required. The next step is related to organizing the self-learning process. Our proposal is to organize the engineer's training path through small units of work, called 'self-training tasks'. The description of the task is designed as a theater scene: the scene is the reference context where action takes place; it aims to maintain unity of place, time, and action, and is a site where a situation can occur and where people perform actions (and learn). It also serves as a location for action scenarios, for role distribution, and for mobilizing resources and means. The various components of a scene, along with their linkages, are depicted on a self-training report card. The card structure is standardized:

- Related ISO/IEC 29110 Process/Activity

This reference (for instance, SI/SI.2 SW Requirements Analysis) provides a smooth link to ISO/IEC 29110, and through the profile to ISO/IEC 12207 and ISO/IEC 15504.

- Role

The role (for instance, Analyst) is a brief reference to ISO/IEC 29110.

- Task title and objectives

These are similar to Process Title, Process Purpose, and Process Outcomes, as defined in ISO/IEC 12207.

- Step-by-step guide

This is a comprehensive description of the work to be done, intended to be a practical guide to completing the task.

- Resources

This is the set of required resources. It may include the hosting of technical support (such as Oracle Metalink) that a technology transfer center is able to afford when the cost is out of reach for a VSE.

- Output products

These are generally a methodological survey, a tool usage guide, or an installation manual.

The set of self-training activities that a VSE engineer should perform is incorporated into a Training Package (TP) (analogous to the ISO/IEC 29110 Deployment Package, or DP). Developing the concept of the TP is outside the scope of this paper, but suffice to say that a TP is primarily intended to provide self-training on SE activities, with the supplementary goal of initiating and developing a strategy of capitalizing on this knowledge and transferring it to VSE employees.

4) Empirical evaluation

The VSE engineer in question was provided with two TPs on Requirements at the end of 2009. The first was intended to provide the engineer with a basic maturity level on ISO/IEC 29110 Requirements Management (through the study of an SI.2 activity and a review of a 'real' WP11 *Requirements Specification*), and the second involved performing a Requirements Analysis on a 'real' case. The first package was made up of 3 training scenes and the second of a single one. Each TP was calibrated to a week of self-training. The VSE engineer worked through both packages in January 2010.

Favoring reflection-in-action through TPs is, in our opinion, a kind of software improvement. Although no measurements can be easily defined and performed to confirm this, there is empirical evidence of it in the form of 'customer' satisfaction.

The VSE engineer reported that he was now ready to apply the SI.2 SW Requirements Analysis to the Eco-Visio project. As the specifications were established by a subcontractor, he merely reviewed the existing Requirements Specification and rewrote parts of it in order to verify conformity with the template provided in the DP, Software Requirement Analysis [23]. Once updated, the WP11 *Requirement Specification [Validated]* served as an input to SI.5, SW Integration and Tests. The system has been deployed since the end of March 2010, and load testing and application optimization should soon be completed. Defects will then have to be corrected through a short cycle of SI activities.

As an empirical measure of satisfaction with the approach, the VSE asked for a similar approach for SI.5 SW Integration and Tests. In particular, the VSE wanted assistance in establishing a disciplined Change Request Process. This TP is under construction, and we plan to begin with the Software Testing DP [45] as a basis for the whole TP. Probably because tests occur in many SE activities, this DP is organized in a manner that spans PM and SI tasks, which raises many new questions.

D. Towards a sustainable model for a VSE

1) Packaging experiences

For a VSE, the investment in Knowledge Management may appear to take too much time before benefits appear. Obviously, it will take time before a critical mass of experiences will be available in the Experience Repository. Schneider and Schwinn report several problems they faced to in order to achieve a suitable repository, the Experience Base, at DaimlerChrysler. They pointed out the importance of "*thinking [of] an Experience Base as something that needs to be seeded in order to grow*" [46].

Experience management assumes that all relevant experience can be collected and packaged for reuse. Rus and Lindwall have established that there is a difference between explicit and tacit knowledge. "*Explicit knowledge corresponds to the information and skills that employees can easily communicate and document, such as processes, templates, and data*" [14]. Packaging raw experiences in the Experience Factory produces explicit knowledge. "*Tacit knowledge is personal knowledge that employees gain through experience; this can be hard to express and is largely influenced by their beliefs, perspectives, and values*" [14]. Relying on tacit rather explicit knowledge is the prevailing model in a VSE because it does not require the documentation of knowledge or the packaging of experiences. Komi-Sirviö et al. analyzed the case of a company that failed in several attempts to improve knowledge reuse. The company was looking for a new solution that should have a minimal impact on the software development organization. "*This new approach consisted of a knowledge-capturing project and customer projects. The former gathered knowledge from relevant sources and packaged and provided it to a customer project for reuse on demand*" [47]. The knowledge-capturing project is similar to the analysis organization in the Experience Factory framework, but it does this for the customers' project needs.

2) Packaging experiences in a VSE

The previous section suggests that packaging experiences should be performed outside the software development organization. As reported in section IV.C, we used the pilot project to solve an immediate need of the VSE: a disciplined management of requirements. Because the VSE was aware of their weakness in requirements management, they agreed to invest enough time and effort to change their working process for this point. But packaging the required materials was performed by the authors rather than the VSE.

Our proposal for an EM system for a VSE is a simplified approach of the EF infrastructure presented in Figure 1.

The simplified EF is made up of two separate parts: an Experience Repository, and a Training Package Repository. The Experience Repository contains raw experiences; as stated in sections IV.A.2 and IV.B.1, the project manager has to store the main deliverables of the completed project according to a fixed storage scheme. The use of the Experience Repository relies only on the copy-paste knowledge transfer pattern. No help in understanding the raw experiences is provided. When a project is experiencing difficulties in completing a software engineering activity and no useful materials can be found in the Experience Repository, an external task force has to build a Training Package on the given activity and store it in the Training Package Repository. Then, VSE employees may perform self-training using this Training Package. VSE employees may store feedback in the repository in order to improve the process. Self-training tasks are designed to develop reflective thinking. They are based on past experiences, either from the

VSE or from elsewhere. Self-training packages are not intended to explain the packaged experiences for reuse; the main goal is rather to initialize the engineer's repertoire regarding the problematic software engineering activity. Once the self-training package has been completed, the hypothesis is made that the engineer will be able to return to his/her practice and interact with the problematic situation in such a way that it will lead to his/her success. Decisions, support, and suggestions are built up by the engineer her/himself rather than provided by the packaged experiences. Figure 6 shows all the Infrastructure and Process issues that we have addressed in this section.

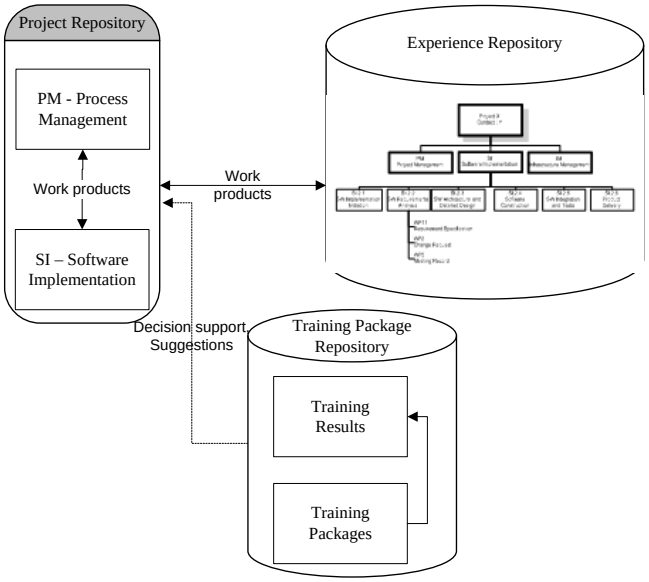


Figure 6. Overview of the EM Infrastructure and Process.

V. CONCLUSION AND FUTURE WORK

We have proposed a simple Experience Management system for a VSE that is compatible with the emerging ISO/IEC 29110 standard. Two hypotheses are posed: (1) the EM infrastructure is kept as simple as possible with the use of a CMS structured with the decomposition of the PM and SI processes; and (2) EM requires dedicated processes that can be taken from D. Schön's reflection-on-action work. The needs of a VSE and the solutions that we have provided are reported as a case study.

Further work is required to consider how the concept of the Training Package could complement that of the Deployment Package.

REFERENCES

[1] V. Ribaud, P. Saliou, and C. Y. Laporte, "Experience Management for Very Small Entities: Improving the Copy-paste Model," in Proc. Fifth International Conference on Software Engineering Advances, New York :IEEE Press, 2010, pp. 311-318.

[2] C. Y. Laporte, "The Development of International Standards for Very Small Entities: Historical Perspectives, Achievements and Way Forward," Joint International Council on Systems Engineering (INCOSE) / Concordia Institute for Information Systems Engineering (CIISE) Distinguished Seminar, 2010.

[3] ISO/IEC TR 29110-1:2011, Software Engineering -- Lifecycle profiles for Very Small Entities (VSEs) -- Part 1: Overview, Geneva: International Organization for Standardization (ISO), 2011.

[4] ISO/IEC 29110-4-1:2011, Software engineering -- Lifecycle profiles for Very Small Entities (VSEs) - Part 4-1: Profile specifications: Generic Profile group, Geneva: International Organization for Standardization (ISO), 2011, available at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51154 (last accessed June 1)

[5] V. Basili, G. Caldiera, and D. Rombach, "Experience Factory," in Encyclopedia of SE, vol. 1, Hoboken:Wiley, 2002, pp. 476-96.

[6] E. Ras, R. Carbon., D. Decker, and J. Rech. "Experience Management Wikis for Reflective Practice in Software Capstone Projects," IEEE Transactions on Education, vol. 50 (4), Nov. 2007, pp. 312-320.

[7] D. Schön, The Reflective Practitioner, New York: Basic Books, 1983.

[8] ISO/IEC TR 10000-1:1998, Information technology -- Framework and taxonomy of International Standardized Profiles -- Part 1: General principles and documentation framework, Geneva: International Organization for Standardization (ISO), 1998.

[9] ISO/IEC 12207:2008, Information technology -- Software life cycle processes, Geneva: International Organization for Standardization (ISO), 2008.

[10] ISO/IEC 15289:2006, Systems and Software Engineering -- Content of systems and software life cycle process information products (Documentation), Geneva: International Organization for Standardization (ISO), 2006.

[11] ISO/IEC TR 29110-5-1-2:2011, Software Engineering -- Lifecycle profiles for Very Small Entities (VSEs) -- Part 5-1-2: Management and engineering guide: Generic profile group: Basic profile, Geneva: International Organization for Standardization (ISO), 2011, available at: http://standards.iso.org/ittf/PubliclyAvailableStandards/c051153_ISO_IEC_29110-5-1-2_2011.zip (last accessed June 1)

[12] C. Y. Laporte, Contributions to SE and the Development and Deployment of International SE Standards for Very Small Entities, PhD thesis, Université de Bretagne Occidentale, Brest, 2009, available at: http://tel.archives-ouvertes.fr/tel-00483255/fr/ (last accessed May 25).

[13] K. Schneider, Experiences and Knowledge Management in Software Engineering, Berlin Heidelberg:Springer-Verlag, 2009.

[14] I. Rus and M. Lindvall, "Knowledge Management in Software Engineering," IEEE Software, vol. 19 (3) , May-June, 2002, pp. 26-38.

[15] V. Basili, G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory," ACM Transactions on SE and Methodology, vol. 1 (1), January 1992, pp. 53-80.

[16] V. Basili and C. Seaman, "The Experience Factory Organization," IEEE Software, vol. 19 (3) , May-June, 2002, pp. 30-31.

[17] T. Chau and F. Maurer, "A Case Study of a Wiki-based Experience Repository at a Medium-sized Software Company," Proc. ACM K-CAP'05, ACM Press, 2005, pp. 185-186.

[18] J. Rech, C. Bogner, and V. Haas, "Using Wikis to Tackle Reuse in Software Projects," IEEE Software, vol. 24 (6), November-December, 2007, pp. 99-104.

[19] C. Argyris, "Organizational learning and management information systems," ACM SIGMIS Database, vol. 13 (2-3), Winter-Spring 1982, pp. 3-11, ISSN:0095-0033.

[20] D. Schön, Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions, San Francisco: Jossey-Bass, 1987.

[21] E. Ras and J. Rech, "Using Wikis to support the Net Generation in improving knowledge acquisition in capstone projects," Journal of Systems and Software, vol. 82, April 2009, pp. 553-562.

[22] ISO/IEC TR 29110-3:2011, Software Engineering -- Lifecycle profiles for Very Small Entities (VSEs) -- Part 3: Assessment guide, Geneva: International Organization for Standardization (ISO), 2011.

[23] S. Alexandre and C. Y. Laporte, "Software Requirement Analysis," http://profs.logti.etsmtl.ca/claporte/VSE/Publications/DP-Software%20Requirements%20Analysis-V1_2.doc, Montréal, 2011 (last accessed May 25).

[24] N. Fenton, S.L. Pfleeger, and R.L.Glass, "Science and substance: A challenge to software engineers," IEEE Software, vol. 11 (4), July 1994, pp. 86-95.

[25] ISO/IEC 15504:2004, Information technology -- Process assessment, Geneva: International Organization for Standardization (ISO), 2004.

[26] C. Argyris and D. Schön, Organizational learning: A theory of action perspective, Reading: Addison Wesley, 1978.

[27] C. Argyris, R. Putnam, and D. McLain Smith, Action Science, Concepts, methods, and skills for research and intervention, San Francisco: Jossey-Bass, 1985.

[28] I. Chan and C. Chao, "Knowledge management in small and medium-sized enterprises," Communications of the ACM, vol. 51 (4), April 2008, pp. 83-88.

[29] J. M. García Alonso, J. J. Berrocal Olmeda, and J. M. Murillo Rodríguez, "Documentation Center – Simplifying the Documentation of Software Projects," Proc. Wiki4SE Workshop – 4th International Symposium on Wikis, Porto, 2008.

[30] J. W. Maxwell, "Using Wiki as a Multi-Mode Publishing Platform," Proc. 25th annual ACM international conference on Design of Communication, ACM, New York, 2001, pp. 196-200.

[31] A. Rauschmayer, "Next-Generation Wikis: What Users Expect; How RDF Helps," Third Semantic Wiki Workshop. at ESWC, Redaktion Sun SITE, Aachen, 2009, poster.

[32] P. M. Senge, The Fifth Discipline. The art and practice of the learning organization, London: Random House, 1990.

[33] E. Ras, "Learning Spaces: Automatic Context-Aware Enrichment of Software Engineeering Experience," PhD Thesis in Experimental Software Engineering no. 29, Stuttgart:Fraunhofer Verlag, 2009.

[34] V. Ribaud, P. Saliou, R. V. O'Connor, and C. Y. Laporte "Software Engineering Support Activities for Very Small Entities," Proc. 17th International Conference on European Systems & Software Process Improvement and Innovation (EuroSPI 2010), Springer-Verlag, September 2010.

[35] A. H. Gold, A. Malhotra, and A. H. Segars, "Knowledge Management: An Organizational Capabilities Perspective," Journal of Management of Information Systems, vol. 18 (1), May 2001, pp. 185-214.

[36] R. Conradi, "From software experience databases to learning organizations," Proc. 11th Int'l Conf. on Software Engineering and Knowledge Engineering (SEKE'99), 1999, Knowledge System Institute, pp. 204-206.

[37] L. Dusink and J. van Katwij, "Reuse dimensions," Proc. Symposium on Software reusability (SSR '95), ACM Press, 1995, pp. 137-149.

[38] M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis, The Capability Maturity Model for Software: Guidelines for Improving the Software Process, SEI Series in Software Engineering, Addison-Wesley, 1995, 640 p.

[39] R. Conradi and T. Dyba, "An empirical study on the utility of formal routines to transfer knowledge and experience," SIGSOFT Softw. Eng. Notes 26(5), 2001, pp. 268-276.

[40] Department of Defense, Defense System Software Development, DOD-STD-2167A, 29 February, 1998.

[41] T. Dyba, "Improvisation in small software organizations," IEEE Software, 17(5), 2000, pp. 82-87.

[42] D. Schön, "Educating the Reflective Practioner" in Meeting of the American Educational Research Association, http://resources.educ.queensu.ca/ar/schon87.htm, 1987, (last accessed January 25).

[43] D. Jonassen, "Designing Constructivist Learning Environments," in Instructional Design Theories and Models: A New Paradigm of Instructional Theory, Mahwah (New Jersey):Lawrence Eribaum Associates, 1999, pp. 215-240.

[44] P. Saliou and V. Ribaud, "Bootstrapping an empty repertoire of experience: The design case," Proc. 1st Workshop on Human Aspects of SE (OOPSLA 2009), ACM, October 2009.

[45] L. Gómez Arenas, "Deployment Package – Software Testing," http://profs.logti.etsmtl.ca/claporte/VSE/Publications/DP-Software Basic Profile Testing-CL00.doc, Montréal, 2010 (last accessed May 25).

[46] K. Schneider and T. Schwinn, "Maturing Experience Base Concepts at DaimlerChrysler," Software Process Improvement and Practice, vol. 6, 2001, pp. 85-96.

[47] S. Komi-Sirvio, A. Mantyniemi, and V. Seppanen, "Toward a practical solution for capturing knowledge for software projects," IEEE Software, 19(3), May/June 2002, pp.60-62.

www.iariajournals.org

International Journal On Advances in Intelligent Systems
ICAS, ACHI, ICCGI, UBICOMM, ADVCOMP, CENTRIC, GEOProcessing, SEMAPRO, BIOSYSCOM, BIOINFO, BIOTECHNO, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS
issn: 1942-2679

International Journal On Advances in Internet Technology
ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING
issn: 1942-2652

International Journal On Advances in Life Sciences
eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO
issn: 1942-2660

International Journal On Advances in Networks and Services
ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION
issn: 1942-2644

International Journal On Advances in Security
ICQNM, SECURWARE, MESH, DEPEND, INTERNET, CYBERLAWS
issn: 1942-2636

International Journal On Advances in Software
ICSEA, ICCGI, ADVCOMP, GEOProcessing, DBKDA, INTENSIVE, VALID, SIMUL, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS
issn: 1942-2628

International Journal On Advances in Systems and Measurements
ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL
issn: 1942-261x

International Journal On Advances in Telecommunications
AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA
issn: 1942-2601