International Journal on Advances in Internet Technology

2016 vol. 9 nr. 3&4

The *International Journal on Advances in Internet Technology* is published by IARIA.
ISSN: 1942-2652
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Internet Technology
Volume 9, Number 3 & 4, 2016

Ajay Chakravarthy, University of Southampton IT Innovation Centre, UK
Chin-Chen Chang, Feng Chia University, Taiwan
Ruay-Shiung Chang, National Dong Hwa University, Taiwan
Tzung-Shi Chen, National University of Tainan, Taiwan
Xi Chen, University of Washington, USA
IlKwon Cho, National Information Society Agency, South Korea
Andrzej Chydzinski, Silesian University of Technology, Poland
Noël Crespi, Telecom SudParis, France
Antonio Cuadra-Sanchez, Indra, Spain
Javier Cubo, University of Malaga, Spain
Sagarmay Deb, Central Queensland University, Australia
Javier Del Ser, Tecnalia Research & Innovation, Spain
Philipe Devienne, LIFL - Université Lille 1 - CNRS, France
Kamil Dimililer, Near East Universiy, Cyprus
Martin Dobler, Vorarlberg University of Applied Sciences, Austria
Jean-Michel Dricot, Université Libre de Bruxelles, Belgium
Matthias Ehmann, Universität Bayreuth, Germany
Tarek El-Bawab, Jackson State University, USA
Nashwa Mamdouh El-Bendary, Arab Academy for Science, Technology, and Maritime Transport, Egypt
Mohamed Dafir El Kettani, ENSIAS - Université Mohammed V-Souissi, Morocco
Marc Fabri, Leeds Metropolitan University, UK
Armando Ferro, University of the Basque Country (UPV/EHU), Spain
Anders Fongen, Norwegian Defence Research Establishment, Norway
Giancarlo Fortino, University of Calabria, Italy
Kary Främling, Aalto University, Finland
Steffen Fries, Siemens AG, Corporate Technology - Munich, Germany
Ivan Ganchev, University of Limerick, Ireland
Shang Gao, Zhongnan University of Economics and Law, China
Kamini Garg, University of Applied Sciences Southern Switzerland, Lugano, Switzerland
Rosario Giuseppe Garroppo, Dipartimento Ingegneria dell'informazione - Università di Pisa, Italy
Thierry Gayraud, LAAS-CNRS / Université de Toulouse / Université Paul Sabatier, France
Christos K. Georgiadis, University of Macedonia, Greece
Katja Gilly, Universidad Miguel Hernandez, Spain
Feliz Gouveia, Universidade Fernando Pessoa - Porto, Portugal
Kannan Govindan, Crash Avoidance Metrics Partnership (CAMP), USA
Bill Grosky, University of Michigan-Dearborn, USA
Jason Gu, Singapore University of Technology and Design, Singapore
Christophe Guéret, Vrije Universiteit Amsterdam, Nederlands
Frederic Guidec, IRISA-UBS, Université de Bretagne-Sud, France
Bin Guo, Northwestern Polytechnical University, China
Gerhard Hancke, Royal Holloway / University of London, UK
Arthur Herzog, Technische Universität Darmstadt, Germany
Rattikorn Hewett, Whitacre College of Engineering, Texas Tech University, USA
Quang Hieu Vu, EBTIC, Khalifa University, Arab Emirates
Hiroaki Higaki, Tokyo Denki University, Japan
Dong Ho Cho, Korea Advanced Institute of Science and Technology (KAIST), Korea
Anna Hristoskova, Ghent University - IBBT, Belgium
Ching-Hsien (Robert) Hsu, Chung Hua University, Taiwan
Chi Hung, Tsinghua University, China
Edward Hung, Hong Kong Polytechnic University, Hong Kong
Raj Jain, Washington University in St. Louis , USA
Edward Jaser, Princess Sumaya University for Technology - Amman, Jordan
Terje Jensen, Telenor Group Industrial Development / Norwegian University of Science and Technology, Norway

Yasushi Kambayashi, Nippon Institute of Technology, Japan
Georgios Kambourakis, University of the Aegean, Greece
Atsushi Kanai, Hosei University, Japan
Henrik Karstoft , Aarhus University, Denmark
Dimitrios Katsaros, University of Thessaly, Greece
Ayad ali Keshlaf, Newcastle University, UK
Reinhard Klemm, Avaya Labs Research, USA
Samad Kolahi, Unitec Institute Of Technology, New Zealand
Dmitry Korzun, Petrozavodsk State University, Russia / Aalto University, Finland
Slawomir Kuklinski, Warsaw University of Technology, Poland
Andrew Kusiak, The University of Iowa, USA
Mikel Larrea, University of the Basque Country UPV/EHU, Spain
Frédéric Le Mouël, University of Lyon, INSA Lyon / INRIA, France
Juong-Sik Lee, Nokia Research Center, USA
Wolfgang Leister, Norsk Regnesentral (Norwegian Computing Center), Norway
Clement Leung, Hong Kong Baptist University, Hong Kong
Longzhuang Li, Texas A&M University-Corpus Christi, USA
Yaohang Li, Old Dominion University, USA
Jong Chern Lim, University College Dublin, Ireland
Lu Liu, University of Derby, UK
Damon Shing-Min Liu, National Chung Cheng University, Taiwan
Michael D. Logothetis, University of Patras, Greece
Malamati Louta, University of Western Macedonia, Greece
Maode Ma, Nanyang Technological University, Singapore
Elsa María Macías López, University of Las Palmas de Gran Canaria, Spain
Olaf Maennel, Loughborough University, UK
Zoubir Mammeri, IRIT - Paul Sabatier University - Toulouse, France
Yong Man, KAIST (Korea advanced Institute of Science and Technology), South Korea
Sathiamoorthy Manoharan, University of Auckland, New Zealand
Chengying Mao, Jiangxi University of Finance and Economics, China
Brandeis H. Marshall, Purdue University, USA
Sergio Martín Gutiérrez, UNED-Spanish University for Distance Education, Spain
Constandinos Mavromoustakis, University of Nicosia, Cyprus
Shawn McKee, University of Michigan, USA
Stephanie Meerkamm, Siemens AG in Erlangen, Germany
Kalogiannakis Michail, University of Crete, Greece
Peter Mikulecky, University of Hradec Kralove, Czech Republic
Moeiz Miraoui, Université du Québec/École de Technologie Supérieure - Montréal, Canada
Shahab Mokarizadeh, Royal Institute of Technology (KTH) - Stockholm, Sweden
Mario Montagud Climent, Polytechnic University of Valencia (UPV), Spain
Stefano Montanelli, Università degli Studi di Milano, Italy
Julius Müller, TU- Berlin, Germany
Juan Pedro Muñoz-Gea, Universidad Politécnica de Cartagena, Spain
Krishna Murthy, Global IT Solutions at Quintiles - Raleigh, USA
Alex Ng, University of Ballarat, Australia
Christopher Nguyen, Intel Corp, USA
Petros Nicopolitidis, Aristotle University of Thessaloniki, Greece
Carlo Nocentini, Università degli Studi di Firenze, Italy
Federica Paganelli, CNIT - Unit of Research at the University of Florence, Italy
Carlos E. Palau, Universidad Politecnica de Valencia, Spain
Matteo Palmonari, University of Milan-Bicocca, Italy
Ignazio Passero, University of Salerno, Italy
Serena Pastore, INAF - Astronomical Observatory of Padova, Italy

Fredrik Paulsson, Umeå University, Sweden
Rubem Pereira, Liverpool John Moores University, UK
Padma Pillay-Esnault, Huawei Technologies, USA
Yulia Ponomarchuk, Far Eastern State Transport University, Russia
Jari Porras, Lappeenranta University of Technology, Finland
Neeli R. Prasad, Aalborg University, Denmark
Drogkaris Prokopios, University of the Aegean, Greece
Emanuel Puschita, Technical University of Cluj-Napoca, Romania
Lucia Rapanotti, The Open University, UK
Gianluca Reali, Università degli Studi di Perugia, Italy
Jelena Revzina, Transport and Telecommunication Institute, Latvia
Karim Mohammed Rezaul, Glyndwr University, UK
Leon Reznik, Rochester Institute of Technology, USA
Simon Pietro Romano, University of Napoli Federico II, Italy
Jorge Sá Silva, University of Coimbra, Portugal
Sébastien Salva, University of Auvergne, France
Ahmad Tajuddin Samsudin, Telekom Malaysia Research & Development, Malaysia
Josemaria Malgosa Sanahuja, Polytechnic University of Cartagena, Spain
Luis Enrique Sánchez Crespo, Sicaman Nuevas Tecnologías / University of Castilla-La Mancha, Spain
Paul Sant, University of Bedfordshire, UK
Brahmananda Sapkota, University of Twente, The Netherlands
Alberto Schaeffer-Filho, Lancaster University, UK
Peter Schartner, Klagenfurt University, System Security Group, Austria
Rainer Schmidt, Aalen University, Germany
Thomas C. Schmidt, HAW Hamburg, Germany
Zary Segall, Chair Professor, Royal Institute of Technology, Sweden
Dimitrios Serpanos, University of Patras and ISI/RC ATHENA, Greece
Jawwad A. Shamsi, FAST-National University of Computer and Emerging Sciences, Karachi, Pakistan
Michael Sheng, The University of Adelaide, Australia
Kazuhiko Shibuya, The Institute of Statistical Mathematics, Japan
Roman Y. Shtykh, Rakuten, Inc., Japan
Patrick Siarry, Université Paris 12 (LiSSi), France
Jose-Luis Sierra-Rodriguez, Complutense University of Madrid, Spain
Simone Silvestri, Sapienza University of Rome, Italy
Vasco N. G. J. Soares, Instituto de Telecomunicações / University of Beira Interior / Polytechnic Institute of Castelo Branco, Portugal
Radosveta Sokullu, Ege University, Turkey
José Soler, Technical University of Denmark, Denmark
Victor J. Sosa-Sosa, CINVESTAV-Tamaulipas, Mexico
Dora Souliou, National Technical University of Athens, Greece
João Paulo Sousa, Instituto Politécnico de Bragança, Portugal
Kostas Stamos, Computer Technology Institute & Press "Diophantus" / Technological Educational Institute of Patras, Greece
Cristian Stanciu, University Politehnica of Bucharest, Romania
Vladimir Stantchev, SRH University Berlin, Germany
Tim Strayer, Raytheon BBN Technologies, USA
Masashi Sugano, School of Knowledge and Information Systems, Osaka Prefecture University, Japan
Tae-Eung Sung, Korea Institute of Science and Technology Information (KISTI), Korea
Sayed Gholam Hassan Tabatabaei, Isfahan University of Technology, Iran
Yutaka Takahashi, Kyoto University, Japan
Yoshiaki Taniguchi, Kindai University, Japan
Nazif Cihan Tas, Siemens Corporation, Corporate Research and Technology, USA
Alessandro Testa, University of Naples "Federico II" / Institute of High Performance Computing and Networking

(ICAR) of National Research Council (CNR), Italy
Stephanie Teufel, University of Fribourg, Switzerland
Parimala Thulasiraman, University of Manitoba, Canada
Pierre Tiako, Langston University, USA
Orazio Tomarchio, Universita' di Catania, Italy
Dominique Vaufreydaz, INRIA and Pierre Mendès-France University, France
Krzysztof Walkowiak, Wroclaw University of Technology, Poland
MingXue Wang, Ericsson Ireland Research Lab, Ireland
Wenjing Wang, Blue Coat Systems, Inc., USA
Zhi-Hui Wang, School of Softeware, Dalian University of Technology, China
Matthias Wieland, Universität Stuttgart, Institute of Architecture of Application Systems (IAAS),Germany
Bernd E. Wolfinger, University of Hamburg, Germany
Chai Kiat Yeo, Nanyang Technological University, Singapore
Abdulrahman Yarali, Murray State University, USA
Mehmet Erkan Yüksel, Istanbul University, Turkey

International Journal on Advances in Internet Technology
Volume 9, Numbers 3 & 4, 2016

CONTENTS

Checklist for the API Design of Web Services based on REST

Pascal Giessler, Roland Steinegger, and Sebastian Abeck

Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Email: pascal.giessler@kit.edu,
Email: roland.steinegger@kit.edu,
Email: sebastian.abeck@kit.edu

Michael Gebhart

iteratec GmbH
Stuttgart, Germany
Email: michael.gebhart@iteratec.de

***Abstract*—The trend towards creating web services based on the architectural style REpresentational State Transfer (REST) is unbroken. Several best practices for designing RESTful web services have been emerged in research and practice to ensure some degree of quality and share solutions to recurring challenges in the area of API design. But, these best practices are often described differently with the same meaning due to the nature of natural language. Also, they are not collected, categorized and presented in a central place but rather distributed across several pages in the World Wide Web, which impedes their application even further. Furthermore, it is often unclear which best practice has to be taken into account when designing a RESTful API for a particular scenario. In this article, we identify, collect, and categorize several best practices for designing APIs for RESTful web services and form a checklist. To support a prioritization of relevant best practices, we have mapped them on quality characteristics of the ISO/IEC 25010/2011. For illustration purpose, we apply the checklist on the CompetenceService as part of the SmartCampus ecosystem developed at the Karlsruhe Institute of Technology (KIT).**

Keywords–REST; RESTful; best practices; checklist; quality-driven; catalog; design; quality; api design; resource-orientation; SmartCampus

I. INTRODUCTION

This article is an extended version of [1]. It represents a collection of common best practices for designing Application Programming Interface (API)s for RESTful web services that have been derived from a range of articles, magazines, and pages on the World Wide Web (WWW). The motivation for this collection was because more and more web services based on the architectural style REST over Hypertext Transfer Protocol (HTTP) were developed and deployed compared to traditional web services with Simple Object Access Protocol (SOAP). This trend can also be seen at big companies, such as Twitter or Spotify, are using REST-like API for their services, which are shown in their API documentations [4] [5]. We are calling it REST-like at this point since we do not want to evaluate if this API considers all constraints of the uniform interface defined by Fielding [6] and can, therefore, be called RESTful. But, there is a lack of standards and guidelines on how to design an appropriate API, for example regarding the usability or the maintainability [2] [3].

Today, an own business model has been established around APIs when looking at the revenue of Salesforce or Expedia [7] [8]. For instance, "Salesforce.com generates 50% of its revenue through APIs," [7] according to the Harvard Business Review. That is why it is more important than ever that APIs have to be designed carefully especially when dealing with a large user base and heterogeneous platforms. For example, a change of an API should not break any consumer and, therefore, must be robust toward evolvability of the API. As discussed in [9], the API design and its strategy were also identified as a solution approach for the challenges of the digital transformation in software engineering.

To meet these challenges regarding the design of APIs, we have collected, categorized and formalized several best practices in the form of a checklist so it can be easily applied during the design of APIs for RESTful web services. To support a prioritization and selection of relevant best practices for a particular application scenario, we have mapped them on quality characteristics of the ISO 25010:2011 [10]. For instance, if it is important to support mobile platforms, then you should consider reducing the necessary requests to get the needed information.

For the purpose of illustration, we first show how we have integrated the checklist in our software development process to ensure an API design with quality in mind. Besides, we show exemplarily the applied best practices on the CompetenceService as part of the SmartCampus system at the KIT. The SmartCampus is a system that provides functionality for students, guests, and members of a university to support their daily life. Today, the SmartCampus is designed according to the trending microservice approach [11] and offers already some services, such as the ParticipationService to support the decision-making process between students, professors and members of the KIT with a new approach called system-consenting [12]. The developed services at the SmartCampus are based on REST, so that they can be used for several different devices as a lightweight alternative to SOAP.

The current paper is structured as follows: In Section II, the architectural style REST will be described in detail to lay the foundation for this article. Afterward, existing papers and articles will be discussed in Section III to show the necessity

of identification, collection, and categorization of existing best practices for the API design of RESTful web services. The CompetenceService is presented in Section IV of which the best practices will be exemplarily illustrated. Besides, our development process will be described to show how APIs will be designed and how the best practices are integrated. In Section V, the checklist containing the best practices will be explained in detail. Section VI focusses the mapping of quality characteristics of the ISO 25010:2011 [10] onto the previously introduced best practices by illustrating them on a concrete scenario. Finally, a summary of this article and an outlook on further work will be given in Section VII.

II. FOUNDATION

This section shall impart the necessary foundation for the scope of this article. First, the architectural style REST and its constraints will be described. Then, the term API will be described while a classification model for APIs based on REST will be introduced. In addition, informal characteristics of a API will be represented that can be found in literature.

A. REpresentational State Transfer (REST)

REST is an architectural style, which was developed and first introduced by Fielding [6] in his dissertation. According to Garlan and Shaw [13], an architectural style can be described as follows: "an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined." [13, p. 6].

For the design of REST, Fielding [6] has identified four key characteristics, which were important for the success of the current WWW [14]. To ensure these characteristics, the following constraints were derived from existing network architectural styles together with another constraint for the uniform interface [6]:

1) Client and Server: A client component sends a request to a server component for executing a remote operation. It is incumbent upon the server component to perform or reject the request [6].
2) Statelessness: Each request from client to server has to contain all necessary information to perform the request, which leads to the following advantage: "There is no need for the server to maintain an awareness of the client state beyond the current request" [6, p. 119].
3) Layered Architecture: A layered-client-server architecture enables the application of the Separation of Concern (SoC) principle and the opportunity to add features like load balancing or caching mechanisms to multiple layers [6] [15].
4) Caching: This constraint allows a client to match its request to a previous response from the server with the result that no request has to be transmitted over the network [14].
5) Code on Demand: With the usage of Code on Demand, additional programming logic can be requested from the server that is needed for processing received information from the server [14].
6) Uniform interface: The term "uniform interface" (hereinafter API) can be seen as an umbrella term, since it can be decomposed into four sub constraints [14]: 6.1) Identification of resources, 6.2) manipulation of resources through representations, 6.3) Self-descriptive messages and 6.4) Hypermedia.

If all of these constraints are fulfilled by a web service, it can be called RESTful. The only exception is "Code on Demand", since it is an optional constraint and has not be implemented by a web service. The mentioned constraints are illustrated in Figure 1.

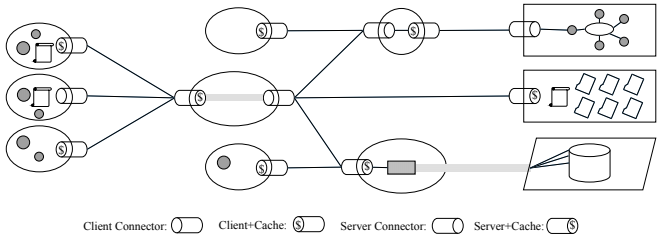

Figure 1. REST style [6, p. 84]

B. (Web) API

An Application Programming Interface (API) is a "local interface from higher-level component to a lower-level component" and can also be called as a horizontal interface [16, p. 915]. It acts as a contract between the service and the service consumer in the area of web services. It describes how the client can communicate with the service and how the request will be processed and responded to. An API can be called a web API, when the interface can be accessed via the internet or via internet-enabled technologies, such as HTTP.

For the classification of APIs based on REST, Richardson et. al. have developed a maturity model that classifies the API according their compliance of the mentioned preconditions for the uniform interface (see Section II-A 6.1 - 6.4) [17]. This so-called Richardson Maturity Modell (RMM) consists of three different maturity levels: 1) Usage of a resource-oriented styles rather than Remote Procedure Call (RPC) style, 2) Usage of HTTP methods according to their semantic and, 3) Usage of Hypermedia so that the API is self-documenting. For better illustration, the RMM is represented by Figure 2.

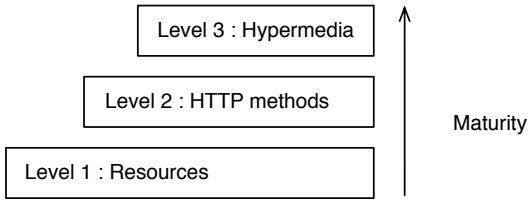

Figure 2. Richardson Maturity Model

Besides this classification, there are also informal criteria that can be found in literature especially when looking for integration technology. According to Newman [11], breaking changes should be avoided. This means that the API should be robust in terms of its evolution since it acts as a contract between service consumer and service provider. Also, the API should be technology-agnostic and hide internal implementation details not to increase the coupling. That is why the

API should abstract from a particular implementation by not exposing internal details, such as the used technology or any proprietary standard. Another important characteristic is the simplicity, and the comprehension from a service consumer perspective since other developers or development teams are the primary target group.

III. Background

This section discusses different articles, magazines and approaches in the context of RESTful best practices, which respect the architectural style REST and its underlying concepts.

In Fielding [6], Fielding presents the structured approach for designing the architectural style REST, while it remains unclear how a REST-based web service can be developed in a systematic and comprehensible manner. Furthermore, there is also a lack of concrete examples of how hypermedia can be used as the engine of the application state, which can be one reason why REST is understood and implemented differently.

In [11], Newman presents a book about the microservice approach that is followed by several companies. Although, there is dedicated chapter regarding the ideal integration technology, it lacks on concrete best practices or guidelines especially for APIs based on REST. Instead of this, the book provides rather an overview about the technology choices that have to be made when choosing an integration technology.

Mulloy [18] presents different design principles and best practices for Web APIs, while he puts the focus on "pragmatic REST". By "pragmatic REST" the author means that the usability of the resulting Web API is more important than any design principle or guideline. But, this decision can lead to neglecting the basic concepts behind REST, such as hypermedia.

Jauker [19] summarizes ten best practices for a RESTful API, which represent, in essence, a subset of the described best practices by Mulloy [18] and a complement of new best practices. Comparable with [18], the main emphasis is placed on the usability of the web interface and not so much on the architectural style REST, which can lead to the previously mentioned issue.

Papapetrou [20] classifies best practices for RESTful APIs in three different categories. However, there is a lack of concrete examples of how to apply these best practices on a real system compared to the two previous articles.

In [21], a checklist of best practices for developing RESTful web services is presented, while the author explicitly clarifies that REST is not the only answer in the area of distributed computing. He structures the best practices in four sections, which addressing different areas of a RESTful web service, such as the representation of resources. Despite all of his explanations, the article lacks in concrete examples to reduce the ambiguousness.

Richardson et. al [14] cover in their book as a successor of [22], among other topics, the concepts behind REST and a procedure to develop a RESTful web service. Furthermore, they place a great value on hypermedia, as well as Hypermedia As The Engine Of Application State (HATEOAS), which is not taken into account by all of the prior articles. But, the focus of this work is the comprehensive understanding of REST rather than providing best practices for a concrete implementation to reduce the complexity of development decisions. s In [23], Burke presents a technical guide of how to develop web services based on the Java API for RESTful Web Services (JAX-RS) specification. But, this work focuses on the implementation phase rather than the design phase of a web service, where the necessary development decisions have to be made.

In [24], Guinard and Trifa provide a guide on how to design and implement Internet of Things (IoT) solutions on the Web to ensure interoperability across different platforms. They recommend using a hypermedia approach when designing an API. But, common challenges, such as troubleshooting of an API, naming of resources and error handling are not discussed in detail.

IV. Scenario

This section addresses the domain at which the best practices for the API design will be illustrated. Besides, it shows the engineering principle behind the design and the development of the SmartCampus at the KIT to highlight the importance of APIs and the necessity of best practices.

A. Domain

The SmartCampus is a modern web application, which simplifies the daily life of students, guests, and members at the university (see Figure 3). Today, it offers several services, such as the ParticipationService for decision-making [12], the SmartMeetings for discussions or the CampusGuide for navigation and orientation on the campus, and the WorkspaceService to find a free working place by using smart devices. By using non-client specific technologies, the services can be offered to a wide range of different client platforms, such as Android or iOS.

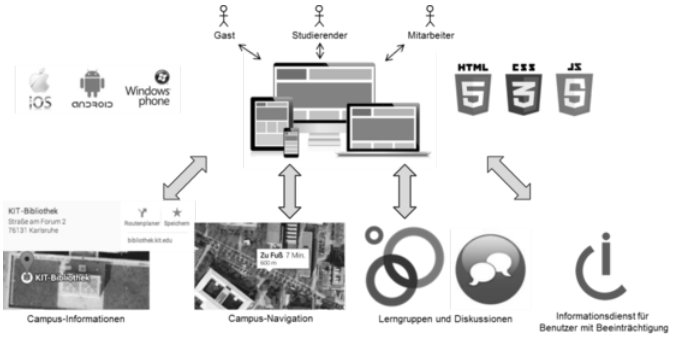

Figure 3. SmartCampus [25].

The CompetenceService is a new service as part of the SmartCampus to capture and semantically search competences in the area of information technology. For easier acquisition of knowledge information, the CompetenceService offers the import of competence and profile information from various social networks, such as LinkedIn or Facebook. The resulting knowledge will be represented by an ontology, while the profile information will be saved in a relational database. SPARQL Protocol And RDF Query Language (SPARQL) is used as the query language for capturing and searching knowledge information in the ontology.

In Figure 4, the previously described CompetenceService is illustrated in the form of a component diagram. As integration

technology according to Newman [11], REST was chosen since we are dealing with semantic information that should be reflect by the API. For implementation of the Competence-Service, the Java framework Spring was used.

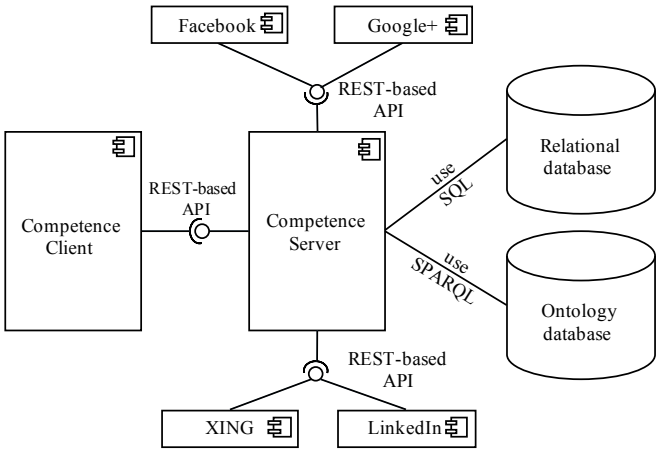

Figure 4. Component model of the CompetenceService.

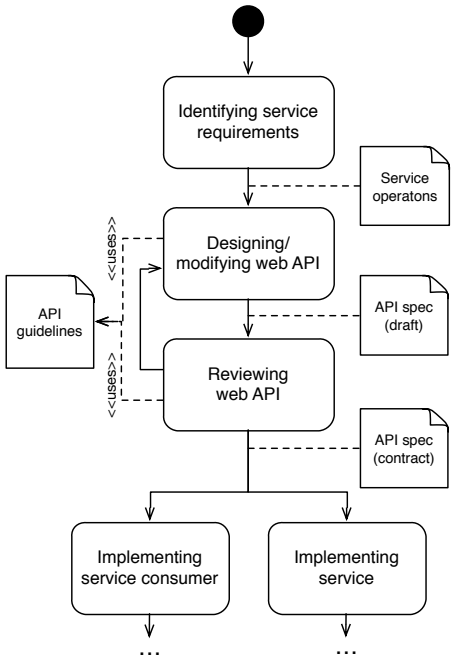

Figure 5. API-First engineering principle.

To demonstrate the benefits of this service, a simple use case will be described in the following. A young startup company is looking for a new employee, who has competences in “AngularJS” and “Bootstrap”. For that purpose, the startup company uses the semantics search engine of the CompetenceService to search for people with the desired skills. The resulting list of people will be ordered by relevance so that the startup company can easily contact the best match.

B. API-Driven Development Process

For the design and development of our services as part of the previously introduced SmartCampus, we have decided to choose the API-First key engineering principle. The API-First engineering principle allows us to focus on the quality of the API design before any implementation effort is made. In our eyes, the APIs are an important and highly valuable business asset since they define what we can do with the system.

In Figure 5, our API-First engineering principle is shown. First, we identify the service requirements that we have gathered from scenarios. Scenarios are described from the perspective of a user that interacts with the software by using its user interface [26]. The resulting service operations are used for the first draft of the API. For the design of the API, we have defined some API guidelines based on the checklist in this article to ensure the consistency, maintainability, and usability of our service landscape. The API guidelines represent decisions that we have taken in previous development projects. After the first draft of the API specification is created, an ample peer review based on the checklist and the application scenario will be conducted by a dedicated team to ensure the quality of the API. For the API specification, we rely on the OpenAPI as a vendor neutral API specification format (former Swagger specification) since it is open source and is supported by a vast majority of big companies, such as Google, Microsoft, and IBM [27]. When reviewing the API, the responsible developers have to explain their design decisions. If the resulting quality report revealed some issues, the iteration cycle starts from the beginning with the modification of the API specification draft. In the other case, the API gets the status of a contract and will be handed over to the appropriate development teams.

V. Checklist for the API Design of Web Services Based on REST

This section presents eight different categories of best practices for designing REST-based web services, whereby each one is represented by a subsection (see Figure 6). The categories semantically group the found best practices and shall act as a checklist of important aspects that have to be considered when designing APIs for web services based on REST. Furthermore, we are providing recommendations for each category based on literature, that we have found during our conduction. The best practices should not be seen as strict guidelines. Furthermore, it is important to point out here that the fulfillment of the following best practices does not guarantee the compliance of the mentioned constraints in Section II. For this, the RMM can be used to analyze the preconditions of a REST-based web service (see Section II-B).

A. Versioning

Versioning of a Web API is one of the most important considerations during the design of web services since the API represents the central access point of a web service and hides the service implementation. This is why a web interface should never be deployed without any versioning identifier according to Mulloy [18]. For versioning, many different approaches exist, such as embedding it into the base Uniform Resource Identifier (URI) of the web service or using the HTTP-Header for selecting the appropriate version [18]. But, web services based on REST do not need to be versioned due to hypermedia. The hypermedia aspect allows us to update the hyperlinks at runtime since the client does not hardcoded them in its

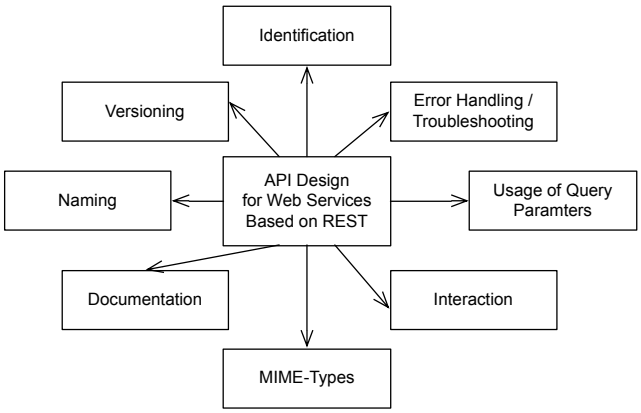

Figure 6. Categories of best practices for designing REST-based web services

client code and is, therefore, up to date as soon as the service provides a new representation of a requested resource.

That is why RESTful web services can be compared with traditional websites that are still accessible on all web browsers when modifying the content of the websites. So, no additional adjustment is necessary on the client side. Furthermore, versioning also has a negative impact on deployed web services, which Fielding states as follows: "Versioning an interface is just a polite way to kill deployed applications" [28] since it increases the effort for maintaining the web service.

B. Naming

The naming and description of resources correlates with the usability of the web service since the resources represent or abstract the underlying domain model. Furthermore, by defining guidelines for naming, we can ensure a consistent naming style across several services within the service-oriented architecture. This leads to a consistent look and feel regardless of the development team, who has designed and developed it. For this category, five best practices could be identified:

1) According to Vinoski [21], Papapetrou [20] and Mulloy [18], nouns should be used for resource names. Since a subresource is simply a resource with a composition relationship to another resource, we think this rule should be applied here as well.
2) The name of a resource should be concrete and domain specific, so that the semantics can be inferred by a user without any additional knowledge [18] [20].
3) The amount of resources should be bounded to limit the complexity of the system, whereby this recommendation depends on the degree of abstraction of the underlying domain model [18].
4) The mixture of plural and singular by naming resources should be prevented to ensure consistency. In addition, a resource should be able to handle several entities instead of just one one. But, there may be exceptions, such as Spotify's me resource [18] [19].
5) The naming convention of JavaScript should be considered since the media type JavaScript Object Notation (JSON) is the most used data format for the client and server communication by this time [3] [18] [29]. For instance, Google has defined an extensive styleguide [30].

Figure 7 illustrates the first, second and third best practice of this category.

```
1  /* ProfileController */
2  @RestController
3  @RequestMapping(value = "/profiles")
4  public class ProfilesController {
5    ...
6    @RequestMapping(method = RequestMethod.GET)
7    public List<Profile> getProfiles() {...}
8    ...
9  }
10
11 /* CompetenceController */
12 @RestController
13 @RequestMapping(value = "/competences")
14 public class CompetenceController {
15   ...
16   @RequestMapping(method = RequestMethod.GET)
17   public List<Competence> getCompetences() {...}
18   ...
19 }
```

Figure 7. Example for description of resources.

C. Resource Identification

According to Fielding [6], URIs should be used for unique identification of resources. If we take it accurately, there is no need for declaring best practices for resource identification when following a hypermedia approach since only the meta-information of the hyperlink will be evaluated and processed by the service consumer. But, we have found several best practices regarding resource identification. On the basis of this result, we assume that most so-called REST-based APIs are positioned on the second level of the RMM rather than third one. That is why we have decided to list the found best practices to improve the usability for this kind of API:

1) An URI should be self-explanatory according to the affordance [18]. The term affordance refers to a design characteristic by which an object can be used without any guidance. Since the main part of a URL consists of resource names, we have to ensure that these names are also domain-related and not termed in an abstract way.
2) A resource should only be addressed by two URIs. The first URI address represents a set of states of the specific resource and the other one a specific state of the previously mentioned set of states [18].
3) The identifier of a specific state should be difficult to predict [20] and not references objects directly according to the Open Web Application Security Project (OWASP) [31], if there is no security layer available.
4) There should be no verbs within the URI since this implies a method-oriented approach, such as SOAP [18] [19].

Figure 8 illustrates the second best practice of this category. Note that there are no verbs within the URIs, hence the fourth best practice is also fulfilled.

```
1 /* Set of profiles */
2 competence-service/profiles
3
4 /* Specific profile with identifier {id} */
5 competence-service/profiles/{id}
```

Figure 8. Example for identification of resources.

D. Error Handling and Troubleshooting

As already mentioned, the API represents the central access point web service, which is comparable with a provided interface of a software component [32]. Each information about the implementation of the service is hidden by the interface. Therefore, only the outer behavior can be observed through responses by the web service, which is why well-known software debugging techniques, such as setting exception breakpoints can not be applied.

For this reason, the corresponding error message has to be clear and understandable so that the cause of the error can be easily identified. With this in mind, we could identify three best practices:

1) The amount of used HTTP status codes should be limited to reduce the feasible effort for looking up in the specification. At this time, there are over 60 different status code with different semantic [18] [19].
2) Specific HTTP status codes should be used according to the official HTTP specification [33] and the extension [34] [19] [21] [20].
3) A detailed error message should be given as a hint for the error cause on client side [18] [19]. That is why an error message should consist of six ingredients: 3.1) An absolute Uniform Resource Locator (URL) that identifies the problem type, 3.2) A short summary of the problem type, that is written in english and comprehensible for software engineers, 3.3) The HTTP status code that was generated by the origin server, 3.4) An application specific error code, 3.5) A detailed human readable explanation specific to this occurrence of the problem and 3.6) An URL with further information about the specific error and occurrence.
4) For operational troubleshooting, it would be beneficial to use an application-specific or unique identifier that will be send with each request. This allows a filtering of service logs, when an issue was reported.

Figure 9 illustrates the mentioned ingredients of an error message according to the third best practice of error handling.

```
1  HTTP/1.1 503 SERVICE UNAVAILABLE
2  /* More header information */
3  {
4   "problem":
5   {
6     "type": "http://httpstatus.es/503",
7     "status": 503,
8     "error_code": 173,
9     "title": "The service is currently under
          maintenance",
10    "detail": "Connection to database timed out",
11    "instance": "http://.../errors/173"
12   }
13 }
```

Figure 9. Example for detailed error message.

E. Documentation

A documentation for Web APIs is a debatable topic in the context of RESTful web services since it represents an out-of-band information, which should be prevented according to Fielding: “Any effort spent describing what method to use on what URIs of interest should be entirely defined within the scope of the processing rules for a media type” [35]. This statement can be explained with the fact that documentation is often used as a reference book in traditional development scenarios. As a result of this, it can lead to hardcoded hyperlinks in the source code instead of interpreting hyperlinks of the current representation following the HATEOAS principle. Also business workflows will be often implemented according to the documentation. In this case, we call it Documentation As The Engine Of Application State (DATEOAS). As a result of this, we have developed a new kind of documentation in consideration of HATEOAS to give developers a guidance for developing a client component.

The new documentation consists of three ingredients: 1) Some examples which show how to interact with different systems according to the principle of HATEOAS due to the fact that some developers are not familiar with this concept [35], 2) an abstract resource model in form of a state diagram, which defines the relationship and the state transitions between resources. Also, a semantics description of the resource and its attributes should be given in form of a profile, such as Application-Level Profile Semantics (ALPS) [36], which can be interpreted by machines and humans and 3) a reference book of all error codes should be provided so that developers can get more information about an error that has occurred.

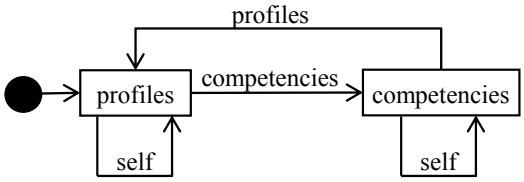

Figure 10. Example for documentation of the Web API.

Figure 10 illustrates an abstract resource model of the CompetenceService. Based on this model, it can be derived which request must be executed to get the desired information. For example to get all competences of a specific profile, we have to first request the resource *profiles*. This results in a set of available profiles, whereby each profile contains one hyperlink for further information. After following the hyperlink by selecting the desired profile, the whole information about the profile will be provided, as well as further hyperlinks to related resources, such as *competences*.

F. Usage of Query Parameters

Each URI of a resource can be extended with parameters to forward optional information to the service. This is important when operating on the result set before transmitting over the network. For instance, the selection of relevant information can reduce the transmitted payload. We have identified five different use cases since they will be supported by several platforms, such as Facebook or Twitter.

1) Filtering: For information filtering of a resource either its attributes or a special query language can be used. The election for one of these two variants depends on the necessary expression power of the information filtering. Figure 11 illustrates how a special user group can be fetched by using a query language [19].

```
1 GET /profiles?filter=(competencies=java%20and%20
      certificates=MCSE_Solutions_Expert)
```

Figure 11. Filtering information by a using query language.

2) Sorting: For information sorting, Jauker [19] recommends a comma separated list of attributes with "sort" as the URI parameter followed by a plus sign as a prefix for an ascending order or a minus sign for a descending order. Finally, the order of the attributes represents the sort sequence. Figure 12 illustrates how information can be sorted by using the attributes *education* and *experience*.

```
1 GET /profiles?sort=-education,+experience
```

Figure 12. Sorting a resource by using attributes.

3) Selection: The selection of information in form of attributes reduces the transmission size over the network by responding only with the requested information. For this purpose, Mulloy [18] and Jauker [19] recommend a comma separated list of attributes and the term *fields* as the URI parameter. Figure 14 represents an example how the desired information can be selected before transmitting over the network.

```
1 GET /profiles?fields=id,name,experience
```

Figure 13. A selection of resource information.

4) Search: The search for relevant information is a common use case. It is recommended to use the default query parameter *q* or using entity attributes, such as *id* or *uuid*.

```
1 GET /profiles?q=Java;Scala
```

Figure 14. A search of relevant information.

5) Pagination: Pagination enables the splitting of information on several virtual pages, while references for the next (*next*) and previous page (*prev*) exist, as well as for the first and last page (*first* and *last*).

```
1 GET /profiles?offset=0&limit=10
```

Listing 1. Requesting 10 profiles by using pagination.

As URI parameter, *offset* and *limit* were recommended, whereby the first one identifies the virtual page and the last one defines the amount of information on the virtual page [18] [19]. A default value for *offset* and *limit* can not be given since it depends on the information to be transmitted to the client, which Mulloy stated [18] as follows: "If your resources are large, probably want to limit it to fewer than 10; if resources are small, it can make sense to choose a larger limit" [18, p. 12]. Figure 1 illustrates a request using pagination on the resource *profiles*.

Although, the mentioned pagination technique is often recommended in literature, there are some issues especially when dealing with big data volumes or when fetching two virtual pages during inserting or deleting operation. These issues are outlined in [37].

G. Interaction with Resources

By using REST as the underlying architectural style of a system, a client interacts with the representations of a resource instead of using it directly. The interaction between client and server is built on the application layer protocol HTTP, which already provides some functionality for the communication. For the interaction with a resource, we could identify five different best practices:

1) According to Jauker [19] and Mulloy [18], the used HTTP methods should be conform to the method's semantics defined in the official HTTP specification. So, the HTTP-GET method should only be used by idempotent operations without any side effects. For a better overview, Table I sums up the most frequently used HTTP methods and their characteristics. These characteristics can be used to associate the HTTP methods with the correct Create Read Update Delete (CRUD)-operation [21].
2) The support of HTTP-OPTIONS is recommended if a large amount of data has to be transmitted since it allows a client to request the supported methods of the current representation before transmitting information over the shared medium. But, this additional HTTP-OPTIONS request is only necessary, if the supported operations were not written explicitly in the representation or when dealing with Cross-Origin Resource Sharing CORS) [38].
3) A compression mechanism, such as GZIP should be supported to reduce the payload. By using the HTTP header field "Accept-Encoding", the client can indicate that he expects an appropriate encoding while on the other side, the server can set the "Content-Encoding" field when using content encoding. If the client does not set the "Accept-Encoding", the server should use compression by default.

4) The support of conditional GET should be considered during the development of a service based on HTTP since it prevents the server from transmitting previously sent information. Only if there are modifications of the requested information since the last request, the server responds with the latest representation. For the implementation of conditional GET, there are two different approaches that are already described by Vinoski [21].
5) The support of partial updates should be considered so that the client does not have to send unchanged information. This is relevant when sending a large amount of data since the bandwidth of upstream is usually much lower than for downstream.

TABLE I. CHARACTERISTICS OF THE MOST USED HTTP METHODS.

HTTP method	safe	idempotent
POST	No	No
GET	Yes	Yes
PUT	No	Yes
DELETE	No	Yes
OPTIONS	Yes	Yes
PATCH	No	No

H. MIME Types

Multipurpose Internet Mail Extensions (MIME) types are used for the identification of data formats, which will be registered and published by the Internet Assigned Numbers Authority (IANA). These types can be seen as representation formats of a resource. For this category, we could identify the following four best practices:

1) At least two representation formats should be supported by the web service, such as JSON or Extensible Markup Language (XML) [18].
2) JSON should be the default representation format since its increasing distribution [18].
3) Existing MIME types should be used, which already support hypermedia, such as JSON-LD (JSON for Linking Data), Collection+JSON and Siren [21].
4) Content negotiation should be offered by the web service, which allows the client to choose the representation format by using the HTTP header field "Accept" in his request. Furthermore, there is the opportunity to weight the preference of the client with a quality parameter [21].

VI. QUALITY-DRIVEN PRIORITIZATION OF BEST PRACTICES

After describing the identified categories of best practices for the API, the next logical step is the selection of relevant best practices for a given scenario. That is why we have mapped them on quality characteristics of the ISO/IEC 25010:2010 [10] (see Figure 15). For illustration purpose, we take the assumption that API of the current CompetenceService has to be optimized for mobile platforms. This means, for example, that the necessary requests for getting the needed information of the service have to be minimized and transmitted as fast as possible to reduce the latency and improve the responsiveness for the consumers. If we look at Figure 15, this will comply with the time behaviour quality sub characteristic as part of the performance efficiency quality characteristic.

(Sub)Characteristic
Functional suitability
Functional completeness
Functional correctness
Functional appropriateness
Performance efficiency
Time behaviour
Resource utilization
Capacity
Compatibility
Co-existence
Interoperability
Usability
Appropriateness recognizability
Learnability
Operability
User error protection
User interface aesthetics
Accessibility

Reliability
Maturity
Availability
Fault tolerance
Recoverability
Security
Confidentiality
Integrity
Non-repudiation
Accountability
Authenticity
Maintainability
Modularity
Reusability
Analysability
Modifiability
Testability
Portability
Adaptability
Installability
Replaceability

Figure 15. Product quality model of the ISO/IEC 25010:2011 [10]

After gathering the non-functional requirements or quality requirements for the API, we can select the appropriate categories and best practices that have to be considered. The method for the quality-driven prioritization approach is illustrated in Figure 16.

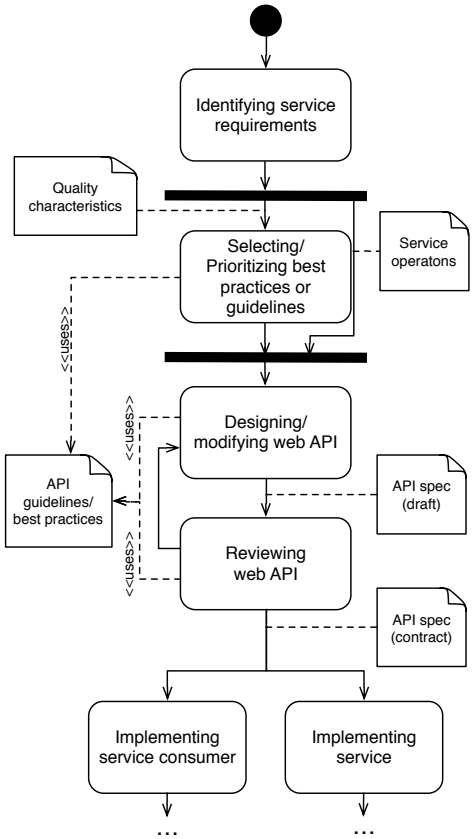

Figure 16. Revised API-First engineering principle with quality-driven prioritization

A. Best Practices for Performance Efficiency

For the improvement of the performance efficiency according to the ISO/IEC 25010:2010 [10], we have identified five relevant best practices that should be considered when designing an API. In particular, we make no statement regarding the completeness of the upcoming list of best practices for the performance efficiency.

1) Usage of Query Parameters (see Section V-F): The required network bandwidth load for transferring the response can be further reduced by a preprocessing step on service side based on given optional parameters. The optional parameters can be handed over through query parameters. For instance, a subset of representation fields can be selected by using the appropriate query parameter *fields*. Besides, the parsing effort on service consumer side can also lead to an increasing responsiveness.
2) Support of HTTP-OPTIONS (see Section V-G-2) should be implemented to allow requesting the supported HTTP methods. Keep in mind, the support is necessary when dealing with CORS and asynchrony.
3) Use Compression (see Section V-G-3): Each transferred information over the shared network should be compressed since mobile networks have often higher latency and lower bandwidth compared to traditional networks.
4) Support of conditional GET (see Section V-G-4) should be implemented by the service so that only new information will be send to the client.
5) Support partial updates (see Section V-G-5): The bandwidth for upstream is lower than for downstream so that the service should be able to handle partials updates and not expect a full representation of a resource.

Besides these mentioned best practices, we have also recognized that the number of requests should be minimized in a mobile scenario. But, we have found no best practice that can be directly mapped onto this. In our eyes, this can be achieved by one of the following approaches. For the sake of completeness, we have also list a non REST-like approach at the end.

1) Combining resources to a new more abstract resource tailored to the specific use case that is needed on the mobile client.
2) Providing an additional orchestration layer (e.g. Backend-For-Frontend (BFF) [11]) that splits the client request in multiple server requests and responds with a collected response.
3) Using a technology-driven and not REST-like approach, such as GraphQL that offers a declarative way for fetching the required information [39].

By following the mentioned best practices, we have could improve the perceptible responsiveness of the mobile client application. Figure 17 illustrates the average latency of two different versions of one mobile client - 1) before and 2) after application of the mentioned best practices that results in an API change. At the beginning, the mobile client application requests some initial data plus some app configuration so the latency is typically higher compared to the further course of the usage.

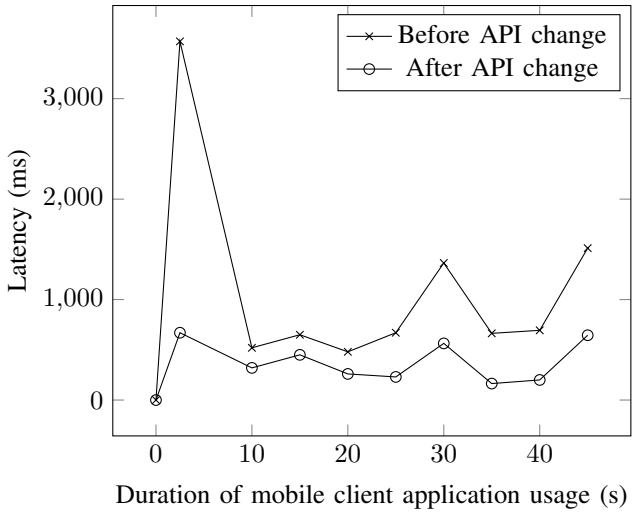

Figure 17. Mobile client application latency before and after applying the quality-driven selection of best practices

B. Mapping of Best Practices and Quality Characteristics

In the previous section, we have illustrated the quality-driven approach by selecting best practices according to their influence on the performance efficiency. This section shall focus the whole set of the quality characteristics by presenting a influence map from the mentioned best practices onto the quality characteristics of the ISO/IEC 25010:2011 [10] (see Figure 18). The dashed lines represent positive impacts of the best practices on the quality characteristics.

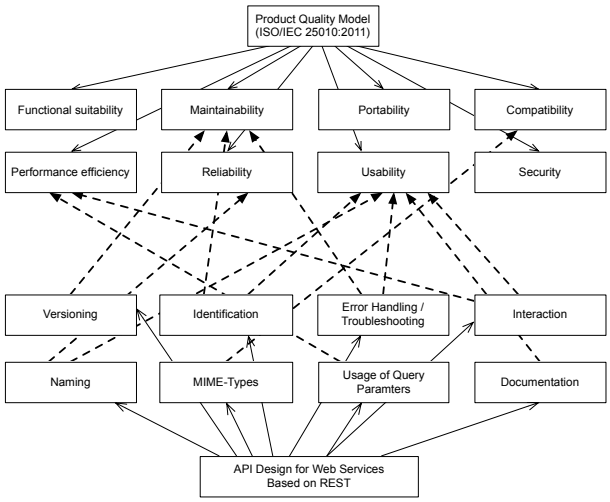

Figure 18. Influence map of best practices for API design for web services based on REST onto the ISO/IEC 25010:2011 [10]

The intention is to support API designers especially for web services based on REST by a kind of prioritization of best practices. In our eyes, the selection of best practices based on quality characteristics is more intuitive than traditional approaches, such as the requirement-level indication of the RFC 2119 [40]. The reason for this is that the importance of different best practices can vary from organization to organization.

VII. Summary and Outlook

In this article, we created a checklist for the API design of web services based on REST. More precisely, we identified and collected best practices for the API design. These best practices were classified into eight different categories that focus on different aspects of an API. The categories and best practices represent the checklist: Each category describes an aspect that has to be considered when designing an API. The best practices within one category represent recommendations that have to be kept in mind when the category as aspect is important for a specific API. For instance, one category is versioning. If versioning is something that is important for the specific API, then the best practices within this category should be kept in mind when designing the API. Finally, we associated the categories with quality characteristics of the ISO/IEC 25010:2010 [10] to show the impact of best practices on the quality of the API.

The intention of this article was not to reinvent the wheel. For this reason, the best practices within this article were re-used from existing work. Instead, the focus of this work was to identify and collect existing best practices, to unify them, and to associate them with certain aspects of an API design. Best practices are only helpful when software architects and developers know when to consider them and what they are for. For this reason, our classification into eight categories helps to decide, whether a certain best practice should be considered or not. Furthermore, our best practices are not meant to be complete. They are more a recommendation about what should be kept in mind when a certain aspect (category) is relevant for an API design.

To illustrate the applicability of our checklist, we applied the checklist, i.e., its categories and their best practices on a concrete scenario. As a scenario, we chose the SmartCampus of the Karlsruhe Institute of Technology. SmartCampus is a modern web application. Its purpose is to simplify the daily life of students, guests, and members at the university. The SmartCampus consists of several provided services. One service is the CompetenceService that semantically searches competences in the area of information technology. By applying our checklist, we could identify relevant best practices for our API. The checklist helped us to identify, which of the best practices are relevant for this service and which ones are not. The best practices are not strict guidelines; they are more recommendations that helped us to keep certain aspects in mind. By using the checklist during the API design, we had a concrete list about what to consider. This helped us to not forget relevant aspects when creating the API.

Summarized, the checklist, i.e., the categories and their best practices help software architects and developers to design the API of web services with certain recommendations kept in mind. As today, best practices are distributed across several existing works, until now, it was hard to find a unified set of best practices. Furthermore, the best practices were isolated. It was not clear, whether a certain best practice should be considered or not. Its impact was not obvious. With our classification into eight categories, software architects and developers get the possibility to filter the best practices and to understand, which ones are necessary and which ones are not relevant for a certain API design. This reduces the amount of best practices to the relevant ones and simplifies the application of best practices. Even though our checklist is not meant to be complete, it is a helpful list of best practices, i.e., recommendations. This list reminds software architects and developers of aspects they should consider when designing an API with certain categories being relevant. Furthermore, for software architects and developers it is not necessary any more to lookup best practices in literature. As the checklist is a first collection and unification of widespread best practices, software architects and developers can directly start with this checklist in their daily projects. As we build on existing work and reference this work, detailed descriptions can be looked up if necessary. But it is not necessary to spend time to collect best practices from scratch. With the association to quality characteristics of ISO/IEC 25010:2010 [10], software architects and developers get an understanding about the impact of the best practices and certain design decisions on the quality of the API. This increases the awareness of design decisions and helps to create APIs in a quality-oriented manner.

In the future, we plan to investigate the impact of our checklist on the development speed, as well as the quality. To evaluate the usefulness of the best practices for API design, we consider setting up two teams of students, Team A and Team B. Both teams get the requirement to develop two services as part of the SmartCampus at the KIT of similar complexity. Both teams are expected to have similar experiences in developing software systems, and both teams should not have knowledge about best practices for API design. However, Team A will be equipped with our checklist. We expect that Team A will spend much less time searching appropriate best practices. The checklist will provide Team A appropriate best practices about how to design the API. Figure 19 shows the expected results.

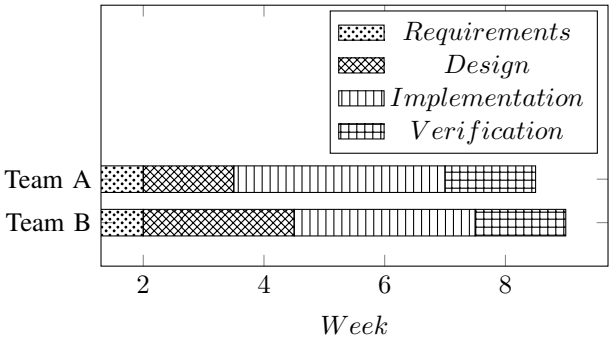

Figure 19. Duration of the development phases in weeks.

In addition, we plan to extend our checklist with further best practices and to describe the best practices by means of technology-independent metrics. In a next step, we plan to map these technology-independent metrics onto concrete technologies, such as Java and JAX-RS. This mapping constitutes the basis for an automated application of the metrics on concrete design or implementation artifacts [41]. Besides, we will use the checklist on upcoming projects an exemplifies the whole API design process in a more detail so that each development team is capable of applying it on its own projects.

References

[1] P. Giessler, M. Gebhart, D. Sarancin, R. Steinegger, and S. Abeck, "Best Practices for the Design of RESTful web Services," International Conferences of Software Advances (ICSEA), 2015. [Online]. Available: http://www.thinkmind.org/download.php?articleid=icsea_2015_15_10_10016

[2] R. Mason, "How rest replaced soap on the web: What it means to you," October 2011, URL: http://www.infoq.com/articles/rest-soap [accessed: 2015-02-20].

[3] A. Newton, "Using json in ietf protocols," the IETF Journal, vol. 8, no. 2, October 2012, pp. 18 – 20.

[4] Spotify, "Web API Endpoint Reference," URL: https://developer.spotify.com/web-api/endpoint-reference/ [accessed: 2016-09-30].

[5] Twitter, "REST APIs," URL: https://dev.twitter.com/rest/public [accessed: 2016-09-30].

[6] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[7] B. Iyer and M. Subramaniam, "The Strategic Value of APIs," URL: https://hbr.org/2015/01/the-strategic-value-of-apis [accessed: 2016-07-26].

[8] Deloitte, "API economy - From systems to business services," URL: http://dupress.com/articles/tech-trends-2015-what-is-api-economy/ [accessed: 2016-09-30].

[9] M. Gebhart, P. Giessler, and S. Abeck, "Challenges of the Digital Transformation in Software Engineering," International Conferences of Software Advances (ICSEA), 2016. [Online]. Available: http://www.thinkmind.org/download.php?articleid=icsea_2016_5_30_10067

[10] ISO/IEC, "Std 25010:2011 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," International Organization for Standardization, Geneva, CH, Standard, 2011.

[11] S. Newman, Building Microservices. O'Reilly Media, Incorporated, 2015.

[12] M. Gebhart, P. Giessler, P. Burkhardt, and S. Abeck, "Quality-oriented requirements engineering for agile development of restful participation service," Ninth International Conference on Software Engineering Advances (ICSEA 2014), October 2014, pp. 69 – 74.

[13] D. Garlan and M. Shaw, "An introduction to software architecture," Pittsburgh, PA, USA, Tech. Rep., 1994.

[14] L. Richardson, M. Amundsen, and S. Ruby, RESTful Web APIs. O'Reilly Media, 2013.

[15] Evans, Domain-Driven Design: Tacking Complexity In the Heart of Software. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[16] C. Pautasso and E. Wilde, "Why is the web loosely coupled? a multi-faceted metric for service design," in 18th World Wide Web Conference (WWW2009), ACM. Madrid, Spain: ACM, April 2009, pp. 911–920.

[17] J. Webber, S. Parastatidis, and I. Robinson, REST in Practice: Hypermedia and Systems Architecture. O'Reilly Media, 2010.

[18] B. Mulloy, "Web API Design - Crafting Interfaces that Developers Love," March 2012, URL: http://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf [accessed: 2015-04-09].

[19] S. Jauker, "10 Best Practices for better RESTful API," Mai 2014, URL: http://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/ [accessed: 2015-02-19].

[20] P. Papapetrou, "Rest API Best(?) Practices Reloaded," URL: http://java.dzone.com/articles/rest-api-best-practices [accessed: 2015-02-26].

[21] S. Vinoski, "RESTful Web Services Development Checklist," Internet Computing, IEEE, vol. 12, no. 6, 2008, pp. 94–96. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4670126

[22] L. Richardson and S. Ruby, Restful Web Services. O'Reilly Media, 2007.

[23] B. Burke, RESTful Java with JAX-RS 2.0. O'Reilly Media, 2013.

[24] D. D. Guinard and V. M. Trifa, Building the Web of Things With examples in Node.js and Raspberry Pi. Manning, 2016.

[25] C&M, "The system SmartCampus and the project SmartCampusbarrier-free," URL: http://cm.tm.kit.edu/smartcampus.php [accessed: 2016-09-30].

[26] M. B. Rosson and J. M. Carroll, "The human-computer interaction handbook," J. A. Jacko and A. Sears, Eds. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 2003, ch. Scenario-based Design, pp. 1032–1050. [Online]. Available: http://dl.acm.org/citation.cfm?id=772072.772137

[27] OAI, "Open API Initiative," URL: https://openapis.org/ [accessed: 2016-09-30].

[28] R. T. Fielding, "Evolve'13 - The Adobe CQ Community Technical Conference - Scrambled Eggs," 2013, URL: http://de.slideshare.net/royfielding/evolve13-keynote-scrambled-eggs [accessed: 2015-09-23].

[29] A. DuVander, "1 in 5 APIs Say "Bye XML"," 2011, URL: http://www.programmableweb.com/news/1-5-apis-say-bye-xml/2011/05/25 [accessed: 2015-02-20].

[30] Google, "Google JSON Style Guide," 2015, URL: https://google.github.io/styleguide/jsoncstyleguide.xml [accessed: 02.12.2015].

[31] OWASP, "Testing for insecure direct object references (otg-authz-004)," 2014, URL: https://www.owasp.org/index.php/Testing_for_Insecure_Direct_Object_References_(OTG-AUTHZ-004) [accessed: 2015-05-12].

[32] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach, ser. ACM Press Series. Addison-Wesley, 2000.

[33] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc 2616, hypertext transfer protocol – http/1.1," http://tools.ietf.org/html/rfc2616, 1999.

[34] M. Nottingham and R. Fielding, "Rfc 6585, additional http status codes," 2012, URL: http://tools.ietf.org/html/rfc6585 [accessed: 2015-02-18].

[35] R. T. Fielding, "REST APIs must be hypertext-driven," October 2008, URL: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven [accessed: 2015-02-20].

[36] M. Amundsen, L. Richardson, and M. W. Foster, "Application-Level Profile Semantics (ALPS) ," Tech. Rep., August 2014, URL: http://alps.io/spec/ [accessed: 2015-04-09].

[37] M. Winand, "We need tool support for keyset pagination," 2014, URL: http://use-the-index-luke.com/no-offset [accessed: 2015-12-23].

[38] W3C, "Cross-Origin Resource Sharing," URL: https://www.w3.org/TR/cors/ [accessed: 2016-09-30].

[39] Facebook, "GraphQL," URL: https://facebook.github.io/graphql/ [accessed: 2016-09-30].

[40] S. Bradner, "Rfc 2119, key words for use in rfcs to indicate requirement levels," 1997, URL: http://www.rfc-base.org/txt/rfc-2119.txt].

[41] M. Gebhart, "Query-based static analysis of web services in service-oriented architectures," International Journal on Advances in Software, 2014, pp. 136 – 147.

Evaluation and Behavioral Analysis of Place-Oriented Radio by the Measurement of Cross-Cultural Understandings

Ayaka Ito
Graduate School of Media and Governance
Keio University
Fujisawa, Japan
e-mail: ayk@sfc.keio.ac.jp

Katsuhiko Ogawa
Faculty of Environment and Information Studies
Keio University
Fujisawa, Japan
e-mail: ogw@sfc.keio.ac.jp

Abstract— **The number of foreigners who visit Japan is increasing and thus it is important to build mutual understanding with people of different cultural backgrounds. In order to enhance foreign visitors' further understanding of Japan, we propose a place-oriented Internet radio called Cross-Cultural Radio (CCR). Consequently, we proposed a new measurement, the Cross-Cultural Understanding Scale (CCUS), to validate the effectiveness of CCR, and conducted a set of evaluation experiments in Tokyo. Our experimental results illustrate that CCR can be effective in certain aspects of cross-cultural understanding. This paper aims to explore foreign visitors' concrete process of understanding by analyzing the behaviors of participants during the experiment. Additionally, the type of place-oriented contents that are suitable for enhancing cultural awareness will be discussed.**

Keywords— place orientation; Internet radio; cross-cultural understanding; measurement; evaluation experiment.

I. INTRODUCTION

Due to the great diversity in the modern world and its continuous change, defining the term "culture" is an extremely difficult activity. However, many scholars have attempted to conceptualize their understanding of culture. Nonetheless, as we claimed in our previous work [1], several aspects of culture, such as goods, feelings, actions, and words can be very specific to a particular region. They are difficult to grasp from guidebooks or simply by browsing the Internet, because in many cases these contents are provided based on visible (and generally superficial) information. It is quite easy to acquire stereotypical ideas about Japan in front of the laptop but there will never be a better experience than direct interaction with local people. This is because they provide foreigners with real cultural ideas, and thus, having a channel to boost such communication is highly important.

For instance, a keyword that seems distinctive to Japanese culture is "Kodawari," which is difficult to translate literally into English, yet "to be particular about a manner" would be the closest. Its meaning is not just to be particular, but to have a strong belief, or an excessive target on the action. Many craftspeople in Japan have "Kodawari" regarding what they create or how they become involved in the industry, and having strong pride in what they do and never compromising their work is regarded as a virtue.

Another example of a keyword is "Omotenashi," which became slightly famous after Japan's presentation to host the Olympics in 2020. "Omotenashi" means to treat everyone sincerely and warmheartedly, whether or not that person is a customer, a guest, a family member, or an acquaintance. The core of this concept is to express consideration and respect to others. This act would also require the person to understand the atmosphere, feel the mood and invisible energy, which is wrapped around the occasion or person. Ultimately, it does not mean entertaining the person or achieving any kind of self-satisfaction, but to quickly perceive the person's needs, desires, and overall mood, and entertain the person accordingly with a warm heart.

Damasio [2] claimed, "culture is a regulator of human life and identity." As people's mobility has increased, so has the number of foreigners who visit Japan [3], and recognizing diversity to build cross-cultural understanding is becoming a matter of great interest in the country. We must be aware that all foreigners are unique individuals, and we should not generalize them by nationality, ethnic group, or religion. Foreigners are visiting Japan for various purposes, such as sightseeing, studying abroad, or working. Likewise, depending on the length of their stay in Japan or their cultural background, the problems they encounter vary greatly, and there will never be a solution that is applicable to everyone. In particular, these problems faced by foreign visitors are derived from not knowing the Japanese cultural keywords previously exemplified (and there are countless others besides "Kodawari" and "Omotenashi"), or occur when the meaning of keywords conflicts with their cultural beliefs in various communicative settings. To propose a way to solve their problems individually, thus creating new media to provide foreigners with opportunities to understand Japanese culture at a deeper level, is meaningful from a cross-cultural perspective. In other words, foreign visitors' further understanding of Japanese culture will be achieved when they listen to local people's stories in a particular place, or the opinions of other foreigners who have visited a given place.

Previous literature (Yoon [4] and Bramwell [5]) has demonstrated that the effect of motivation and satisfaction is prominent in the decision of tourists to re-visit places. Alegre [6] and Ekinci et al. [7] also pointed out the eagerness of tourists to visit based on the characteristics of a place. In terms of information systems, Masuda [8] and Takagi [9] proposed a recommender system for tourists, which provides customized tour information depending on users' needs, including the use of smartphone applications. However, there is almost no research on using Internet radio specifically as a tool for building cross-cultural understanding in Japan.

In this paper, we propose a place-oriented Internet radio called CCR, which helps foreigners to recognize Japan from a cross-cultural perspective by providing place-oriented content. In addition, we created some original criteria named CCUS and conducted an evaluation experiment in Tokyo to measure the actual effectiveness of the content and CCR itself.

The paper is structured as follows: First, Section II describes the design phase of CCR, including its concept and system configuration. Secondly, a detailed explanation of CCUS measurement is offered in Section III, including background research. Section IV describes a complete set of evaluation experiments conducted in Tokyo, and Section V examines the result in a further behavioral analysis. Lastly, the conclusion and future works are mentioned in Section VI.

II. DESIGN OF CROSS-CULTURAL RADIO

A. Concept

The above-mentioned previous research, particularly Masuda and Takagi's recommender system for tourists, is designed for usage in a specific place. However, the information they provide to listeners only focuses on tourists' preferences and does not include the cultural perspective of the host country, which promotes cross-cultural understanding amongst international listeners.

Regarding the type of information available, visual material, such as detailed information on smartphones, contributes to a certain extent to obtaining a general idea about a place. Nevertheless, aural information is far superior to visual information in terms of listener flexibility, by allowing listeners to stretch their imagination regarding what they have heard. Furthermore, aural information can provide direct interaction with the place, including local people's stories or comments from other tourists. This may also be a trigger to increase international listeners' understanding of Japanese culture.

For these reasons, this paper proposes a place-oriented Internet radio called CCR as a new sound-focused media, by providing international listeners with several types of content. The detailed concept is shown in Figure 1.

CCR works in three steps. The first step consists in content design for different listeners. The second step is the listening process, which various listeners engage in, such as international tourists, study-abroad students, and employees of multinational corporations who are not yet familiar with Japanese culture. The third step is obtaining feedback from listeners, plus revision of the content. To maximize the influence of content, the preferred target of CCR is international visitors who are staying in Japan for a relatively long period of time, from a few months to years, rather than just for a couple of days, because, in general, understanding a certain culture takes time and the experience in a host country is enriched by daily life communicative settings.

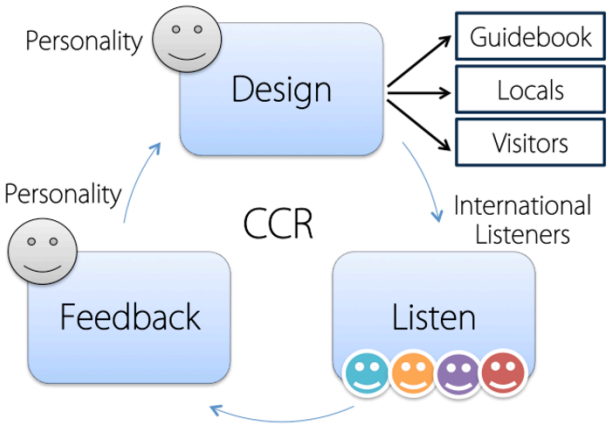

Figure 1. Concept of Cross-Cultural Radio "CCR"
(This paper only deals with Guidebook and Locals content)

Three types of content are available for international visitors: the Guidebook (audio clips from famous guidebooks such as Lonely Planet), the Locals (stories or tips from local people), and Visitors (feedback from listeners to be shared with other listeners). As a first step of the cycle, this paper deals explicitly with the content of the Guidebook and the Locals.

B. System

Previous research [10]-[13] has shown that an acceptable duration of content should be approximately 1 to 1 and a half minutes. Several companies produce audio guide players, supporting the delivery of such content as described above.

1) Selecting the Location

As CCR is designed for international visitors to Japan, the selection of a place where content is mapped is also important. In this research, Asakusa, one of the most famous and popular tourist spots in Tokyo, was selected because it has a rich cultural heritage, including Japanese traditional temples and shrines, as well as dining venues and souvenir shops that attract many international tourists. In addition, Asakusa is located at the heart of Tokyo and has great accessibility, which enables us to conduct fieldwork effortlessly.

2) "Guidebook" Content

For the guidebook content, several tips about accommodation, introduction to restaurants, and explanation of famous architecture were selected from Lonely Planet Tokyo [14] and recorded using voice synthesis software (Figure 2).

> "Asakusa Engei-hall"
> Have you ever seen standup comedy in your country? If you want to experience Japanese traditional comedy performance, here is the place. This is called Asakusa Engei-hall, and provides humorous speech by classic rakugo speakers. The audience also enjoys stage arts unique to the theater, including the paper cutout and funny music played using a carpenter's tool.

Figure 2. Example of Guidebook content

3) "Locals" Content

For the locals content, several interviews with locals were conducted in Japanese and stories related to their daily lives in Asakusa were selected. Each story was translated into English and supplementary explanation of cultural activities was added if necessary (Figure 3).

> "Future of Asakusa"
> Before World War 2, Asakusa was one of the most energetic, cutting edge cities in Japan. But unfortunately nowadays it has been overtaken by other big cities like Roppongi or Shinjuku. She feels that to revitalize Asakusa as a vivid city, collaboration with the local community is important, not just bringing lots of tourists from outside. Using social networking services can be one way; so the young generation helps older shop owners to introduce these up-to-date technologies into traditional Japanese shops.

Figure 3. Example of Locals content

4) Mapping content into CCR

Figure 4. CCR can be accessed via a QR code

The audio clips are stored on the website, and linked to icons using JavaScript code. When the user clicks on an icon, the associated audio clip is played. The website can be accessed by URL [15] or by using the QR code shown in Figure 4.

III. MEASUREMENT CCUS

To validate the credibility of CCR, an evaluation process with appropriate criteria is essential. Since CCR has a unique concept, inventing a new and suitable measurement tool is more realistic than using conventional criteria without localization. Related literature about measurement design and cross-cultural adjustment has been demonstrated by Cui & Awa [16], and Yellen [17]. Ten dimensions of cross-cultural understanding have been determined, which are:

A. Mobility

According to Benson [18], an individual's ability to find his/her way around in a foreign place is one of the most important dimensions of cross-cultural understanding. Knowing the local geography and usage of public transportation systems are two potential items for this dimension. It also includes the ability to ask for directions when one is uncertain, as well as the usage of appropriate tools, such as map applications on a smartphone.

B. Food/Diet

Although food allergies are not addressed here, this dimension involves being open-minded about trying new foods. Accepting foreign food and culinary manners cannot be omitted when understanding a certain culture, and for many people eating food is a major aspect of cultural exchange [19].

C. Flexibility

As Hofstede defined "uncertainty avoidance" in his prominent work [20], people from any cultural background may face culture shock to a certain extent, and may attempt to escape from that anxiety. Being flexible, patient, and tolerant of such uncertain activity or unexpected cultural norms is one dimension.

D. Knowledge

Whether one accepts it or not, acknowledgment of the host culture is an essential aspect of cross-cultural understanding. In terms of socially appropriate behaviors, host country nationals have certain expectations as to how foreigners in their country should behave, and this includes avoiding offensive actions. Webb et al.'s unobtrusive measure [21] could be useful in this regard.

E. Language Skills

This dimension appears consistently in the literature as a core criterion of mutual understanding [22]-[24]. However, we should be aware that when cultural adaptation or acculturation occurs, an adapted individual will learn the

language, but an individual who learns the language may or may not adapt.

F. Interaction

The nature and frequency of interactions with host country individuals is an indication of an individual's level of cross-cultural understanding [25]. This involves one's ability to initiate interaction, as well as the extent of one's eagerness to communicate with Japanese people, regardless of language ability.

G. Awareness of Cultural differnce

A question such as "to what extent are you aware that Japanese culture/society is different from yours?" is asked in this dimension. Recognition of cultural difference from one's own culture is a starting point to build mutual understanding in any circumstances [26].

H. Nonverbal Communication

In addition to language, there are a variety of ways to communicate nonverbally. Understanding visible gestures and appreciating personal space are some of them [27]. Also, having a reasonable repertoire of "communicative currency" may be useful as a criterion dimension.

I. Respect

Being interested in the host country citizens and casual friendliness towards them should be part of cross-cultural understanding [28]. For instance, willingness to participate in activities distinctive to the host country will increase fundamental respect for others and might lead to an appreciation of one's current state.

J. Relationship

The inclination to establish and maintain relationships regardless of skills is one crucial dimension. Although this can be influenced by an individual's personal character, such as extroversion or introversion, we should be aware that every individual has his/her own pace for building relationships [29]. For instance, not all introverts are weaker at relationship building than extroverts; they often establish deeper and more stable relationships with others.

After the relevant literature was reviewed and the dimensions mentioned above were rationalized, these new criteria were named CCUS. In the evaluation phase, we measured users' scores on each dimension from 1 to 10 (Figure 5), using the self-evaluation method.

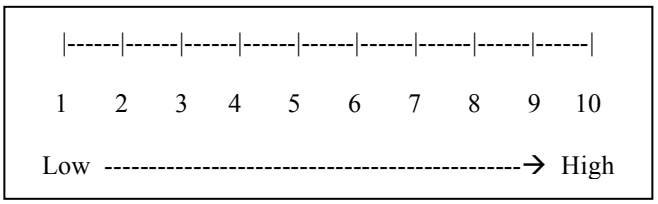

Figure 5. Scoring system of CCUS

This evaluation was conducted twice, before and after any related experiments such as fieldwork or interviews. Afterwards, the two score results were compared and discussed.

IV. EVALUATION EXPERIMENT

A. Method

Fieldwork was conducted with 12 international visitors as CCR listeners, using the same scheme to explore how the cycle of CCR works as an evaluation experiment. In order to observe various cultural exchanges, we tried to select tourists from diverse cultural backgrounds, as well as their length of stay in Japan. Fieldwork details and participants' attributes are shown below (Table 1).

TABLE I. FIELDWORK DETAILS AND PARTICIPANTS' ATTRIBUTES

Nationality / Code (xx)	**Participants' Attributes**		
	Age	***Sex***	***Date / Time***
China (CH1)	28	F	October 31st, 2015 / 11:00 – 13:00
Malaysia (ML)	23	F	October 31st, 2015 / 14:00 – 16:00
Taiwan (TW)	20	F	November 1st, 2015 / 11:00 – 13:00
Japan/Korea (JP)	22	F	November 1st, 2015 / 14:00 – 16:00
England (UK)	22	M	November 1st, 2015 / 14:00 – 16:00
Korea (KR)	18	F	November 7th, 2015 / 11:15 – 13:00
India (IN)	19	M	November 7th, 2015 / 15:00 – 16:30
Uzbekistan (UZ)	22	M	November 16th, 2015 / 11:00 – 13:00
China (CH2)	24	F	November 18th, 2015 / 10:00 – 12:00
China (CH3)	25	F	November 18th, 2015 / 10:00 – 12:00
Vietnam (VN)	24	F	November 18th, 2015 / 15:30 – 17:30
Russia (RU)	28	M	November 28th, 2015 / 15:00 – 16:30

B. Fieldwork Route

In the experiment, two fieldwork routes were prepared for participants and they were allowed to choose whichever they preferred. The routes were determined by the reference of Asakusa's rickshaw company Jidaiya [30], because their tours are recognized as a popular activity in Asakusa and in general they are successful at suggesting appropriate sightseeing routes. Route 1 (Figure 6) goes through Asakusa's most touristy district, a major temple called Senso-ji in the green area on top. As illustrated in the blue line, walking along the main street named Nakamise-dori is the so-called golden route of Asakusa sightseeing.

Figure 6. Fieldwork route 1

On the other hand, route 2 (Figure 7) is set in a rather local district, including the place for community daily life. Compared to route 1, route 2 is less crowded with tourists and may be similar to other towns, but still holds the flavor of historic Tokyo downtown.

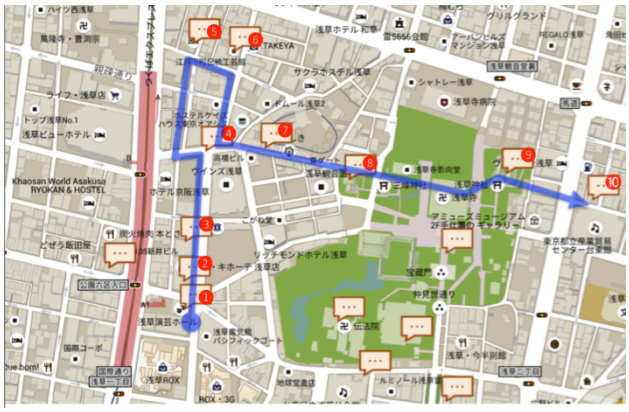

Figure 7. Fieldwork route 2

C. Contents

Figures 6 and 7 illustrate how each route has 10 mapped contents, shown as numbered speech bubble icons. We attempted to select random contents from several sources such as guidebooks and original conversation extracted from interviews with local people, and placed them as equal a distance apart as possible on the map. Content *a)* mapped on route 1 is specified as "C1a" for instance, and brief explanations of all contents are as follows:

1) Route 1

a) People who visit Senso-ji temple (C1a)

This is "Locals" content. The local fabric shop owner (henceforth Mr. M) talks about the people who visit Senso-ji temple for various reasons, such as sightseeing, religious visits, and souvenir hunting. After he talks, the English translation follows.

b) Kaminari-mon of Senso-ji temple (C1b)

This is "Guidebook" content. Kaminari-mon (Thunder gate), the entrance of Senso-ji and one of the most famous gates in Japan at the approach to a temple, is explained.

c) Tohoku earthquake and tsunami in 2011 (C1c)

This is "Locals" content. Mr. M discusses what his and other shops in the arcade were like on the day following the Tohoku earthquake and tsunami. He also talks about his experience of electricity power-saving related to the Fukushima Daiichi Power Plant Nuclear Disaster [31].

d) Asakusa visitors about a half century ago (C1d)

This is "Locals" content. Mr. M talks about his shop customers from all over the world. During the previous Tokyo Olympics in 1964, he remembers how many international visitors from Eastern Europe and the Soviet Union came to try Japanese handmade fabric, and says they were widely accepted.

e) Daikoku-ya restaurant (C1e)

This is "Guidebook" content. Daikoku-ya, the famous tempura place is recommended for its menu.

f) Senso-ji Kindergarten (C1f)

This is "Locals" content. Mr. M explains that there is a kindergarten in Senso-ji premises, and as an alumnus approximately 60 years ago he recalls his experience of the Japanese cultural event "mamemaki (bean throwing)" [32].

g) Demboin and its garden (C1g)

This is "Guidebook" content. Demboin, the residence of the head priest of Senso-ji for generations is described.

h) Fortune slip of Senso-ji (C1h)

This is "Guidebook" content. Senso-ji's Omikuji, a kind of fortune-telling using a small piece of paper on which one's fortune is written, is explained.

i) Asakusa shrine (C1i)

This is "Guidebook" content. At the very next premises to Senso-ji, there is a relatively large shrine called Asakusa Jinja. It is explained that the coexistence of two different religions, Buddhism and Shintoism, in the same district reflects Japanese animistic religious belief, and something that some international visitors do not fully comprehend depending on their religious or cultural beliefs.

j) Subsidy for Asakusa residents (C1j)

This is "Locals" content. As a resident who was born and raised in Asakusa, Mr. M tells of the recent subsidy policy of Asakusa city to promote the younger generation including newlyweds to settle down in the city.

2) Route 2

a) Asakusa Engei-hall (Figure 2, C2a)

This is "Guidebook" content. Asakusa Engei-hall, a Japanese comedy theatre playing traditional Rakugo and Yose performances, is explained.

b) Don Quixote (C2b)

This is "Guidebook" content. Don Quixote, or Donki for short, is a popular discount store franchise throughout Japan. Asakusa store's localized Japanese souvenir collection for international visitors is explained.

c) Asakusa Rockza (C2c)

This is "Locals" content. Asakusa Rockza is one of the oldest and most famous striptease theatres in Japan, established in the 1940s. A professional Mikoshi, which is a divine palanquin or "portable Shinto shrine" carrier (henceforth Mr. T) tells its history and how the building and performance has remained, and is accepted as the cultural heritage of Asakusa.

d) Sukerokuno-yado Sadachiyo hotel (C2d)

This is "Guidebook" content. This hotel is of the historical Edo period's ryokan type and intrigues many international tourists as well as Japanese guests. Their Japanese style hot spring service is also explained.

e) Japanese traditional craft museum (C2e)

This is "Guidebook" content. Gallery Takumi is a free admission Japanese handicraft museum, which contains many types of craftsmanship and in which their elaborate works are introduced.

f) Hanayashiki theme park (Figure 3, C2f)

This is "Locals" content about Hanayashiki, one of the oldest theme parks, which began its history as a botanical garden. The theme park's PR manager (henceforth Ms. H) talks about its current target users and its marketing strategy focused on local families and kids, as well as international visitors by providing interactive activities such as the "Ninja experience."

g) Future of Asakusa (C2g)

This is "Locals" content as already mentioned in Figure 3. Ms. H gives her opinion about Asakusa in comparison with other famous Tokyo cities and suggests what the local community can contribute to revitalizing the city.

h) Awashimado-hall in Senso-ji (C2h)

This is "Guidebook" content about Awashimado-hall, a small segregated garden in Senso-ji premises. A unique ritual ceremony showing animistic belief called "needle funeral" happens in this garden, which is a memorial service for broken needles to show gratitude for bringing about outstanding fabric and clothes through their works.

i) Asakusa shrine (C2i)

This is "Guidebook" contents and is shared with route 1 during the experiment.

j) Subsidy for Asakusa residents (C2j)

This is "Locals" contents and is also shared with route 1 during the experiment.

D. Instruction

In the first phase of the fieldwork, a sheet of paper giving the experiment instructions was distributed to the participants. The fieldwork route was printed and the participants were asked to walk and listen to the contents mapped on the route in numerical order. Before they began walking, they filled in the CCUS form. We observed and took pictures of participants while they were walking (Figure 16), and asked participants questions on each content, such as "what did you think about the place or object, which is explained in the content?" or "do you have any implications or comments compared to your home culture?"

The fieldwork was conducted in either English or Japanese, depending on the participant's language ability. The conversation was recorded and, after they had listened to all the content, they completed the CCUS form again.

E. Result

Figure 8 shows the average scores for each dimension. The blue line shows the results prior to the fieldwork; the red line shows the results after the fieldwork was finished and participants had listened to the "Guidebook" and "Locals" content.

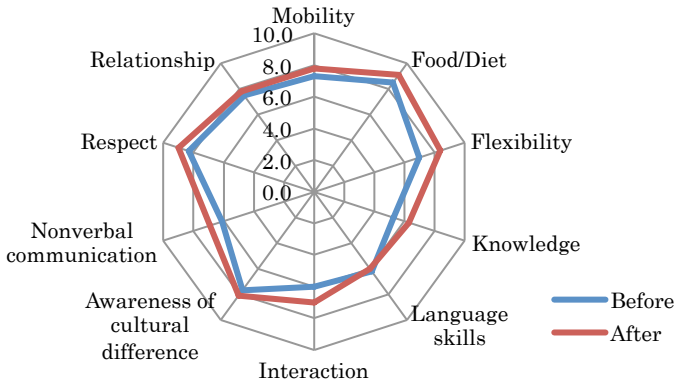

Figure 8. 12 participants' average CCUS score

According to the results, most dimensions improved slightly after the fieldwork. Specifically, "Flexibility" (+1.4pt), "Knowledge" (+1.0pt), and "Interaction" (+1.0pt) improved more than other dimensions, while "Language

skills" declined slightly (-0.2pt). However, these scores are based on only 12 international visitors' experiment results and are therefore highly dependent on participants' individual characteristics, such as cultural backgrounds, attitudes, and personalities. We believe it is rather important to conduct further behavioral analysis for individual participants in the next section.

V. Behavioral Analysis of Participants

As shown in the section V findings, CCR has enriched most aspects of the dimensions. We will have a closer look at specific participants' scores based on arbitrary choices, which recorded notable differences for "Flexibility," "Knowledge," "Interaction," and "Language skills," as well as those who formed a distinct shape of the 10 dimensions.

A. Participant CH2 (Figure 9)

CH2 is a close friend of CH3 and they participated in the evaluation experiment together. As she has never been in Asakusa before, she was a beginner tourist in a way.

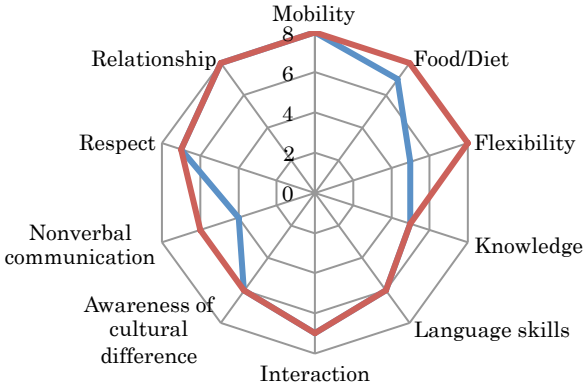

Figure 9. Participant CH2's CCUS score

Particularly after listening to "Guidebook" content about small museum shows and offering of Japanese traditional handcrafts (refer to C2e), she seemed interested in the place mentioned and took a number of pictures in front of it. She had a conversation with CH3 in Chinese and invited CH3 to go into the museum. CH2 told us they were talking about the elaborate work of Japanese craftspeople and its comparison with Chinese merchandise, including price. They mentioned that the handcrafts sold in the museum were very expensive and unfortunately they were unable to purchase any; nevertheless, they were surprised by their high quality.

Talking with a peer participant in her native language raised her satisfaction levels with "Flexibility" and "Nonverbal communication," which represents the acculturation process [33]–[35] including elimination of uncertainty about Japanese culture. It is assumed that CH2 encountered the "Kodawari" of Japanese craftspeople through their works at the museum as a tangible experience, and the content acted as a trigger for this cultural encounter.

B. Participant UZ (Figure 10)

UZ is a university student who has been studying Japanese for two years, and shows a great enthusiasm for understanding local cultures. He was particularly interested in the concept of CCR and was cooperative about participating in the evaluation experiment. He walked to the main street of Asakusa called Nakamise-dori, and after listening to content in which a local person discussed the future of Asakusa (refer to C2f or Fig. 3), he mentioned his hometown Samarkand. He said he genuinely loved his hometown but, for financial reasons, many residents are leaving the city and flowing into Toshkent, the capital of Uzbekistan; he feels sad about this. He wishes the people in Samarkand would love their city just as Asakusa locals do. Obviously, he felt some kinship with the Japanese people and had cultivated an affinity toward Japanese culture.

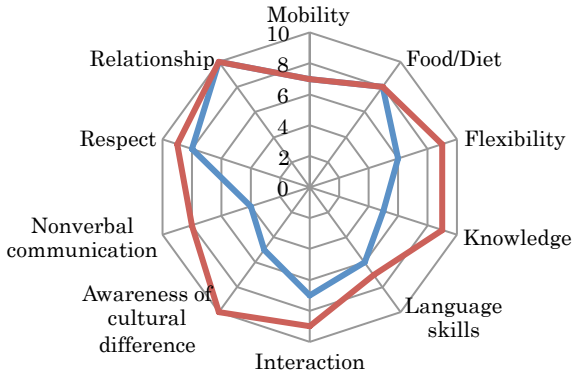

Figure 10. Participant UZ's CCUS score

He commented, "it was fun and I learned some internal/external factors of Japan, especially Asakusa city's culture and society." What he implies to be internal and external factors regard the context of both content, the Guidebook, and the Locals. Internal factors are invisible cultural aspects such as Asakusa locals' attitudes or value for the place, in relation to his radical improvement of "Awareness of cultural difference." In contrast, external factors are attainable by information input, corresponding to "Knowledge." The synthesis of these noticeable two dimensions has appeared as the improvement of "Flexibility."

C. Participant JP (Figure 11)

JP is a friend of UK and they participated in the experiment together. Although born in Japan and a Japanese citizen, she has an international background. She is half Korean and was raised in Hawaii. After listening to "Locals" content about the founding story of Nakamise-dori and a

kindergarten nearby (refer C1f), she remembered learning phonetics during her childhood in Hawaii.

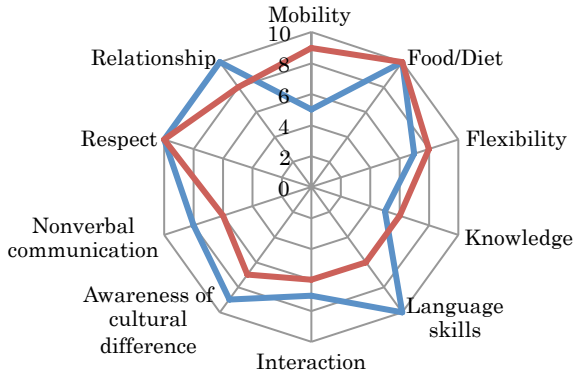

Figure 11. Participant JP's CCUS score

As she looks Asian, some of her peers automatically assumed that she did not understand any English; hence, she had a difficult time building close friendships with them. Now, English is her native language and a similar circumstance occurs when she encounters Japanese people who think she will understand Japanese perfectly whilst, in fact, she does not. JP admits "that awkward and annoying moment" frequently occurs whenever she recognizes disappointment on their faces. JP's biggest decline in "Language skills" is not unrelated with her story. On the other hand, she improved in "Mobility," as explained in her comment "now I feel more confident walking in Asakusa without GoogleMaps."

D. Participant UK (Figure 12)

The fieldwork for JP and UK was conducted in English since we wanted to encourage casual conversation between two peers, which enabled us to observe frequent cultural exchange.

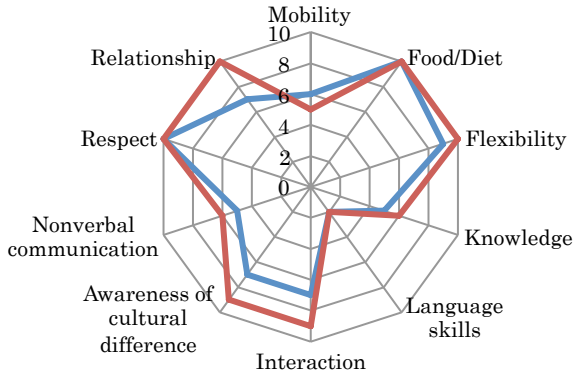

Figure 12. Participant UK's CCUS score

UK arrived in Japan approximately two months before the fieldwork, and had just started learning Japanese language and culture. According to his feedback, "Locals" content was more enjoyable than "Guidebook" although he had never visited Asakusa before. JP and UK were taking the same university courses in their study abroad and had already established a good rapport. UK is researching theories of traditional Japanese music for his master's degree, so it is reasonable to assume that he is more interested in Japanese culture than most other international visitors.

After listening to "Locals" content about subsidy policy to promote the movement of newlyweds into Asakusa city due to the population decline, particularly of the young generation (refer C1j), he commented, "Here is very packed and I didn't even know that (the population drop) was an issue. As for in Britain, honestly I really don't know (about the government policy)." He may have felt sympathy with the local people talking about the city's problem and a possible solution, which might have influenced his improvement in "Relationship," "Awareness of cultural difference," and "Interaction." The listening experience gave him recognition of Japanese culture to a certain extent.

E. Participant IN (Figure 13)

IN is a university freshman and quite new in Japan, as he arrived in Tokyo approximately two months before the fieldwork. His family has been working in Japan for a while, and he came to live with them and to pursue his academic career. He had never been in Asakusa before, and since his Japanese is still at beginner level, his fieldwork was conducted in English.

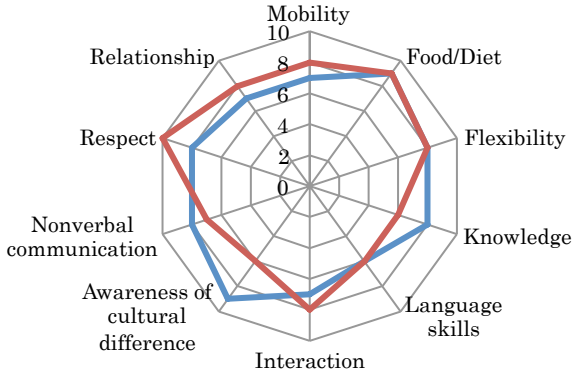

Figure 13. Participant IN's CCUS score

After listening to the content about Asakusa Engei-hall (refer C2a), he was asked if he would be interested in watching such a Japanese traditional comedy performance if they were available in a more understandable language (not only in Japanese, but with an English audio guide for instance). He answered, "Honestly, I'm not interested, I

never wanted to. Because I wouldn't even understand, as it's connected to the very localized humor. That's why I never really had the motivation (to go to a Japanese comedy performance)."

When we meet international visitors to Japan, they normally have some type of positive motivation, such as interest in Japanese language or culture, and many are open to knowing new things. IN's straightforward remark is noteworthy, because what differentiates him from other internationals is that he is very truthful in his attitude toward Japanese culture. His initial motivation for visiting Japan, which is that he simply followed his family, may be relevant. This gives us the insight that CCR might not contribute to those who already have a fixed impression of Japanese culture, and the contents will not be sufficiently strong to change their attitudes. IN's decrease in certain dimensions, particularly his 3pt drop in "Awareness of cultural difference" seems to prove this hypothesis.

F. Participant ML (Figure 14)

ML is a senior university student who has been in Japan for a few years. Although she speaks fluent Japanese, she preferred to conduct the fieldwork in English, as it is still a better language of communication for her.

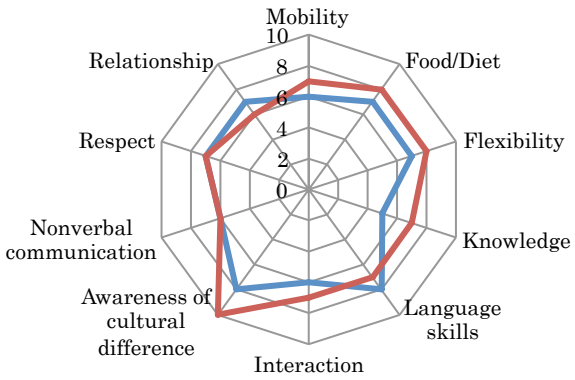

Figure 14. Participant ML's CCUS score

After listening to the content about when the Tohoku earthquake hit Japan in March 2011 (refer to C1c), she related her experience of university enrollment four years ago. As the Japanese university semester starts in April, she had a family discussion to persuade her mother, who was worried about radioactive contamination after the quake. Luckily, she was passionate enough to have a new endeavor in Japan and her father was very supportive. She commented, "You have to overcome the scare, or you just never learn anything," and admitted that her mindset helped her to decide to study abroad in Japan.

Looking at her CCUS score result, she has improved in terms of "Awareness of cultural difference." Presumably, listening to the contents and having a conversation reminded her of her initial motivation to study in Japan. It is also notable that similar to JP, who is an advanced Japanese speaker, ML has also dropped in terms of "Language skill."

G. Summary

As shown in the previous result, it is reasonable to say that CCR has contributed to the enhancement of some aspects of cultural understanding. In particular, some dimensions, such as "Flexibility," "Awareness of cultural difference," "Knowledge," "Interaction," and "Language skills," recorded dominant changes (both positive and negative).

1) Flexibility/Awareness of cultural difference

Most participants have improved in these dimensions. The biggest difference was made by CH2, but other participants, such as TW, CH3, and RU, also had a similar CCUS score distribution.

2) Knowledge

UZ and ML are the tractors of this dimension's increase, whereas some participants have not improved at all. These two participants had both been in Japan for a few years, and their Japanese was at an advanced level. Their listening experience, particularly of "Locals" contents, might have worked as a lecture to boost their knowledge of Japanese culture. VN also improved in this dimension.

3) Language Skills

Most participants did not change in this dimension at all, while some, who were relatively fluent in Japanese such as JP and ML, decreased in this dimension (JP dropped 3pts), which caused an average overall decrease. JP and ML's score drop will be revisited further in the next section.

4) Interaction

More than half of the participants improved in this dimension by approximately 1pt, which also reflects the average 1pt increase. CCR has an interactive characteristic to promote international listeners' "cultural exchange" by allowing them to discuss freely while walking, during and after they listened to each content.

VI. FINDINGS AND DISCUSSION

According to the overall result of the evaluation experiment conducted in Tokyo for 12 participants, it is reasonable to assume that CCR has contributed to the enhancement of several aspects, such as "Flexibility," "Awareness of cultural difference," and "Interaction," which are cultivated by listening to locals' stories, and "Knowledge" in relation to the information provided by the guidebook.

1) CCUS dimensions categorized into 4 types

From each participants' episodes found in behavioral analysis, it is reasonable to assume that CCR contributes to international listeners' awareness of some aspects of culture.

Furthermore, based on the episodes, it is possible to relate the 10 dimensions of CCUS into four categories of contribution (Figure 15). Hence observing CCUS score distribution enables us to roughly categorize each participant's mode of cultural understanding for these four types. This time, we observed 12 participants, and adding more participants will validate the effectiveness of this categorization.

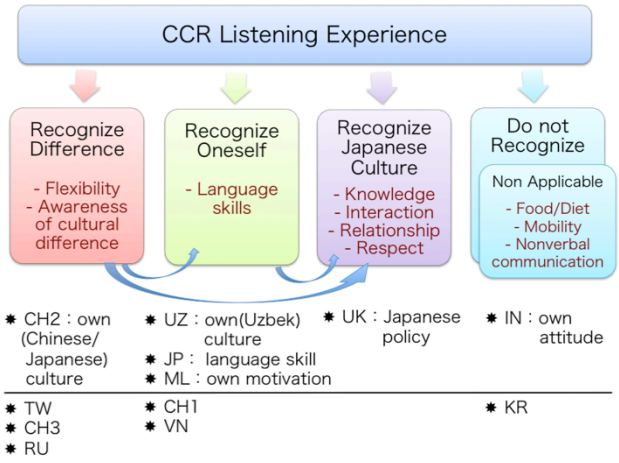

Figure 15. CCR contribution to international listeners

Through the contents listening experience, some participants realized the difference between their own culture and Japanese culture. For instance, CH2 pointed out the elaborateness of Japanese craftsmanship compared to Chinese merchandize.

Others may recognize themselves through self-reflection and conversation with peers as a next step. UZ recalled his own Usbek culture after listening to the "Locals" content. JP and ML were made more aware of their Japanese language skills by listening to interviews with local people. The discussion after the fieldwork became a trigger for ML to remember her initial motivation for studying abroad in Japan.

To a certain participant, CCR directly approaches recognition of Japanese culture or knowledge itself. UK, who was already fond of Japan, gained information about Japan's social policy after listening to a local's story.

On the other hand, the evaluation experiment has revealed that, for some participants, CCR may not contribute greatly to enhancing their cultural understanding. For example, IN clarified his lack of motivation to attend Japanese comedy performances through his honest feedback. Their personality, attitudes, and experiences will greatly influence the result.

Regarding Japanese culture itself, although more or less all participants showed recognition of it, it is assumed that "Visitors" contents will make a greater contribution in this regard.

2) "Locals" versus "Guidebook" Contents

In terms of the effectiveness of place-oriented contents, for each participant's approximately two-hour fieldwork, one outstanding episode was taken and discussed in the paper. Amongst the six participants individually analyzed in the paper, five were "Locals" content and 1 was "Guidebook" content. In addition, feedbacks elicited in the follow-up questionnaire after the fieldwork showed that "Locals" contents were more enjoyable for participants to listen to. More consultation is required for the result; however, we may rationalize that "Locals" content is more effective than "Guidebook" content for cultural understanding in this setting specifically.

VII. Conclusion and Future Studies

In this paper, a place-oriented Internet radio called CCR was proposed by providing three types of content, "Guidebook," "Locals," and "Visitors," which give an idea of real cultural aspects of Japan, represented as "cultural keywords." The paper explicitly deals with the "Guidebook" and "Locals" content. To validate the effectiveness of these unique media, we also proposed CCUS as a new criteria set to measure the level of cross-cultural understanding. The evaluation experiment and subsequent behavioral analysis of individual participants illustrated that CCR can be effective for certain international visitors' cultural understanding. Furthermore, place-oriented contents, including locals' real voices, focused on their attitudes or values, are found to be more enjoyable to listen to.

For future work, we will develop a variety of place-oriented content and add more participants to the evaluation experiment so that CCUS will be more reliable. Additionally, the introduction of "Visitors" content to encourage listeners' self/mutual reflection with other listeners is required, to compare with conventional "Locals" and "Guidebook" content. This time, we chose Asakusa for a fieldwork location as one of Tokyo's most touristy cities; however, we may need to investigate if the same scheme can be applied not only in Asakusa but also in different cities, or if it significantly varies depending on the characteristics of the place. In addition, one possibility of the evaluation experiment is to employ Japanese tourists as participants for comparative research.

References

[1] A. Ito and K, Ogawa, "Design and evaluation of place oriented internet radio by the measurement of cross-cultural understandings," IARIA ICDS 2016 conference proceedings, 2015, pp. 49-55.

[2] R. Damasio, "Self comes to mind: Constructing the concious mind," New York, Pantheon, 2010.

[3] Japanese Ministry of Justice. Retrieved on March 12th 2016 from: http://www.moj.go.jp/ENGLISH/index.html

[4] Y. Yoon and M. Uysal, "An examination of the effects of motivation and satisfaction on destination loyalty - a structural model," Tourism Management, 2005, vol. 26(1), pp. 45-56.

[5] B. Bramwell, "User satisfaction and product development in urban tourism," Tourism Management, 1998, vol. 19(1), pp. 35-47.

[6] J. Alegre and M. Cladera, "Repeat visitation in mature sun and sand holiday destinations," Journal of Travel Research, 2006, vol. 44(3), pp. 288-297.

[7] Y. Ekinci, M. Riley, and J. Chen, "A review of comparisons used in service quality and customer satisfaction studies: Emerging issues for hospitality and tourism research," Tourism Analysis, 2001, vol. 5(2) pp. 197-202.

[8] M. Masuda, T. Izumi, and Y. Nakatani, "A system that promotes repeat tourists by making sightseeing unfinished," Human Interface Society, 2012, vol. 14(3), pp. 259-270.

[9] S. Takagi, M. Masuda, and Y. Nakatani, "Tour navigation system using landmarks customized by personal preferences," Information Processing Society of Japan, 2012, vol. 3, pp. 305-306.

[10] M. Hatala and R. Wakkary, "Ontology-based user modeling in an augumented audio reality system for museums," User Modeling and User-Adapted Interaction, 2005, vol. 15(3), pp. 339-380.

[11] S. Bodker, "Through the interface: a human activity approach to user interface design," Malwah, NJ: Lawrence Erlbaum Associates, Inc, 1990.

[12] D. Dean, "Museum exhibition: theory and practice," London, Routledge, 1994.

[13] C. Hummels and A. Helm, "ISH and the search for resonant tangible interaction," Personal Ubiquitous Computing, 2004, vol. 8(5), pp. 385–388.

[14] T. Hornyak and R. Milner, "Lonely Planet Tokyo (Travel Guide) 9th Edition," NY: Lonely Planet, 2012.

[15] Cross-Cultural Radio. Retrieved on March 12th 2016 from: http://web.sfc.keio.ac.jp/~ayk/reradio/map.php

[16] G. Cui and N. Awa, "Measuring intercultural effectiveness: an integrative approach," International Journal of Intercultural Relations, 1992, vol. 16, pp. 311–328.

[17] T. Yellen, "The cross-cultural interaction inventory: development of overseas criterion measures and items that differentiate between successful and unsuccessful adjusters," National Technical Information Service, 1975, vol. 3, pp. 1-19.

[18] P. Benson, "Measuring cross-cultural adjustment: the problem of criteria" International Journal of Intercultural Relations, 1978, vol. 2(1), pp. 21–37.

[19] J. Chrzan and J. Brett, eds., "Food culture: anthropology, linguistics and food studies," NY: Berghahn Books, 2016.

[20] G. Hofstede, "Cultures and organization: software of the mind," NY: McGraw-Hill, 1991.

[21] E. Webb, D. Campbell, R. Schwartz, and L. Sechrest, "Unobtrusive measures – nonreactive research in the social sciences," IL: Rand McNally and Co, 1966.

[22] C. Kinginger, "Student mobility and identity-related language learning," Intercultural Education, 2015, vol. 26, pp.6-15.

[23] W. Chan, S. Bhatt, M. Nagami, and I. Walker, "Culture and foreign language education: Insights from research and implications for the practice," Berlin, Walter de Gruyter GmbH & Co, 2015.

[24] C. Yao and G. Zuckermann, "Language vitality and language identity – which one is more important?" Language Problems and Language Planning, 2016, vol. 40(2), pp. 163-186.

[25] K. Hwang, "Culture-inclusive theories of self and social interaction: the approach of multiple philosophical paradigms," Journal for the theory of social behaviour, 2014, vol. 45(1), pp. 40-63.

[26] D. Caganova, M. Cambal, and S. Luptakova, "Intercultural management – trend of contemporary globalized world," Elektronika ir Elektrotechnika, 2015, vol. 102(6), pp. 51-54.

[27] M. Remland, "Nonverbal communication in everyday life," NY: Sage Publications, 2016.

[28] J. Banks, "Cultural diversity and education," London, Routledge, 2015.

[29] K. Matzler, A. Strobl, N. Sauer, A. Bobovnicky, and F. Bauer, "Brand personality and culture: The role of cultural differences on the impact of brand personality perceptions on tourists' visit intentions," Tourism Management, 2016, vol. 52, pp. 507-520.

[30] Asakusa Jidaiya. Retrieved on August 23rd 2016 from: http://www.jidaiya.biz/kanko-j-eng.html

[31] BBC news press release. "Reactor breach worsens prospects". Retrieved on August 23rd 2016 from: http://www.bbc.com/news/science-environment-12745186

[32] Setsubun – Bean throwing festival in Japan. Retrieved on August 23rd 2016 from: http://gojapan.about.com/cs/japanesefestivals/a/setsubun.htm

[33] J. Berry, "Acculturation: living successfully in two cultures," International Journal of Intercultural Relations, 2005, vol. 29(6), pp. 697-712.

[34] T. Graves, "Psychological acculturation in a tri-ethnic community," South-Western Journal of Anthropology, 1968, vol. 23, pp. 337-350.

[35] A. Ryder and L. Alden, "Is acculturation uni-dimensional or bi-dimensional?" Journal of Personality and Social Psychology, 2000, vol. 79, pp. 49-65.

Figure 16. Participants engaging in the evaluation experiment in Asakusa

A Mass Storage System for Bare PC Applications Using USBs

William V. Thompson, Hamdan Alabsi, Ramesh K. Karne, Sonjie Liang, Alexander L. Wijesinha, Rasha Almajed, and Hojin Chang
Department of Computer & Information Sciences
Towson University
Towson, MD
(wvthompson, halabsi, rkarne, sliang, awijesinha, ralmajed, hchang) @towson.edu

***Abstract*—Bare machine applications eliminate the overhead and the security vulnerabilities that are due to operating systems. This paper describes a mass storage system for bare PC applications that uses USBs. It is implemented by extending a scalable FAT32 USB file system for a bare PC. First, details of the bare PC file system including the file API, file system internals, and file operations are given. Then the architecture of the mass storage system and its design and implementation are presented. A mass storage system based on this architecture is built by using four USBs on a desktop PC. Capabilities of the mass storage system are demonstrated by storing conventional files and SQLite database files on multiple USBs. Experiments to measure raw versus conventional file system performance show a 12% improvement for writes and a 33% improvement for reads with 30 MB files. This work is a first step towards building mass storage systems to support future bare machine big data and mobile applications with improved security and performance.**

Keywords-bare machine computing; bare PC; FAT32 file system; mass storage; USB.

I. INTRODUCTION

Mass storage systems are used with Web servers, database systems, clients, and numerous applications. Media for mass storage include hard disk drives, optical drives, memory cards, and USB flash drives. A mass storage system for a conventional application typically requires the support of an operating system (OS) or kernel. This paper considers a mass storage system for bare PC applications using USBs.

Bare PC applications are based on the BMC (Bare Machine Computing) paradigm, which eliminates the vulnerabilities and overhead of an OS. In the BMC paradigm, no OS, kernel, or middleware is required to communicate between an application and the hardware, i.e., an application contains everything it needs to run on a bare machine or bare PC. The BMC application uses direct interfaces to communicate with the hardware.

Mass storage systems are associated with file systems that manage both conventional data files and raw data files. File systems provide a means for organizing and retrieving the data needed by many computer applications. Typically, they are closely tied to the underlying operating system (OS) and mass storage technology. The mass storage system for bare PC applications uses a file system that is independent of any OS or platform. Such a file system can also be used by OS-based applications.

We first describe the bare PC file system including the file API [1]. We then show how the file system can be used as a basis for the architecture, design, and implementation of a mass storage system for bare PC applications using USBs. USB flash drives are ideal for file systems and mass storage in bare machine computing as they are inexpensive and able to store increasing amounts of data. We also demonstrate the capabilities of a bare PC mass storage system consisting of four USBs on a desktop PC by storing conventional files and SQLite database files on multiple USBs. We lastly present experimental results using the bare PC mass storage system to compare raw versus conventional file performance for both writes and reads. The bare PC file system and mass storage system can be used to support future bare machine database management systems and big data applications, or Web servers and mobile applications.

The rest of the paper is organized as follows. Section II gives a brief overview of BMC applications and related work. Section III gives details of the bare PC file system and its operation. Section IV describes the system architecture for the mass storage system, and Section V presents its design and implementation. Section VI illustrates functional operation, and presents the experimental performance results. Section VII concludes the paper and suggests possibilities for future research.

II. BMC APPLICATIONS AND RELATED WORK

A BMC application suite consists of an application, the necessary protocols and drivers, and the code to boot and load the application. A hard disk is not used in a bare PC, so all the code must be stored on a removable device such as a secured USB flash drive. Currently, BMC applications run on any x86-based bare PC. Bare PC applications include a Webserver [2][3], Webmail and email servers [4][5], split protocol servers [6], server clusters [7], SIP servers and user agents [8], and peer-to-peer VoIP systems [9].

Many approaches have been used to reduce OS overhead or build high-performance systems. Some use lean kernels, while others move OS-functionality into applications. Examples include Exokernel [10], IO-Lite [11], OS-Kit [12], and Palacios and Kitten [13].

The BMC paradigm [14] eliminates the OS and uses direct interfaces to the hardware to run applications on a bare PC. This approach to computing differs from a conventional approach as there is no underlying OS to manage resources, i.e., the application programmer manages memory and schedules tasks. The application is written primarily in C/C++ (with some assembly code) and runs as an AO (Application Object) that includes its own interfaces to the hardware [16] and the necessary OS-independent device drivers.

There are many types of file system specifications such as FAT32 [17], NTFS [18] and exFAT [19]. The BMC file system currently uses FAT32 as it is simple and easy to implement. The FAT32 file system has been used for building high performance clusters [20]. The Umbrella file system, which also integrates two different types of storage devices, is an example of a mass storage system that uses USB flash drives [21]. Driver-level caching can be used to improve file system for removable storage devices [22]. However, removable storage media can be exploited through the OS [23]. Bare PC USB file systems and mass storage systems have no OS-related vulnerabilities. Since there is no OS, a USB device driver needs to be integrated with the bare PC application [24].

SQLite is a lean database management system [25]. It is self-contained, self-configured, and stand-alone (i.e., it does not require a separate client and a server). SQLite is included in Web browsers, mobile devices and embedded systems. It requires an OS and the amalgamated version has about 130K lines of code. SQLite has been transformed to run on a bare PC with no OS or kernel [26].

III. BARE PC FILE SYSTEM

The material in this section previously appeared in [1]. The bare PC USB file system depends on the USB architecture [27], USB Mass Storage Specification [28], USB Enhanced Host Controller Interface Specification [29], FAT32 standard [17], and the BMC paradigm [14]. The file system is stored on a USB along with its application. The USB layout is similar to a memory layout providing a LBA (Linear Block Addressing) scheme. That is, a USB address map is similar to a memory map. However, a USB is accessed with sector numbers that are directly mapped to memory addresses. It uses SCSI (small Computer System Interface) commands [30] that are encapsulated in USB commands. Thus, a bare PC USB driver that works with this file system is needed [24]. The FAT32 standard is complex and has a variety of options that are needed for an OS based system as it is required to work with many application environments. The FAT32 options implemented in this system and the file API are designed for bare PC applications.

In [31], the design of a lean USB file system for bare PC applications was discussed and an initial version of the file system was built and tested. However, the file system was not easy to modify or use with existing bare PC applications. The rest of this section describes the implementation of an enhanced USB file system with a simple file API for bare PC applications.

A. File API

In a bare PC application, code for data and the file system reside on the same USB. In addition to the application as noted above, the USB has the boot code and loader in a separate executable, which enables the bare PC to be booted from the USB. The application suite (consisting of one or more end-user applications) is a self-contained AO that encapsulates all the needed code for execution as a single entity. For example, a Webmail server, SQLite database and the file system can all be part of one AO. Since no centralized kernel or OS runs in the machine, the AO programmer controls the execution of the application on the machine. When an AO runs, no other applications are running in the machine. After the AO runs, no trace of its execution remains.

An overview of the USB file system for bare PC applications is shown in Fig. 1. The simple API for the file system consists of five functions to support bare PC applications. These are (1) createFile(), (2) deleteFile(), (3) resizeFile(), (4) flushFile() and (5) flushAll(). These functions provide all the necessary interfaces to create and use files in bare PC applications. The fileObj (class) uses a fileTable data structure to manage and control the file system. A given API call in turn interfaces with the USBO object, which is the bare PC device driver for the USB [24]. This device driver has many interfaces to communicate directly with the host controller (HC). The HC interfaces with USB device using low-level USB commands.

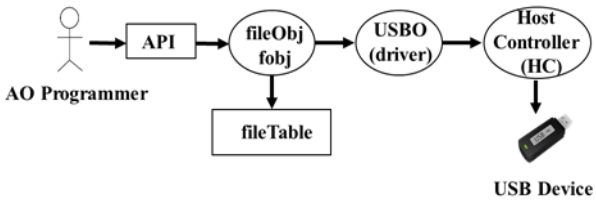

Figure 1. Bare machine USB file system.

Fig. 2 lists the file API functions, and Fig. 3 shows an example of their usage. The parameters for the createFile() function are file name (fn), memory address pointer (saddr), file size (size) and file attributes (attr); it returns a file handle (h).

```
h= createFile(fn, saddr, size, attr)
deleteFile (h)
resizeFile (h, size)
flushFile (h)
flushAll ()
```

Figure 2. File API functions.

The file handle is the index value of the file in the fileTable structure, which has all the control information of a file. This approach considerably simplifies file system design as it can be used as a direct index into the fileTable without the need

for searching. The deleteFile(h) function uses the file handle to delete a file. When a file is deleted, it simply makes a mark in the fileTable structure and its related structures such as the root directory and FAT (File Allocation Table).

```
char *ptr;
char *readArray;
FileObj fobj;
h = fobj.createFile(fileName,
  &startAddress, &fileSize, attr);
ptr = (char *)startAddress;
for(i = 0; i < fileSize; i++)
   ptr[i] = 0; //write to file
for(i = 0; i < fileSize; i++)
   readArray[i] = ptr[i]; //read from file
fobj.flushFile(h);
```

Figure 3. File API usage.

The resizeFile() function is used to increase or decrease a previously allocated file size. Thus, an AO programmer needs to keep track of the growth of a file from within the application. The flushFile() function will update the USB mass storage device from its related data structures and memory data. An AO programmer has to call this function periodically or at the end of the program to write files to persistent storage. The flushAll() interface is used to flush all files and related structures onto the USB drive. Note that the programmer gets a file address, uses it as standard memory (similar to memory mapped files), and manages the memory to read and write to a file. There is no need for a read and write API in this file system. All standard file IO operations are reduced to the list shown in Fig. 2.

A significant difference between the bare PC file system and a conventional OS-based file system is that an AO programmer directly controls the USB device through the API. That is, a user program directly communicates with the hardware without using an OS, kernel or intermediary software. For instance, the createFile() function invokes the fileObj function, which in turn invokes the USBO function. The latter then calls the HC low-level functions. In this approach, an API call runs as a single thread of execution without the intervention of any other tasks. Thus, writing a bare PC application is different from writing conventional programs as there is no kernel or centralized program running in the hardware to control the application. These applications are designed to run as self-controlled, self- managed and self-executable entities. In addition, the application code does not depend on any external software or modules since it is created as a single monolithic executable.

B. File System Internals

Building a USB file system for bare PC applications is challenging. The system involves several components and interfaces, and it is necessary to map the USB specifications to work with the memory layout in a bare PC application and the bare machine programming paradigm. Details of file system internals are provided in this section to illustrate the approach.

1) USB Parameters: Each USB has its own parameters depending on the vendor, size and other attributes. Some parameters shown in Fig. 4 are used for identification and laying out the USB memory map. These parameters are analogous to a schema in a database system and are located in the 0th sector.

```
GetReservedSectors() 0xe - 0xf (0x0236)
GetNumOfFats() 0x10 (02)
GetNumOfSectorsPerFat() 0x24 - 0x27 (0x0ee5)
GetSectorsPerCluster() 0x0d (08)
GetNumOfSectorsInPart() 0x20 -0x23 (0x003bafff)
GetClusterOfStartRootDir() 0x2c - 0x2f (02)
GetNumOfClustersInRootDir() (third entry in FAT, 04)
GetFATEntryPoint()
GetDirectoryEntryPoint()
```

Figure 4. USB parameters.

2) USB and Memory Layout: Fig. 5 displays the USB layout for a typical file system with 2GB mass storage.The boot sector contains many parameters as shown in Fig. 4. The reserved sectors parameter is used to calculate the start address of FAT1 table. The number of sectors per FAT defines the size of FAT1 and FAT2 tables, which are contiguous. The root directory entry follows the FAT2 table as shown in Fig. 5. The number of clusters in the root directory and number of sectors per cluster defines the starting point for the files stored in the USB. The root directory has 32 byte structures for each file on the USB. These 32 byte structures describe the characteristics of a FAT32 file system. The layout in Fig. 5 shows two files prcycle.exe and test.exe. The first file is the entry point of a program after boot and the second one is the application. Other mass storage files created by the application are located after test.exe. The bare PC file system has to manage the FAT tables, root directory and file system data.

3) Memory Map: The USB layout and its entry points are used to map these sectors to physical memory. A memory map is then drawn as shown in Fig. 6. During the boot process, the BIOS will load the boot sector at 0x7c00 and boot up the machine. This code will run and load prcycle.exe using a mini-loader. When prcycle.exe runs, it provides a menu to load and run the application (test.exe). The original boot, root directory and FATs as well as other existing files and data in the USB are also stored in memory to manage them as memory mapped files. The cache area stores all the user file data and provides direct access to the application program. In this system, the USB and memory maps are controlled by the application and not by middleware.

4) Initialization: Fig. 7 illustrates the initialization process after the bare PC starts. During initialization, existing files from the USB are read into memory and file table attributes are populated. In addition, FAT tables and other relevant parameters are read and stored in the system. If the file data size is larger than the available memory, then partial data is read as needed and the file tables are updated appropriately. A contiguous memory allocation strategy is used to manage real memory. Because the file handle serves as a direct index to the file table, the file management system is simplified.

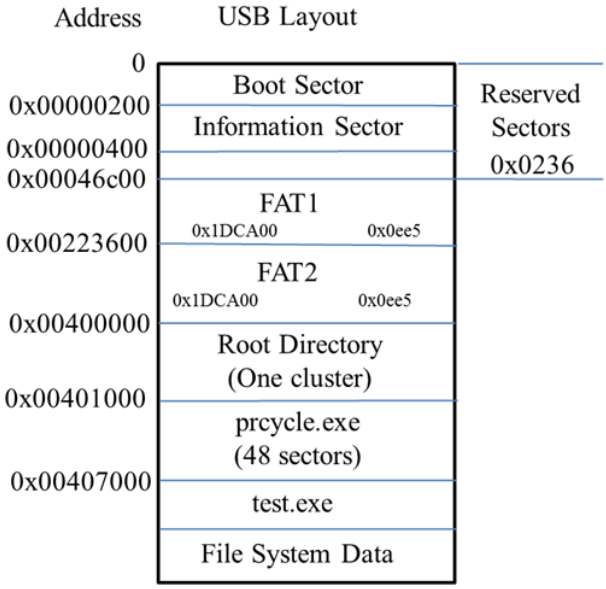

Figure 5. USB layout.

5) File Table Entry (FTE): The FTE is a 96-byte structure as shown in Fig. 8. The file name is limited to 64 bytes including name and type. 32-byte control fields are used to store the file control information needed to manage files. These attributes are derived from the root directory, FAT tables and memory map. The file index is the first entry in the FTE and it indicates the index of the file table. The index is also used as a file handle to be returned to the user for file control.

6) File Operations: The five file operations in the bare PC system use the data structures file table and device driver interfaces. The file system only covers a single directory structure. When createFile() is called, it first checks the file table for any existing file using the file name. If this file does not exist, a new file is created with the given file name and requested file size. Then an entry is made in the file table, memory is assigned, and the root directory and FAT entries are created for the file. When flushFile() is called, it updates the USB and the call returns the file handle, which is an index into the file table. Similarly, deleteFile() will delete the file from the file table and flushAll() will update the USB with all the USB data fields. The resizeFile() interface simply uses the same entry with a different memory pointer and keeps the data"as is" unless the size is reduced. When the size is reduced, the extra memory is reset. All API calls and their internals are visible to the programmer.

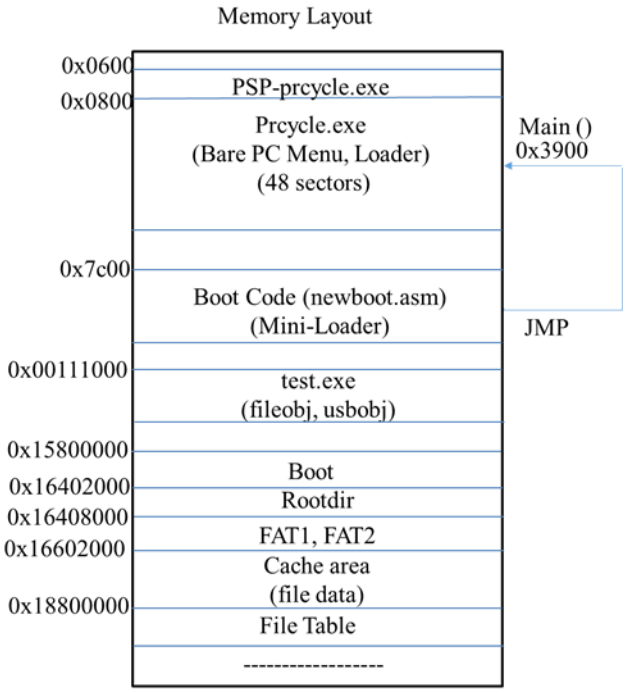

Figure 6. Memory map.

7) File Name: The file system supports both short and long file names. At present, long file names are limited to 64 characters by design since they introduce difficulties when creating the root directory and file table entries. The FAT32 root directory structure also results in complexity that affects file system implementation.

8) System Interfaces: The USB file system runs as a separate task in the bare PC AO. The AO has one main task, one receive task and many application tasks such as server threads. The main task enables plug-and-play when the USB drive is plugged into the system. Each USB slot in the PC is managed as a separate task. Tasks and threads are synonymous in bare PC applications as threads are implemented as tasks in the system. Each event in the system is treated as a single thread of execution without interruption. Thus, each file operation runs as one thread of execution.

C. Operation

The file system is written in C/C++, while the device driver code is written in C and MASM. The MASM code is 27 lines and provides two functions that read and write to control registers in the host controller. The fileObj code is 4262 lines including comments (30% of the code), and one

class definition. State transition diagrams are used to implement USB operations and their sequencing. For example, some of the state transitions occurring during the initialization process are shown in Fig. 7. The fileObj in turn invokes the USB device driver calls shown in Fig. 9.

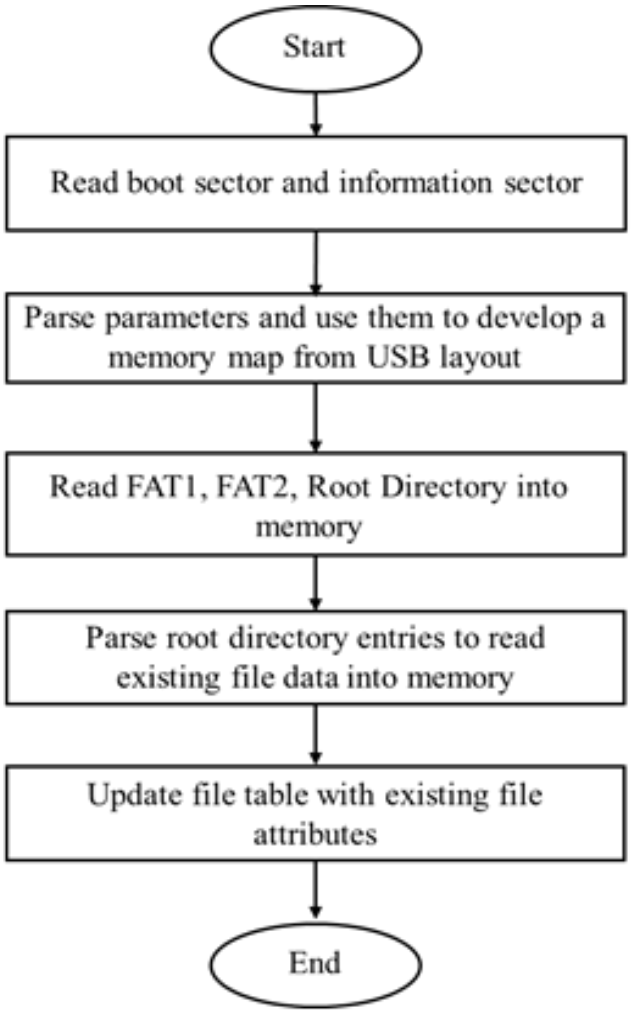

Figure 7. Initialization.

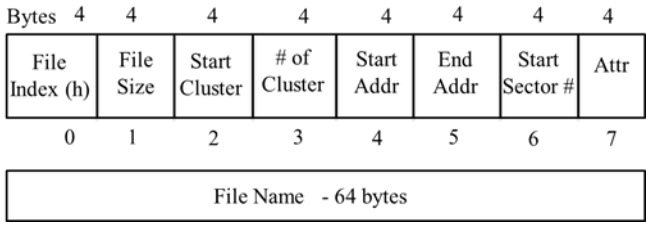

Figure 8. File Table Entry (FTE).

File operations can be done anywhere in the bare PC application. The task structure that runs in the bare PC file system is similar to that used for the bare Web servers, and runs on any Intel-based CPU that is IA32 compatible. Bare PC applications do not use a hard disk; instead, the BIOS is used to boot the system. The file system, boot code and application are stored on the same USB. A bootable USB along with its application is generated by a special tool designed for bare PC applications. The USB file system was integrated with the bare PC Web server for functional testing.

```
usbo.ResetUSBPluggedIn()
usbo.ReadUSBDesc()
usbo.SetupUSB()
usbo.ClearFeature()
usbo.WriteOp()
usbo.ReadOp()
```

Figure 9. USB operations.

The operation of the bare PC file system is demonstrated by having two existing files (prcycle.exe and test.exe) on the USB along with the boot code. Small and large files are created by the application with file sizes varying up to 100K. To demonstrate file operations, four files were created and tested as described here in addition to the two files prcycle.exe and test.exe on the USB (after the program runs, there is a total of six files on the USB). The data were read from the files and also written to them using the file API. A USB analyzer [32] was used to test and validate the file system and the driver. Fig. 10 shows a sample trace from the analyzer that illustrates reset, read descriptors, set configuration and clear. These low level USB commands are directly controlled by the programmer (they are a part of the bare PC application).

Fig. 11 shows the six files that exist on the USB displayed on the screen of a Windows PC. The four created files can be read from the Windows PC. Fig. 12 shows the file system in the bare PC root directory in memory. This directory is used to update the files until they are flushed. Fig. 13 shows the root directory entries on the USB after the program is complete. Fig. 14 is a screen shot on the bare PC showing the four files (short and long) created successfully by the system. The bare PC screen is divided into 25 rows and 8 columns to display text using video memory. This display is used by the programmer to print functional data, and for debugging. The programmer controls writing to the display directly from the bare PC application, with no interrupts used for display operations.

IV. ARCHITECTURE

We now describe a mass storage system for BMC applications by integrating the bare PC file system and the SQLite database system. This mass storage system can then be adapted for use with existing BMC applications. Fig. 15 shows the system architecture for the mass storage system.

A USB flash drive is a complete file system by itself, which consists of boot, FAT, root directory and file data. As main memories are getting cheaper and larger, it is becoming feasible to map high-capacity flash drives into main memory. Such memory maps enable easier implementation and high

performance while avoiding the need for an intricate memory management system.

The file system resident on the USB flash drive is represented by a 32-byte data structure capturing its file attributes. This is similar to a 32-byte structure in a FAT32 root directory structure. Conventional and SQLite files can all be represented with the same data structure.

Record	Summary
<Reset>	
<High-speed>	
[240 SOF]	[Frames: 266.x - 295.7]
Get Device Descriptor	Index=0 Length=64
[3 SOF]	[Frames: 296.0 - 296.2]
Get Configuration Descriptor	Index=0 Length=9
[1 SOF]	[Frame: 296.3]
Get Configuration Descriptor	Index=0 Length=255
[200 SOF]	[Frames: 296.4 - 321.3]
Set Address	Address=105
[3 SOF]	[Frames: 321.4 - 321.6]
Get Device Descriptor	Index=0 Length=18
[1 SOF]	[Frame: 321.7]
Set Configuration	Configuration=1
[2 SOF]	[Frames: 322.0 - 322.1]
Control Transfer	00
Get Device Status	
[201 SOF]	[Frames: 322.2 - 347.2]
Clear Endpoint Feature	Halt Endpoint 01 OUT
[3 SOF]	[Frames: 347.3 - 347.5]
Clear Endpoint Feature	Halt Endpoint 02 OUT
[215 SOF]	[Frames: 347.6 - 374.4]
OUT txn (NYET) [1 POLL]	55 53 42 43 44 43 42 41 00 1
[6 SOF]	[Frames: 374.5 - 375.2]
IN txn [174 POLL]	E9 B0 00 4D 53 44 4F 53 35 2
IN txn [6 POLL]	52 52 61 41 00 00 00 00 00 0
IN txn	00 00 00 00 00 00 00 00 00 0
IN txn [3 POLL]	00 00 00 00 00 00 00 00 00 0
IN txn	00 00 00 00 00 00 00 00 00 0
IN txn [2 POLL]	00 00 00 00 00 00 00 00 00 0
IN txn	EB 58 90 4D 53 44 4F 53 35 2
[1 SOF]	[Frame: 375.3]
IN txn [3 POLL]	52 52 61 41 00 00 00 00 00 0

Figure 10. Analyzer trace.

The **"file index"** field is the entry point used as an index into the root directory. The **"file size"** shows the number of bytes in the file. The**"start cluster"** and **"# of clusters"** show the starting point of the cluster on the physical media and the number of clusters needed for the entire file. Usually, a cluster is defined as 8 sectors, but a larger value can be used for larger files. The **"start addr"** and **"end addr"** fields define the start and end of the physical memory map in the system. In BMC systems, all memory is physical memory, which avoids virtual memory and paging overhead. The **"start sector#"** identifies the LBA needed to access the flash drive. The LBA scheme on the USB provides a convenient way to address it, which is similar to addressing main memory. However, the USB device needs to use SCSI (small computer system interfaces) [30] commands for access. The **"file attr"** field defines the file permissions needed to store the file in the system. Each file in the system needs one 32-byte record, which provides all the control information needed to manage the mass storage system.

Name	Date modified	Type	Size
PRCYCLE.EXE	9/17/2015 3:52 P...	Application	23 KB
test.exe	9/17/2015 3:52 P...	Application	217 KB
test1.txt	9/17/2015 3:52 P...	Text Docu...	98 KB
test2.txt	9/17/2015 3:52 P...	Text Docu...	69 KB
testing 123456.txt	9/17/2015 3:52 P...	Text Docu...	49 KB
This is a long filename.txt	9/17/2015 3:52 P...	Text Docu...	98 KB

Figure 11. Windows trace.

As a mass storage system has high capacity (order of terabytes or more), a large number of flash drives is needed. Although a desktop usually provides only a dozen USB ports, this limit can be increased to 127 ports by using a USB HUB. Each device is addressed by a unique **"portno"** in the range 1 to 127. The system architecture allows creation of a separate task for managing each port resulting in a **"taskindex"** from 0-126 (=portno–1). In a BMC application, one can define as many tasks as needed for managing ports; alternatively, a single task can be used to manage multiple USBs.

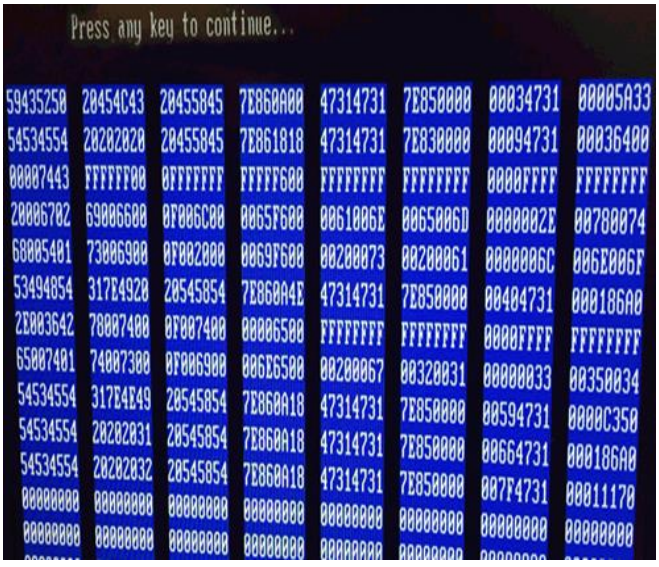

Figure 12. Bare PC root directory.

Mass storage needed for BMC applications can use up to 4 GB of real memory as this is the limit for a 32-bit address

space. If more storage is required, then the resident memory can be swapped in and out of persistent storage on to flash drives, without the need for virtual memory or paging. Raw file structures in temporary storage on the flash drives can be then used for swap space. We can also use a 64-bit CPU that can provide larger main memory capacity and a larger address space. The BMC architecture for the mass storage system is scalable and simple, and can be extended to meet the needs of new applications and future advances in technology. The design and implementation of the mass storage system based on the bare PC file system are detailed in the next two sections.

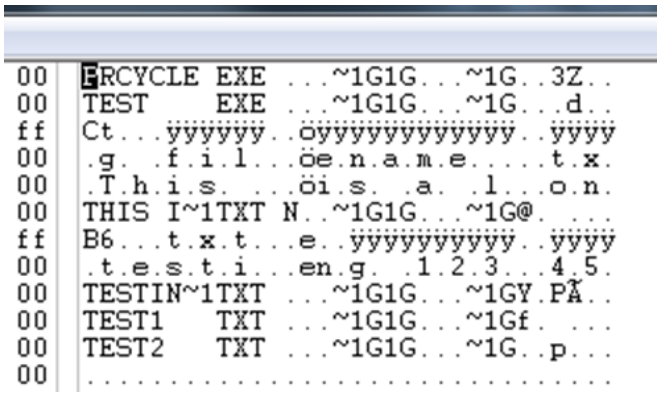

Figure 13. USB root directory.

V. DESIGN AND IMPLEMENTATION

The BMC mass storage system requires mechanisms to integrate the file system with the bare PC application. We integrated the SQLite file system with the application by using a bridge and interfaces from C to C++. The multiple USBs in a desktop need to host and manage multiple file systems in memory. The plug-play feature of USBs requires task structures that can detect the activity of flash drives and provide appropriate functionality. As the USB driver is now part of the application suite, timing-related and device-related knowledge must be integrated with the application. The device driver has to be managed by a separate file system task as it requires internal transactions and setup to perform USB operations. When SQLite is included as part of the mass storage system, it requires special handling of database functions to include the user interface and background operations.

A. Bridge between C/C++

The BMC code is written as object-oriented C++ programs with some C and Microsoft assembly code. The API to address hardware takes a path from a C++ function to a C function and then to an assembly function as needed. The C functions are used in C++ by defining them in **"extern"** blocks. This is a normal operation where C++ can call C code as is acceptable to go from strict type checking to no type checking. The SQLite code is written in C and it requires to communicate to the bare PC application code for the file system, device driver, and other function calls. Calling C++ from C to C++ violates object-oriented principles and weakens the strict type checking of C++. The bridge shown in Fig. 16 enables communication from C to C++. There are variety of ways to implement this bridge [33]. The C to C++ bridge can be summarized in four steps. In Step 1, capture the C++ member function address and store it in shared memory. In Step 2, define a C function header and a **"typedef"** for a dummy function. In Step 3, implement the C function, where the dummy function address is derived from the member function address in C++, which is stored in shared memory. In Step 4, simply define a C prototype where it is needed and call the C function. Notice that the **"typedef"** function signature must be the same as the C function call. We have defined many such functions in SQLite database to call the bare PC C++ functions to achieve the necessary integration.

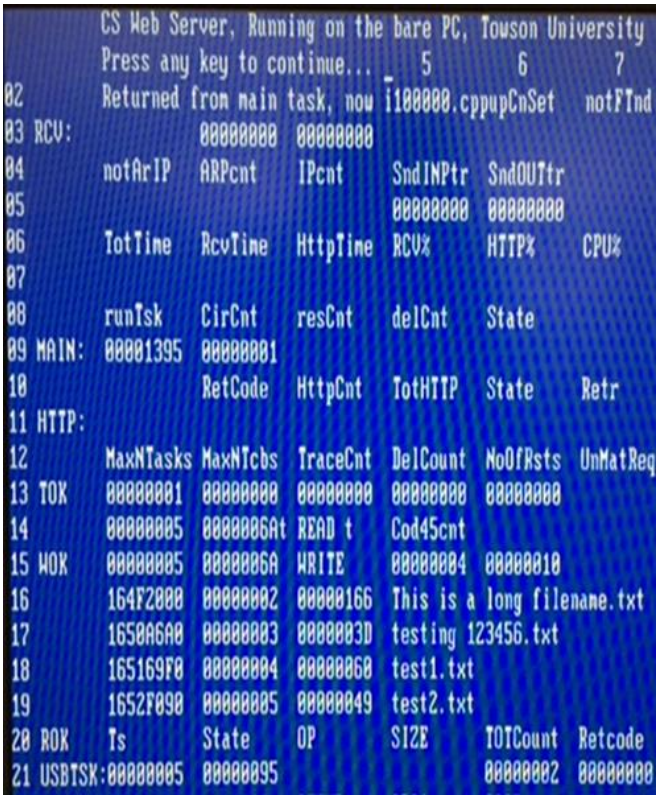

Figure 14. Bare PC screen shot.

B. USB Operation Flow Diagram

The USB operation flow diagram is shown in Fig. 17. Every time a USB is plugged in, it goes through a sequence of operations including: reset, read descriptors to capture device parameters, setup, clear feature to enable its end points, test unit ready, and read/write. The control flow for each USB has to go through these steps before it can be used for read and write operations. Note that some of these operations are SCSI commands encapsulated in the USB commands. The order of these operations are very important to make the USB operational. In addition, there are some built-in delays needed for reset operation. Determining these delay values and adjusting them as needed in the bare PC USB device driver proved to be somewhat challenging. Fig. 18 shows the design

and implementation of the approach used for managing the USB ports. There are two USB controllers in the Optiplex 960 desktop system. One controller provides four ports in the front of the machine, which are used for testing this architecture. The second controller's ports are in the back of the machine. A single task is designed to manage these four ports. These port numbers vary from 3, 4, 5, and 6. Their task indexes are one less than the port numbers. Each USB has its own file system that is resident on the flash drive. The control program is designed to check each port for its operation and functionality.

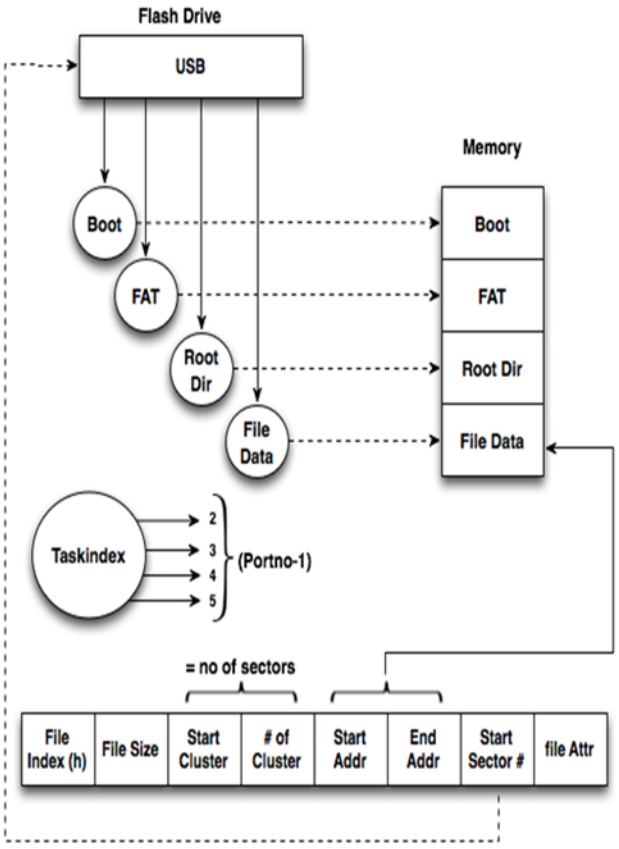

Figure 15. System architecture.

We found that the USB controller behaves differently depending on whether there is only one USB in operation or many of them plugged in. In the latter case, it requires a special reset known as mass storage reset. This is in addition to the operations as shown in Fig. 17.

A mass storage requires a sequence of USB operations test unit ready, read data, write data, clear feature, and sense data. The single USB task will go through each device and perform read or write operations as needed by an application. This task will stay in the loop until it is terminated by the user.

C. Task Structure

The task structure shown in Fig. 19 illustrates the integration of the mass storage system with the bare PC Web server application, which requires HTTP tasks. A Web server also requires resource files that are sent to clients. It may also use the SQLite database to provide dynamic content to clients. A USB task provides all USB interfaces to the user (it could be "n" tasks for "n" ports). A SQLite task manages all SQLite operations including user interfaces.

We did not address the integration of the USB file system or SQLite database files with the bare PC Webserver. However, the architecture of the mass storage system provides all the functionality needed for integration. The "Main task" is the main task that continually runs in the bare PC. When a network packet arrives, a "RCV task" runs to process the request. Similarly, when HTTP data has to be sent to a client, the "HTTP task" runs. Each task type has its own task pool created during the initialization process and kept in a stack. When a task is needed, it is popped from its appropriate task pool and placed in a circular list. The circular list tasks are processed on a first-come-first-serve basis. When a running task is complete, it will be pushed back on to its appropriate stack. When a task is waiting for an event, it is suspended and placed back in the circular list. This simple approach to managing tasks is used in the BMC paradigm. It is scalable as other types of task pools can be added in the same way.

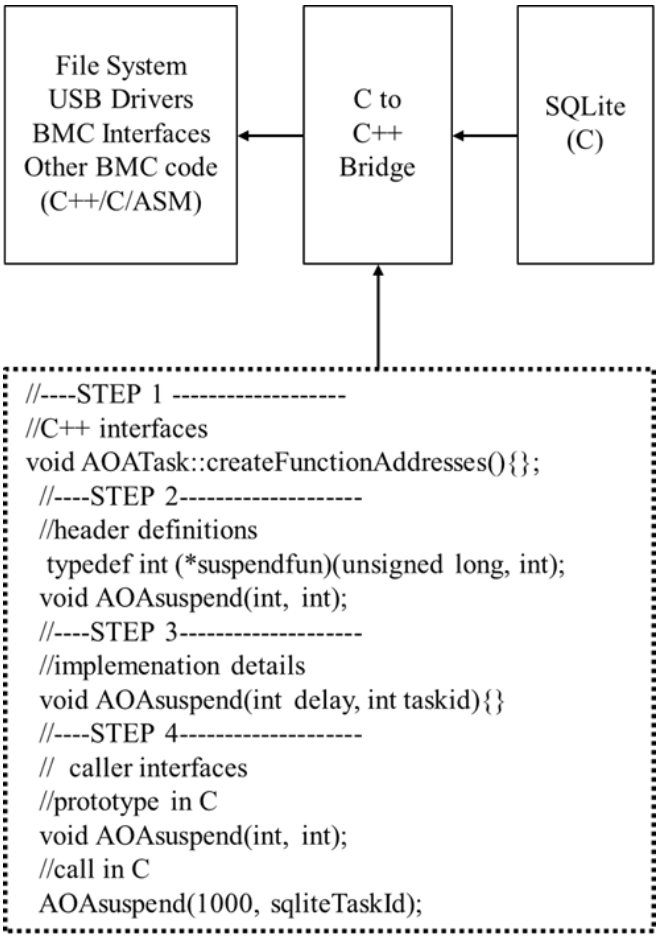

Figure 16. C to C++ bridge.

D. Class Flow Diagram

The mass storage system consists of three key class objects as shown in Fig. 20. The "fileobj" class provides a lean and efficient file API [1] [31] for bare PC applications. "USBFObj" consists of USB plug-play functions and interfaces to "fileobj". This object is managed by the USB file task. The file system API and all other interfaces can call "USBObj" interfaces for low-level USB commands such as test unit ready, read, write, sense, reset, clear feature, etc.

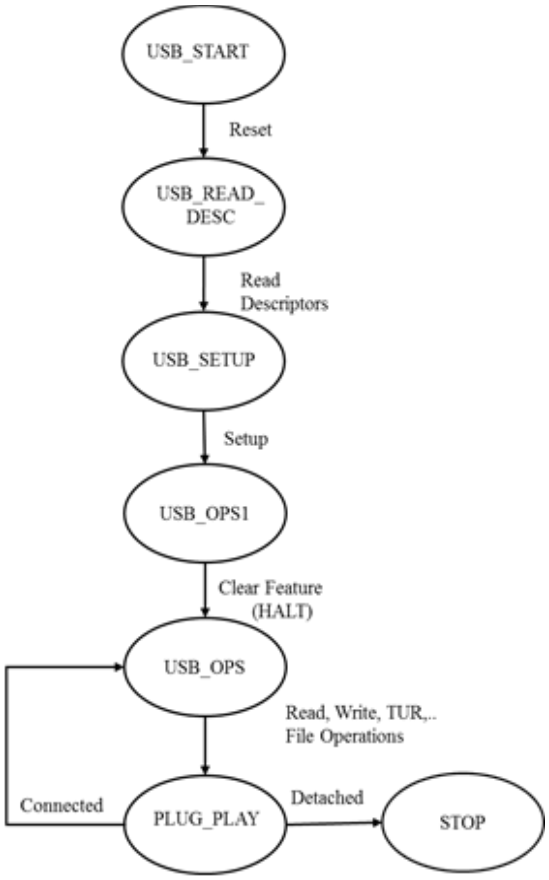

Figure 17. USB operation flow diagram.

The "USBObj", which is the USB device driver, communicates with USB controllers and devices. Each device has its own file system that is managed by the mass storage system. In the BMC paradigm, all the code needed for a given application suite is a single monolithic executable that runs by itself without the need for any external software or a kernel. Thus, a bare PC programmer has to manage all the intricacies of a given application suite. The application suite itself is independent of any external software and includes its own application and execution environment. A given bare PC application only carries the interfaces and code it needs (it avoids implementing unnecessary OS or kernel functionality).

E. Memory Map

In bare PC applications, the physical memory is managed by the application/system programmer. For a given physical memory, there is a need to organize a memory map at design time. Fig. 21 shows a typical memory map for the mass storage prototype. The first 1GB of memory is used for the Web server and other bare PC code including stack memory. The second 2GB of memory is used for USB file storage including SQLite database files.

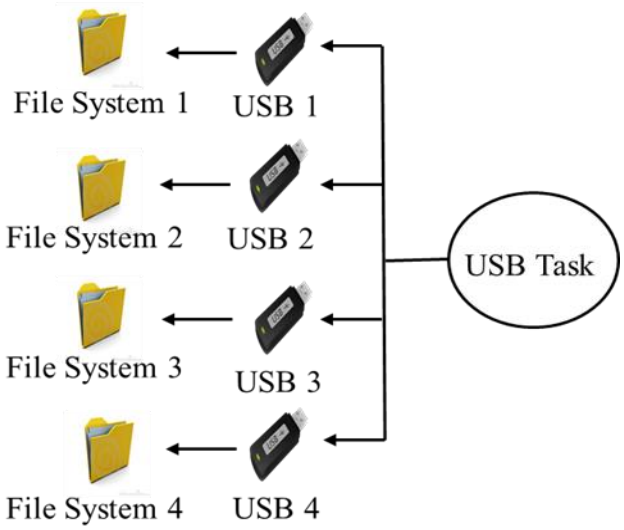

Figure 18. USB task diagram.

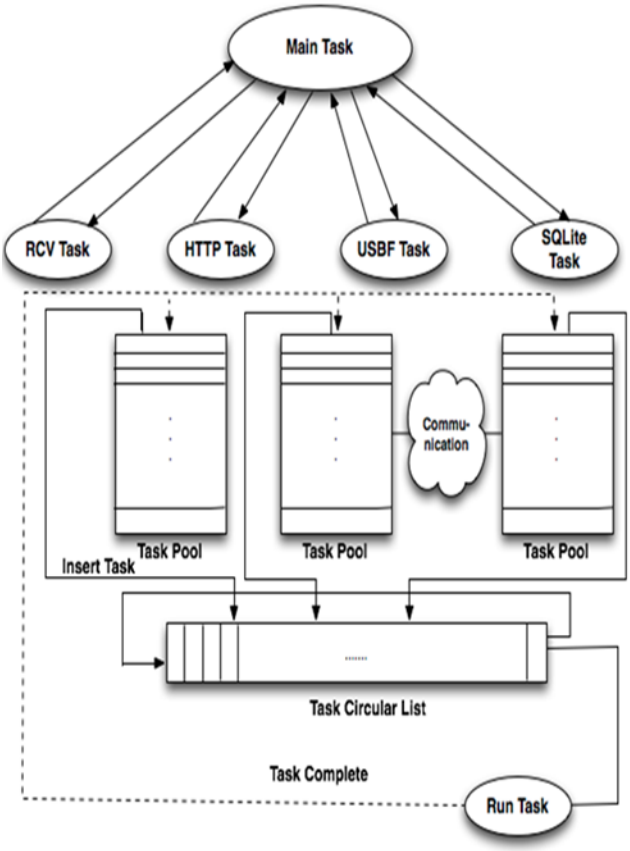

Figure 19. Task structure.

Two more GBs of memory can also be used for mass storage as needed. For four USBs, 256 MB storage is used for each USB (consisting of a total of 1 GB). 12 USBs can be mapped into a 4GB physical memory. When large memory is needed, the mass storage has to be swapped in and out of the USB devices.

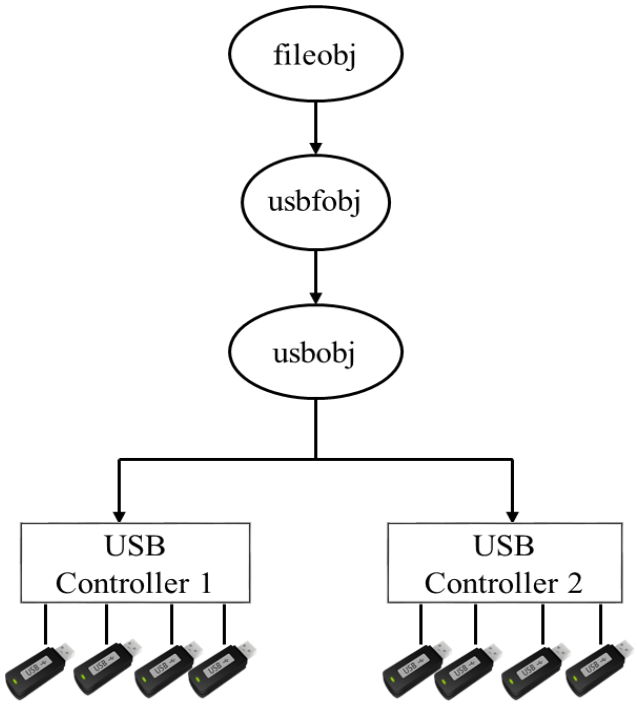

Figure 20. Class flow diagram.

F. Inter-process Communication

It is necessary to communicate between the SQLite and USB file tasks to invoke file operations such as flush, read and write. As shown in Fig. 19, the communication block is the inter-process communication element in the system. When SQLite is ready to flush, read, or write, it issues a command to the USB file task and waits synchronously until the command is complete. We use shared memory in real memory (< 1 MB) to communicate between these two processes. A single lock is used to implement this mechanism.

G. Implementation

The mass storage system was implemented in C/C++ with a small amount of assembly code for the direct hardware interfaces. The existing implementations of the bare FAT32 file system [1] [31], bare SQLite code transformation [26], and the bare Web server [2] [3] were used in building the mass storage system. The bare PC design is modular and extensible and allows new features and new applications to be added easily. The bare PC design methodology is described in [34].

VI. FUNCTIONAL OPERATION AND MEASUREMENTS

The mass storage system was tested on a Dell Optiplex 960 desktop with 2 GB Verbatim USBs. Four USBs were plugged in to the first controller and file operations were performed sequentially on the port numbers 3, 4, 5, and 6. File

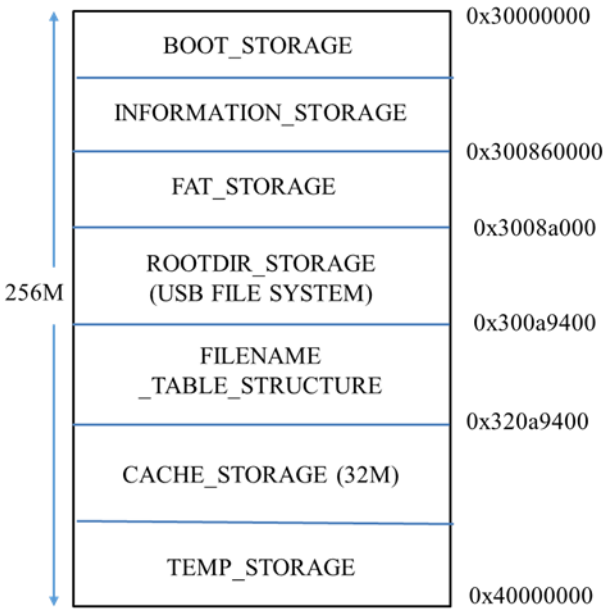

Figure 21. Memory map for each USB.

flush, read, write, and other file operations were tested to validate the mass storage system. SQLite database files were also stored on the above four USBs using four different database files. The read and write operations for regular files and the database files are same as they use the same file system. We varied USB file sizes from 1 MB to 30 MB to measure write and read timings. Fig. 22 shows write times using the file system and also using raw files.

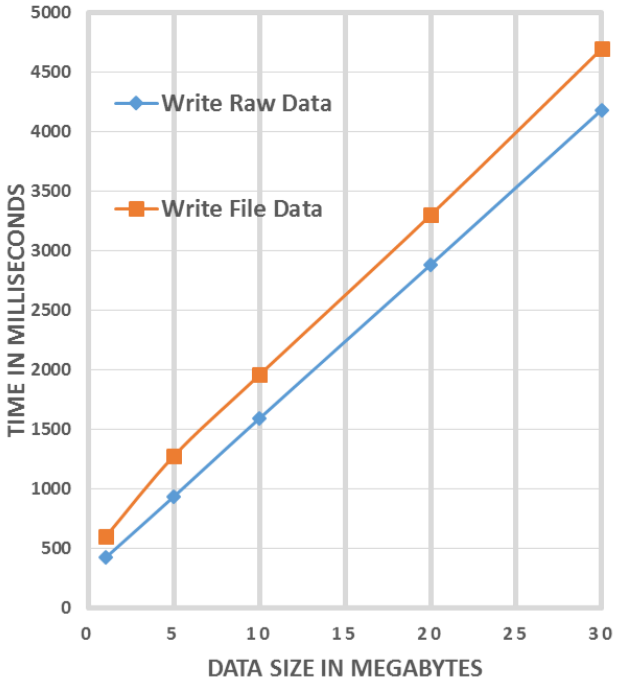

Figure 22. Write raw data/file.

A 30 MB file was written in 4.698 seconds. A 30 MB raw file (not using any file system) was written in 4.185 seconds. Thus, raw file writes can provide an approximately 12% performance improvement. This implies that bare PC applications should use raw files instead of a conventional file system to improve performance.

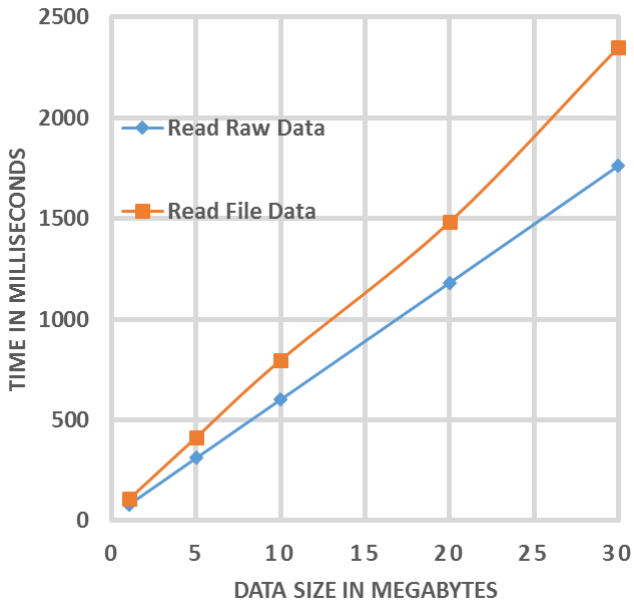

Figure 23. Read raw data/file.

In a Windows desktop, the same 30 MB file took **8.2** seconds to complete a write on the USB. This indicates that the bare PC file system has better performance than a Windows file system. In the bare PC, a 30 MB file was read in **2.351** seconds. The same file was read as a raw file in **1.762** seconds as shown in Fig. 23. This is a **33%** improvement in raw read versus a file system read. The bare PC systems are efficient and lean, and the footprint for executable files is small. The executable file size for this mass storage system is about 252 KB including the Web server and other bare PC code.

VII. CONCLUSION

We presented the architecture, design, and implementation of a mass storage system for bare PC applications. Large files and SQLite database files were used to demonstrate and test the feasibility of the system. Four USB flash drives were used to validate the design and measure basic performance. Timings for USB file write and read operations with large files were measured. Also, large raw file write and read timings were compared with the file operations. The results show that raw write and reads yield performance gains for bare PC systems. The mass storage system described in this paper is lean, simple, and scalable. It also has no OS-related vulnerabilities. As the code is simple and lean, it is easier to analyze for security flaws. The system is user-centric and runs on any x86-based architecture in bare mode.

We also presented a file API for bare PC applications. The bare PC file system enables a programmer to build and control an entire application from the top down to its USB data storage level without the need for an OS or intermediary system. This implementation can be used as a basis for extending bare PC file system capabilities in the future. The file system and mass storage system can be integrated with bare PC applications such as Web servers, Webmail/email servers, SIP servers, and VoIP clients. Future research could investigate the use of these systems for big data applications and cloud storage.

REFERENCES

[1] W. Thompson, R. K. Karne, S. Liang, A. L. Wijesinha, H. Alabsi, and H. Chang, “Implementing a USB File System for Bare PC Applications,” 12th Advanced International Conference on Telecommunication, 2016, pp. 58-63.

[2] L. He, R. K. Karne, and A. L. Wijesinha, “The design and performance of a bare PC Web server,” International Journal of Computers and Their Applications, IJCA, Vol. 15, No. 2, June 2008, pp. 100-112.

[3] L. He, R. K. Karne, A. L. Wijesinha, and A. Emdadi, “A. A Study of Bare PC Web Server Performance for Workloads with Dynamic and Static Content,” 11th IEEE International Conference on High Performance Computing and Communications (HPCC), 2009, pp. 494-499.

[4] P. Appiah-Kubi, R. K. Karne, and A. L. Wijesinha, “The Design and Performance of a Bare PC Webmail Server,” 12th IEEE International Conference on High Performance Computing and Communications, (HPCC) 2010, pp. 521-526.

[5] G. H. Ford, R. K. Karne, A. L. Wijesinha, and P. Appiah-Kubi, “The design and implementation of a bare PC email server,” 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC), 2009, pp. 480-485.

[6] B. Rawal, R. Karne, and A. L. Wijesinha, “Splitting HTTP requests on two servers,” 3rd Conference on Communication Systems and Networks (COMSNETS), 2011, pp. 1-8.

[7] B. Rawal, R. K. Karne, and A. L. Wijesinha. “Mini Web server clusters for HTTP request splitting,” IEEE Conference on High Performance, Computing and Communications (HPCC), 2011, pp. 94-100.

[8] R. Yasinovskyy, A. Alexander, A. L. Wijesinha, and R. K. Karne, “Bare PC SIP user agent implementation and performance for secure VoIP,” International Journal on Advances in Telecommunications, vol 5 no 3 & 4, 2012, pp. 111-119.

[9] G. Khaksari, A. Wijesinha, R. Karne, L. He, and S. Girumala, “A peer-to-peer bare PC VoIP application,” IEEE Consumer Communications and Networking Conference (CCNC) 2007, pp. 803-807.

[10] D. R. Engler and M.F. Kaashoek, “Exterminate all operating system abstractions,” Fifth Workshop on Hot Topics in Operating Systems,USENIX, 1995, p. 78.

[11] V. S. Pai, P. Druschel, and W. Zwaenepoel, “IO-Lite: A unified i/o buffering and caching system,” ACM Transactions on Computer Systems, Vol.18 (1), Feb. 2000, pp. 37-66.

[12] "The OS Kit Project," School of Computing, University of Utah, SaltnLake ,UT, June 2002, http://www.cs.utah.edu/flux/oskit.

[13] J. Lange et al, "Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing." 24th IEEEInternational Parallel and Distributed Processing Symposium (IPDPS), 2010, pp. 1-12.

[14] R. K. Karne, K. V. Jaganathan, N. Rosa, and T. Ahmed, "DOSC: dispersed operating system computing," 20th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA), 2005, pp. 55-61.

[15] S. Soumya, R. Guerin and K. Hosanagar, "Functionality-rich vs. Minimalist Platforms: A Two-sided Market Analysis," ACM Computer Communication Review, vol. 41, no. 5, pp. 36-43, Sept. 2011.

[16] R. K. Karne, K. V. Jaganathan, and T. Ahmed, "How to run C++ applications on a bare PC,"6th ACIS Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) 2005, pp. 50-55.

[17] Microsoft Corp, "FAT32 file system specification," http://microsoft.com/whdc/system/platform/firmware/fatgn.rnspx, 2000. [retrieved: April 8, 2016]

[18] R. Russon and Y. Fledel, "NTFS Documentation," http://dubeyko.com/development/FileSystems/NTFS/ntfsdoc.pdf. [retrieved: April 8, 2016]

[19] R. Shullich, "Reverse Engineering the Microsoft ExFAT File System,"https://www.sans.org/readingroom/whitepapers/forensics/reverse-engineering-microsoft-exfat-file-system-33274. [retrieved: April 8, 2016]

[20] M. Choi, H. Park, and J. Jeon, "Design and implementation of a FAT file system for reduced cluster switching overhead," International Conference on Multimedia and Ubiquitous Engineering, 2008, pp. 355-360.

[21] J. A. Garrison and A. L. N. Reddy, "Umbrella file system: Storage management across heterogeneous devices," ACM Transactions on Storage (TOS), Vol. 5, No. 1, Article 3, March 2009.

[22] Y. H. Chang, P. Y. Hsu, Y. F. Lu, and T. W. Kuo "A driver-layer caching policy for removable storage devices," ACM Transactions on Storage, Vol. 7, No. 1, Article 1, June 2011, p1:1-1:23.

[23] J. Larimer, "Beyond Autorun," Exploiting vulnerabilities with removable storage," 1–66, Jan. 2011. https://media.blackhat.com/bh-dc-11/Larimer/BlackHat_DC_2011_Larimer_Vulnerabiliters_w-removeable_storage-wp.pdf. [retrieved: April 8, 2016]

[24] R. K. Karne, S. Liang, A. L. Wijesinha, and P. Appiah-Kubi, "A bare PC mass storage USB device driver," International Journal of Computers and Their Applications, Vol 20, No. 1, March 2013, pp. 32-45.

[25] SQLite, http://www.sqlite.org/download.html. [retrieved: April 8, 2016]

[26] U. Okafor, R. K. Karne, A. L. Wijesinha and B. Rawal Transforming SQLITE to Run on a Bare PC," 7th International Conference on Software Paradigm Trends, 2012, pp. 311-314.

[27] Perisoft Corp, Universal serial bus specification 2.0, http://www.perisoft.net/engineer/usb_20.pdf. [retrieved: April 8, 2016]

[28] Universal serial bus mass storage class, bulk only transport, revision 1.0, 1999, http://www.usb.org [retrieved: April 8, 2016]

[29] Intel Corporation, Enhanced host controller interface specification for universal serial bus, March 2002, Rev 1, http://www.intel.com/technology/usb/download/ehci-r10.pdf [retrieved: April 8, 2016]

[30] SCSI2.0 Specifications, http://ldkelley.com/SCSI2/index.html. [retrieved: April 8, 2016]

[31] S. Liang, R. Karne, and A. L. Wijesinha, "A lean USB file system for bare machine applications," 21st Conference on Software Engineering and Data Engineering (SEDE), 2012, pp. 191-196.

[32] Total Phase Inc., USB analyzers, Beagle, http://www.totalphase.com. [retrieved: April 8, 2016]

[33] "How to mix C and C++," The C Programming Language, https://isocpp.org/wiki/faq/mixing-c-and-cpp. [retrieved: April 8, 2016]

[34] G. H. Khaksari, R. K. Karne and A. L. Wijesinha. "A Bare Machine Application Development Methodology," International Journal of Computers and Their Applications (IJCA), Vol. 19, No.1, March 2012, pp. 10-25.

Semantic Service Management and Orchestration for Adaptive and Evolving Processes

Johannes Fähndrich, Tobias Küster, and Nils Masuch

DAI-Labor, Technische Universität Berlin
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
e-mail: {johannes.faehndrich, tobias.kuester, nils.masuch}@dai-labor.de

***Abstract*—Introducing adaptiveness into service compositions allows for a next generation of services, which adapt to their context of use and posses self-* properties like self-healing and self-configuring. However, the development of adaptive and flexible services is challenging and lacks tool support. For this next step in service development there are a multitude of requirements to be met: a service discovery needs to keep the available services up-to-date, a semantic layer needs special development resources to introduce interoperability, ontologies need to be managed and merged, a service selection mechanism has to find the fitting service for a context out of a vast amount of service advertisements, and runtime components need to surveil the execution of such a service composition. In this paper, we review multiple projects, in which those components have been subject to research, and we present our own approach of a semi-automatic development methodology for adaptive service compositions and finally discuss future challenges.**

Keywords–Semantic Service Matching; Automated Service Composition; BPMN Processes; OWL-S.

I. Introduction

In recent years, the increasing digitalization of our societies has led to a vast amount of new possibilities. Many companies, administrations, and devices share their data or functionalities with others via application programming interfaces (APIs), or services, respectively. Examples are the smart home, or the transportation domain: In the first case, many different devices, such as smart meters and household appliances, are addressable and can be regulated remotely. In the second case, new services are provided digitally, such as car-sharing offers, where the user can find, reserve and unlock a nearby car via an API, and many of the actual cars or charging stations are accessible via services, as well. Those are just two examples of new services, leading towards a sophisticated *Internet of Things* (IoT), which is often eagerly anticipated to connect services across domain borders.

By introducing adaptiveness into service compositions, those services can be dynamically combined and orchestrated, allowing for a next generation of services, which adapt to their context of use and posses self-* properties like self-healing and self-configuring. However, there are some significant challenges that have to be overcome in order to exploit their potential. To begin with, there is the requirement of finding an appropriate service in the first place. Different approaches like Universal Description, Discovery and Integration (UDDI) have been proposed, but none really has made it into the market. Second, there is the need for interoperability. Since a homogeneous data environment in open, extensible platforms is unrealistic, automated mapping solutions between models or ontologies respectively are one potential approach. And finally, due to the increasing amount of services, there is a strong requirement for automatic understanding of services and their composition to value-added functionalities.

Especially for the last challenge, semantic technologies are an appropriate approach by providing structured data to machines. However, this does not come without a price. The management overhead can be immense, especially for developers not familiar with semantic technologies.

In this article, which is an updated and extended version of a paper previously presented at The Eleventh International Conference on Internet and Web Applications and Services (ICIW 2016) [1], it is our goal to develop a semantic-based service management methodology that considers the whole life-cycle of semantic services including more sophisticated algorithms for automation. More concretely, we provide development tools for model transformation, for the semantic description of services, and their deployment in order to set up a service. Furthermore, we propose how to find and match services at design-time and how to easily integrate them either to Java code or into an editor for the Business Process Model and Notation (BPMN). Based upon that we developed comprehensive matching and service composition techniques that can be used both at design-time and at runtime.

The remainder of this paper is structured as follows: At first, we motivate our case in Section II. In the following sections, we introduce the core components used for matching and planning (Section III), as well as development tools (Section IV) and a runtime environment (Section V), in which those are used. In each of those sections, we will also contrast our approach and contribution to the respective state-of-the-art. Then, in Section VI, we combine the core components, tools, and runtime to a comprehensive development method for semantic service engineering, and show how it was applied in research projects (Section VII), before we finally wrap up and conclude in Section VIII.

II. Motivation

Proper service management is highly important to build reliable and reusable software systems. Service management can be divided into several phases, whereas each of them contains requirements that are not fully met so far. As illustrated in Figure 1, the service management starts with the most fundamental part, the service engineering. Under the term service engineering we subsume the concrete specification of a service, which also contains the embedment of existing services into a new process. At this point, the challenge of finding such a service comes into play for the first time. When we think about platforms containing hundreds or thousands of

services, a comprehensive search, matching, and even planning mechanism is necessary. In our case, we try to address this challenge by providing a semantic-based service matching mechanism, which will be described in Section III. Furthermore, this feature has to be embedded into a development environment. Many approaches that provide service matching and planning stick to very specific planning languages for the whole process definition, which makes it hard to use them for daily problems. Therefore, in this paper it is a clear focus to provide an approach that relies on a specification language that is commonly used and expressive enough. After having specified the process of the service its proper declaration has to be processed. Currently, this step is neglected since most of the service management approaches do not rely on a service declaration that can be analysed and interpreted by machines. In future scenarios however, each service, even a value-added one that is already using other services, should be made available to other instances in order to enable efficient service composition. In Section IV we will provide an approach how to semantically enhance services based on a semi-automatic process. After the declaration the service has to be tested and where required the service engineering has to be launched again in order to adapt the service process. There are already lots of interpreter components that enable the step-through and validation of a process. However, an interpreter component that is integrating service matching, planning and service selection features during testing is – to the best of our knowledge – not available so far. We will present our approach for this in Section IV, as well. After a successful testing phase the developer has to deploy the service in order to make it available. Here the important issue is to transform the specified process from the Service Engineering phase into an executable language. The more transformations the tool chain supports the more flexible the service can be launched. In our case we provide different target languages as well as direct interpretation, which we describe in Section V.

Furthermore, deployed processes currently are not very flexible in practice. In our approach we aim to support the dynamic integration of services or even service chains at runtime. This leads to a highly adaptive service that can select services according to functionality and Quality-of-Service parameters. The approach of adaptive service selection at runtime is also discussed in Section III.

The described adaptability is also important for the last phase of the service management lifecycle: the service monitoring. While executing the service, another component has to surveil the status of the service. Think about a situation when a street network routing service that is being used for a multimodal routing service is immediately out of order and not usable any more. Usually, some error message might occur, that will be sent to a maintenance person that has to check why the service can not be invoked any more and has to decide how to proceed next. It is our goal to simplify this by using the before described adaptive service approach. In our concept the service composition itself is searching for an alternative service within the platform based on the semantic description of the missing service.

In summary it can be stated that in each of the service management phases there are still open challenges to fulfil in order to provide an adaptable service management methodology, that can be used for actual problems. In the following we will describe our concept of adaptive service selection and planning.

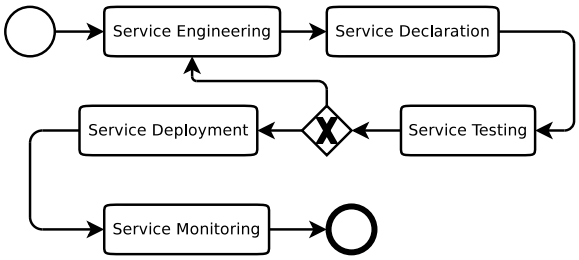

Figure 1. Basic service management lifecycle.

III. Core Components

In this section, we will have a look at the core components used for matching semantic services and for planning with those services. While those components are used in the development tools and in the runtime environment that we will introduce in the next sections, they are self-contained and can be used independently from the rest, e.g., in a different context.

A. Semantic Service Matcher

Since the beginning of research in semantic service matching, matchmakers have matured in precision and recall [2]. Thus, the focus of service matching has shifted to the integration of non-functional parameters and formal modelling of system properties. The development on the Service Matcher that had the best Normalised Discounted Cumulative Gain (NDCG) value in the last *Semantic Service Selection* contest (S3) in 2012 [2], called $SeMa^2$, has been focused on formalising and distributing the architecture of $SeMa^2$ and enabling a learning mechanism to customise the matching results to a given domain. In the following we first describe the approach of $SeMa^2$ and then discuss the current state-of-the art for service matchmaking.

1) Approach: Within this section we describe how we modelled the architecture, the matching probability, its aggregation and which parameters for the learning can be extracted. For an even more detailed discussion about $SeMa^2$, we refer to [3].

a) Architecture of a modern Service Matcher: The service matching task can be broken down into subtasks like matching the inputs of the request and the advertisement, or comparing their textual descriptions. In the $SeMa^2$ architecture each of these subtasks has been explicitly encapsulated in a so-called *expert*, which can be distributed following the agent paradigm.

As shown in Figure 2, the $SeMa^2$ consists of 33 different experts, which are dependent from each other (edges of the graph). The "Matching Result" represents the overall result of a matching request. It is also defined as an expert as it aggregates the results from the opinions of six types of experts: the text similarity expert, comparing the textual descriptions of a service; the in- and output parameter expert looking at the parameters and results of the services; the effect structure expert, evaluating the similarity of effects; the rule reasoning expert, which evaluates whether the precondition and effect rules are satisfied with the same parameters; as well as the rule structure expert, and the marker passing expert, which connects the describing ontologies through ontology matching.

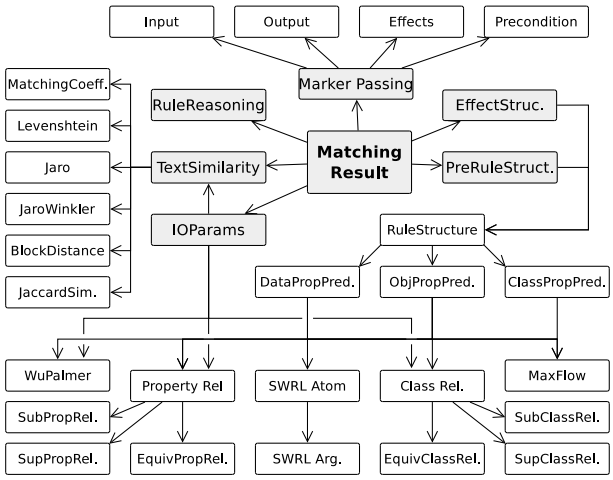

Figure 2. Expert System of the $SeMa^2$. High-level experts are composed of low-level experts, all contributing to the Matching Result.

Each expert creates an opinion about the matching of two services. These opinions are aggregated by experts which use other experts to create their opinion. Six types of experts form the top level of the aggregation. Five of those experts analyse structural and logical relatedness of the two services. The sixth expert is a semantic expert. It analyses the semantic similarity between the different aspects of the two services. For its opinion it created a semantic graph through lexical decomposition and passes markers along the edges of this graph to measure the semantic distance from, e.g., the input of one service to another. The interested reader is referred to [4] for more details. Each of those experts uses other experts to help forming its opinion, expressing the matching score of one aspect of a service. Thus, each expert encapsulates such a scoring method, which can be reused by multiple experts or extended with new scoring as the architecture evolves. The opinions of the experts are weighted due to their performance in an offline learning phase. For more details please find [3].

b) Probabilistic model of opinion: The different opinions of the experts are formalised by utilising the results of Morris [5], as probabilities $p_i(R, A)$. As an expert i observes aspects of a request R and advertisement A and calculates their distance. We can abstract this opinion as $p_i(\Theta|d)$ where Θ is the subject of interest and d are the observations. $p_i(\Theta|d)$ could be interpreted as a degree of belief of Θ observing data d. For more details see [3].

To aggregate the opinions of the different experts, an Opinion Pool is used. Here, a weighted mean of the opinions is created, for which we chose a weighted arithmetic mean called linear opinion pool [6] in a previous work [3]. This arithmetic mean has been generalised by Genest [7] to be able to use weights in the interval $[-1, 1]$ in a more general class of linear opinion pools. With this formalisation, the quality of the different aspects can be weighted during the aggregation. Choosing those weights is done during the learning phase. This selection of weights for the experts enable the matcher to adapt to the specifics of the semantic service descriptions of a special domain.

c) Learning Semantic Service Matcher: Selecting weights for each experts instances ($SeMa^2$ for now has 133 experts instances), we do not only assess the performance of the expert, but also the quality of the description of the service, the ontologies of the domain and if present specific description aspects of a domain. These interdependencies are the reason why we are unable to learn the performance of an expert in general and reuse the weights for other matching domains.

For the learning, $SeMa^2$ implements different standard learning mechanisms, reaching from genetic algorithms implemented with the Watchmaker Framework [8] to simulated annealing [9]. For the statistical evaluation the Semantic Web Service Matchmaker Evaluation Environment (SME2) tool [10] is used, calculating the NDCG of each expert and adapting its weight according to the optimisation strategy used during the learning. As a drawback, this ability to adapt to the domain makes an offline learning phase necessary, where a test collection of example services needs to be defined, including a relevance rating for the training set of service to be used by the SME2 tool.

2) State-of-the-Art: In order to optimise the result of the service matching a learning phase can be introduced to adjust the parameters of a service matchmaker to the properties of the domain. The parameters to learn depend on the service matchmaker and thus its flexibility depends on the parameters that can be observed.

In *Klusch et al.* [11] the authors introduced a formal model of defining weights for the aggregation of different similarity measures with the names w_w – similarity and w_s – structural similarity measure. The aggregation method has been learned using a Support Vector Machine (SVM) approach based on training data. The matchmaker component that invokes this approach is designed to match SA-WSDL services (Semantic Annotations for Web Service Description Language).

Klusch and *Kapahnke* [12] introduce another learning service matchmaker by extending the approach of a prior work [13] for OWL-S service descriptions (Web Ontology Language for Web-services). Here, matching results of different matching types are aggregated using a weighted mean. The authors introduce different types of matching results that are weighted. Firstly, approximated logical matching, which is divided into approximated logical plug-in and subsumed-by matching. Secondly, non-logic-based approximated matching, which are text and structural semantic similarity-based signature matching. The weights of this aggregation are also learned using a SVM. This supervised learning approach is replicated in our work, but with a different learning algorithm. The relevance set that is used to rank the matching results are reused with a genetic algorithm and a hill-climbing search.

Gmati et al. [14] use an architecture similar to ours, which uses parameters as weights to combine the results of the different matching components. This idea was published earlier in [3] with an additional learning component for the introduced weights.

To the best of our knowledge there exist only these approaches that utilise machine-learning techniques in order to cope with the challenge of aggregating service matchmaking techniques. For future research, service matchers will also have to be extended with the capability of comparing QoS parameters and Service Level Agreements of services.

B. Semantic Services Planner

During the creation of a service composition, automation might help the developer to save time. The capability of general purpose planners to create solutions for complex problems is sometimes seen with conflicting opinions: The creation of a plan is resource intense and can produce use case specific plans, which are rarely reusable. On the other hand, they can find solution humans are not capable of finding. This is due the the fact that a human is insufficiently effective when scanning through the vast amount of available services and comprehend their contextual usage.

This leads to the question when to use a general purpose planner during the creation of service compositions. The answer is flexible as it is unspecific: The planner can be used to the extend the developer needs its help. If the developer wants to prove feasibility before creating a hand-made service composition, then the whole service composition can be planned to get an idea of the available services. If the overall service composition is already designed, the inclusion of new services might entail other services, which can be found through planning. This can help a developer to chose between the fit of a service into its compositions. At the end the suggestion of single services in a context of a service composition (between a service layer $i-1$ and another $i+1$) can be based on QoS parameters as well.

This allows a specification of the adaptive parts of a service composition, since the parts that are provided by a planner can be changed during runtime. By removing the grounding of a service, the so-created service template needs to be filled with a service instance available at runtime, which enforces adaptation. In this way, a compromise between development time and adaptiveness can be found.

1) Approach: The ability to automatically compose services to reach a given goal is called service planning [3]. The service planner based on the $SeMa^2$ utilises the service matcher for three tasks: first, to reason about effects and preconditions to find applicable service. Second, to reason on parameter selection for grounding the services, and third, to apply the execution of a service to reach a new state.

The algorithm in Figure 3 describes a standard planning approach applied to service planning. Here, the contribution is a planning in the service world without translating the service to the Planning Domain Description Language (PDDL) or similar to solve the planning problem.

The search used is defined in the function *StateSearch.next(Open)*. Depending on the implementation of the state search, the next state to be extended is selected. Here an A^* or equivalent algorithm can be used. In each state s that will be extended next, the selection of the services and their grounding is formalised in the function *ServiceSearch.UsefulServices(s)*. Here a set of grounded services is selected, which define the transition to the following open states. The state transition function is given by *execute(s, g)*, where the output and the effect of a service are integrated into the given state s. This is a theoretical execution, since the execution at runtime includes backtracking and a context sensing mechanism to sense the effect of a service. After extending a multitude of nodes during the search of the state space, the function *reconstructPath(path)* reduces the path from the goal to the start state to a minimal call of services.

Name: ServicePlan
Input: S_{start}, S_{goal}, Services **Output:** Service Composition

```
1:  path ← [ ]
2:  Closed ← ∅
3:  Open ← {S_start}
4:  while s ← StateSearch.next(Open) do
5:      if s ∉ Closed then
6:          if s = S_goal then
7:              return reconstructPath(path + [s])
8:          end if
9:          grounded ← ServiceSearch.UsefulServices(s)
10:         if grounded ≠ ∅ then
11:             succ ← {execute(s, g) | g ∈ grounded}
12:             Open ← Open ∪ succ \ Closed
13:             path ← path + [s]
14:         end if
15:         Closed ← Closed ∪ {s}
16:     end if
17: end while
18: return failure
```

Figure 3. Service Planner algorithm.

The complexity of the algorithm depends on the implementation of the state search and state pruning mechanism, being the heuristic that selects useful services, including the complexity of the service matcher used. In general, the worst case complexity of such an algorithm is exponential [15, p.72].

By planning on services we accept a number of challenges:

- *Service Grounding* checks all parameters of services to be executed next and creates all combinations of individuals that fit those parameters. These combinations lead to multiple (possibly infinite) grounded services out of one service description. Here the challenge lies in the selection of continuous parameters.
- *Output Integration* into the state poses a challenge since it is not clear how a service without effect can influence the state. One example of such services are information providing services, which are not world altering services [16]. Thus, here we create an assertion of the class of the output, creating an appropriate individual, equivalent to the "AgentKnows" of Doherty et al. [17].
- *Semantic Web Rule Language built-ins (SWRLb)* are mathematical extensions like "greater than", string manipulations or description of time. Additionally, lists are modelled in SWRLb but are not supported by reasoners like Pellet [18].
- *Semantic Web Rule Language XML Concrete Syntax (SWRLx)* is an extension to the Semantic Web Rule Language (SWRL) allowing to model individual creation, creation of classes and properties. This is vital to the service planning, because service execution might create individuals or classes that can not be modelled without SWRLx built-ins.

2) State-of-the-Art: There is a multitude of related work in using planning techniques for service composition. Rodriguez et al. [19] analyse three parts of Service-Oriented Architecture (SOA) in connection to artificial intelligence planning: i) service discovery techniques, ii) service composition systems,

and iii) service development tools. Even though Rodriguez et al. noticed that the use of QoS parameters for the selection of a service in a service composition is "a significant research problem" [19] their analysis of the state of the art does not reflect QoS parameters. Markou and Refanidis focus on non-deterministic planning approaches for automated service composition where the approaches are analysed for their heuristics but only in the sense of using them or not [20]. Zou et al. [21], [22], [23] focus more on efficiency of the planning techniques for the automated service composition but neglect to see the importance of semantic information given by semantic web service descriptions. This leads them to look at approaches translating the service composition problem into a PDDL planning domain and with that they lose every semantic description available before. Transforming facts described in OWL (Web Ontology Language) and, e.g., SWRL into a PDDL domain leaves with no basis for ontology matching if the same fact is formulated in different ways.

We argue that the performance of a general purpose planner depends on the heuristic used during the search for a path from start to goal state. This leads us to the conclusion to analyse the state of the art for factors neglected by other surveys, e.g., how a service description is used to create heuristics for the used planning technique.

We start out with Rodriguez-Mier et al. [24] who neglect non-functional parameters as well, but they describe a general heuristic that is used in an A^* algorithm. This heuristic combines the amount of already executed services in a backward search: $h(n) = distance(S_n, S_{goal})$, where S_n is the current state and S_{goal} is the goal state. In addition the cost of a state S_n is the number of services still needed to reach the start state. This is denoted with $c(n)$. This is combined to the heuristic function $f(n) = h(n) + c(n)$. This heuristic has the drawback that it only is applicable after a solution has been found. Thus it can be used to select the best path, from start to goal state, but it can not be used during the planning itself.

Meyer and Weske [25] as well count the number of service executions as a heuristic for their planning mechanism. They restrict this heuristic further, arguing that it is only an upper limit since the plan could include parallel executions and thus the amount of service execution steps can be further reduced.

Hoffman and Nebel [26] use a relaxed plan heuristic by removing the deletions out of the effect of a service. This can be done in PDDL since there are only additions and deletions of facts. Such a heuristic becomes research worthy if the semantics of the problem looked at comply to the open world assumption. This is because we can not decide if a left-out fact is true or not. Further effects could conflict each other because they are coming from different conceptualisations, e.g., different ontologies describing the same or opposite fact.

Klusch et al. [27] use the heuristic of Hoffman and Nebel [26] where a breadth first search is used if services have the same heuristic value.

Bonet and Geffner [28] build a heuristic by evaluating each fact of the goal. Here a fact f_g of the goal has an estimated cost of the length of a minimal path through the planning state space from the initial state. This leads to a heuristic:

$$h(s) = \sum_{f_g \in S_{goal}} g_s(f_g) \quad (1)$$

where

$$g_s(f_g) = \begin{cases} 0 & \text{if } f_g \in S \\ \max\limits_{s \in Services} [1 + g_s(s.pre)] & \text{else} \end{cases} \quad (2)$$

Here $Services$ is the set of all available services and $s.pre$ is the set of preconditions of the service s. The maximum in Equation (2) is chosen because it forms an admissible heuristic [28].

Mediratta and Srivastrava [29] introduce the heuristic of Hoffman and Nebel [26] to an AND-OR graph as a plan. Here the cost of each OR path through the graph is selected with a minimum. For AND-connected nodes in the Graph the cost function is summed up.

Fanjiang and Syu [30] use genetic algorithms for the service composition, which does not contain any heuristics at all. This is the same with Lécué et al. [31] who use reasoning in OWL-DL (Web Ontology Language Description Logic) for the selection of fitting services but no heuristics.

Regarding this state of the art we suggest that heuristics are part of the future work in this domain [32]. This is based on the performance issues described in the evaluation of the here analysed papers. Furthermore, the heuristic can be used during design time, to suggest services to the developer as part of a semi-automatic service composition framework. During design time, the actual execution and parametrisation of the service is left to the developer. Thus, a heuristic does not need to select purely executable services for a given state.

IV. Development Components

The method for semantic service management and development makes use of two development tools, which are both implemented as Eclipse plugins [33] and thus can seamlessly be integrated into the developer's usual workflow.

The overall architecture of the proposed development components is shown in Figure 4. We differentiate between the **runtime**, in which a service can be grounded to its real parameters and execution environment, and the **design time**, in which there are fewer resource constrains for the planning and an expert is able to fine-tune the service composition in a BPMN editor.

Here, black arrows describe the information flow, e.g., the service matcher gets the service descriptions out of a service repository and supplies the state-space-planner with fitting services for a current state.

a) Design Time: At design time, the developer is supported by a **BPMN editor** to create service compositions. This BPMN editor uses **semantic annotations** to describe the functional and non-functional aspects of a service. These descriptions are used by a planning component to build service compositions (a sequence of services) to suggest to the user while he is searching for reusable service for his composition. This **state space planner** uses a **service matcher** to identify useful services out of a service repository.

The result of this process is a BPMN description of a service composition that can be deployed into a runtime. This description can be quite abstract as we will describe in Section IV-B. This abstraction allows for flexibility and thus for an adaptation of the composition at runtime.

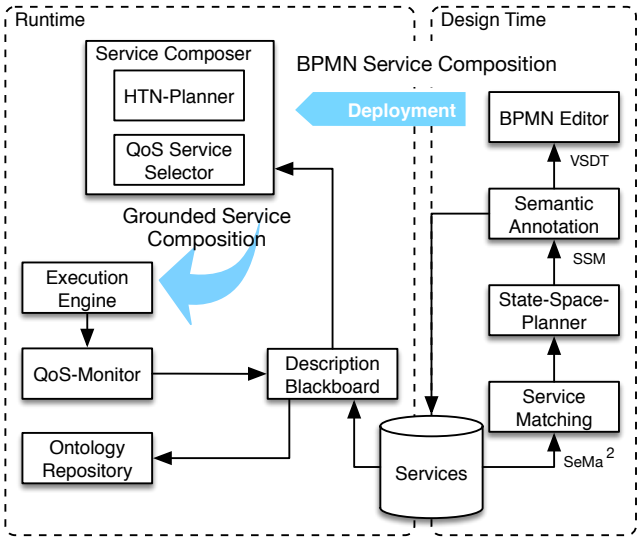

Figure 4. Architecture of the development components.

b) Runtime: At runtime, the given service composition is concretised. This is done since the resource consumption of the general purpose planning is too high to be used at runtime. To keep the introduced flexibility, the **service composer** uses a **HTN-planner** (Hierarchical Task Network) to select between alternative sub-plans. This is thought as a first principle planner where plans are selected from a plan library [34]. As a one-step plan, a service call is the atomic entity that can be replaced. This service composer can replace unavailable services, or use a **QoS service selector** to optimise the service composition to some criteria. With ever more service compositions available and thus more alternatives of sub-plans to search from, this service composer becomes a fast planner with domain specific, optimised service composition.

Another task of the service composer is the selection of unknown parameters. Those are called the service grounding. This might be, e.g., the resolution of a display device or the rendered models, which are unknown during design time. Furthermore, if a template of a service is part of the composition, the concrete service instance needs to be chosen, before the composition can be executed. Again this selection can be based on the QoS parameters of the services.

The resulting grounded service composition is passed to an **execution engine**. This execution engine reports QoS parameters back to the QoS-Monitoring, which in parts then ranks the services used to learn their quality parameters for future references.

The **service blackboard** describes the available services and their QoS behaviour. The service blackboard thus does restrict the accessibility of services theoretically available during design time and practically executable during runtime – there might be political, organisational, or financial reasons as to by whom services can be accessed – and lets the service composer choose from alternatives.

All in all, this separation of runtime and design-time is a trade-off between complexity and adaptability. Since the automated creation of a service composition from scratch is too expensive, the adaptation of existing plans might lose on adaptiveness, but renders the system profitable through reusability of plans.

In the following we describe the two involved development components, namely the Semantic Service and Ontology Manager (SSM) and the Visual Service Design Tool (VSDT) in more detail.

A. *Semantic Service and Ontology Manager*

The description of the Semantic Service and Ontology Manager is divided into an approach section and a short state-of-the art section related to the semantic annotation of services.

1) Approach: In order to be able to integrate intelligent planning algorithms, the environment has to come up with the necessary infrastructure. One essential requirement in this respect is the semantic description of functionalities or services. Since current standards such as OWL-S [35] are not easy to describe from scratch, we developed a plug-in called Semantic Service Manager (SSM) [36], providing a set of features supporting a semi-automatic description of services. The core of SSM is an Ontology Manager (see Figure 5), which enables the developer to include and utilize OWL ontologies for the application in semantic service descriptions. However, since many development approaches use other languages to specify the domain of concern, such as the Eclipse Modeling Framework (EMF), the Ontology Manager also provides a transformation process from EMF to OWL.

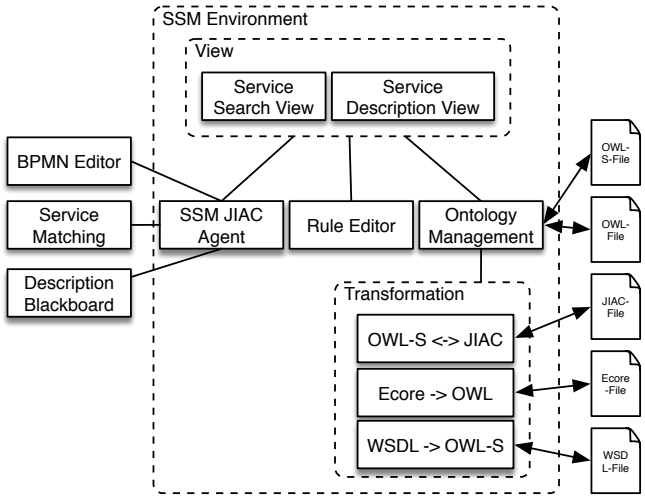

Figure 5. Components of the Semantic Service Manager.

Based on the Ontology Manager the developer is then able to describe the service according to name, description, input and output parameters and finally preconditions and effects. The latter ones can be described via SWRL, and for this purpose SSM comes with a syntax highlighting editor and structure parser. The description can then be utilized in different ways. Either it can be deployed to a semantic service repository (see Section V), it can be sent to a BPMN process (see the next paragraph), or it can be linked to a service of the multi-agent framework JIAC V (Java-based Intelligent Agent Componentware, version 5) [37]. With these options at hand, the developer can easily connect semantic descriptions to services and is able to deploy them immediately.

The second purpose of the SSM is the search and utilization of existing and running services within a distributed environment. Therefore, the SSM provides a *Service Discovery View* where the developer can define (incomplete) parameters of a service and search the platform directory using the $SeMa^2$ matcher. The developer can also adapt the weightings of the different matching techniques used. After selecting one of the services they can either be pushed to the *Visual Service Design Tool* (VSDT) to use it within a BPMN process (see Figure 7), or a code inclusion function can be triggered that inserts the service call code into the open Java window.

The different ways how the SSM can be used for describing or searching services or service templates and for importing them into a complex process are shown in Figure 6.

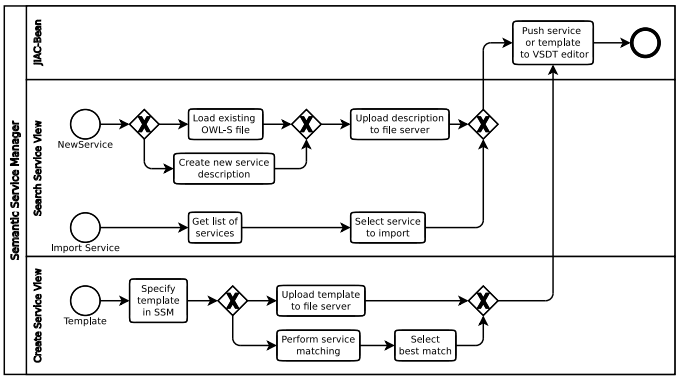

Figure 6. Different ways of creating or searching services with the SSM and using them in complex processes.

2) State-of-the-Art: Many of the works in the context of semantic service management merely focus on service matchmaking, but forget the design process, although being just as important. However, some focus on the semantic markup. The *OWL-S editor* [38] is a plug-in for Protégé, an open source ontology editor. With it a complete creation and editing of OWL-S descriptions is possible. Furthermore, the service behind the description can be tested via a graphical user interface. Drawbacks of the OWL-S editor are that it cannot handle multiple ontologies, because of limitations of Protégé. There is no connection to a framework, meaning that the editor lacks usability. The authors also do not see the benefits of a Java-to-OWL transformation. They argue that most commonly the service is developed before the implementation in code. Another editor for OWL-S is the *OWL-S IDE* [39]. It is a plug-in for Eclipse and, contrary to the OWL-S editor, supports the generation of OWL-S skeletons out of Java code. However, this generation is limited to basic types due to the missing support of ontologies. Furthermore, it does not support preconditions nor effects.

B. Visual Service Design Tool

This section starts with a detailed description of the Visual Service Design Tool followed by a state-of-the-art paragraph related to process modelling.

1) Approach: While basic services are usually implemented in the form of Java classes or equivalent, for service compositions the Business Process Model and Notation (BPMN) [40] has proven useful. Using the VSDT, existing semantic services can be imported from the SSM and orchestrated to complex processes using the BPMN notation (Figure 7).

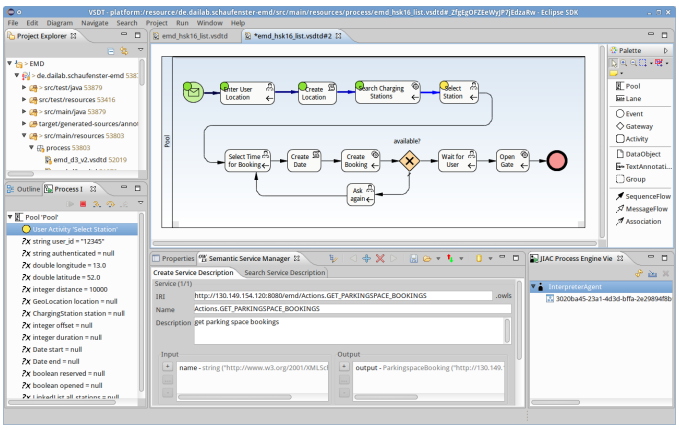

Figure 7. Semantic service development tools with example usecase. Top: VSDT editor showing process diagram; bottom: SSM view.

The VSDT is based on the Eclipse Graphical Modelling Framework (GMF) and provides a rich visual editor for BPMN processes [41]. It also provides means for process validation, simulation/debugging, and export features to different executable languages.

The BPMN editor integrates with the Semantic Service Manager view in the way that services from the SSM can be imported into the VSDT. Via a function in the User Interface (UI), an accordant service description is added to the currently opened VSDT process, together with data types representing the different ontology concepts. That service can then be used in a *service* task and combined with control flow, short scripts, and other services to a complex process. Instead of an actual service, the same approach can also be used for importing a service template into the VSDT process, which will then be matched to an actual service at runtime.

Accordingly, the BPMN service model used in the VSDT had to be extended to allow for semantic information. While the BPMN specification only accounts for Web service implementations – both for *service-* and for *send-* and *receive*-tasks – we extended the model to allow for the implementation to be either a Service or a Message Channel, according to the more diverse means of interaction in JIAC, and in multi-agent systems in general. Also, while the service description can still be used for Web services, it supports additional attributes for the service's preconditions and effects, e.g., in the form of SWRL expressions, and whether the service refers to an actual service or a service template (Figure 8).

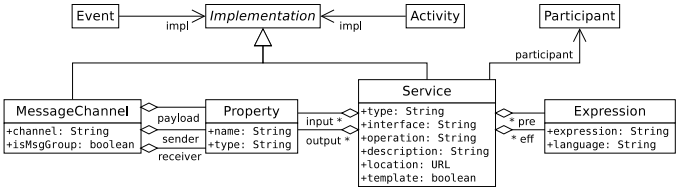

Figure 8. Extended Message- and Service-model used in VSDT BPMN editor.

Next, those processes can be exported to executable languages such as BPEL (Business Process Execution Language)

processes [41] or JIAC agent behaviours [42], [43], the latter being the execution environment used in this approach, which will be discussed in detail in Section V. In the case of JIAC agents, VSDT processes can either be compiled to JIAC beans, encapsulating an accordant behaviour, or they can be interpreted directly. In this work, we will focus on the interpreting approach.

2) State-of-the-Art: BPMN [40] can be used for describing services and service orchestrations in particular on a high level of abstraction. BPMN provides a rich syntax for modelling both the internal processes as well as the interactions of the system, and can thus be seen as a combination of Activity Diagrams and Sequence Diagrams of the Unified Modelling Language (UML). Further, while the process diagrams are easily understandable, the underlying formal model provides the attributes necessary to describe readily-executable programs.

BPMN is being used for modelling and generating service-oriented systems in a number of other works and can be seen as a de-facto standard for this task. Besides the mapping from BPMN to BPEL that is included in the specification itself [40, Chapter 14], alternative mappings have been proposed, e.g., by Ouyang et al. [44] and Mendling et al. [45]. Today, many process management systems can also execute the BPMN diagrams directly.

Besides those well-established paths, there are also approaches using BPMN for modelling agents and multi-agent systems. For instance, in GPMN, Jander et al. [46] combine BPMN processes with goal-hierarchies for Jadex agents equipped with BPMN interpreters. In WADE [47], on the other hand, a proprietary notation similar to BPMN is used, and the processes are transformed to executable code for JADE (Java Agent Development Framework).

Concerning the integration of semantics into BPMN, Barnickel et al. extended the Oryx BPMN editor with ontology matching capabilities, using OWL-DL [48], but to the best of the authors' knowledge there are no approaches towards integrating semantic service matching into BPMN or accordant process engines.

V. Runtime Components

In this section we will discuss the different components of the runtime environment. The services are executed as part of a JIAC multi-agent system. This way, each service is running on an individual agent, providing an adequate level of modularity and encapsulation. The environment also provides interfaces to other types of (web) services, such as WSDL (Web Service Description Language), SOAP (Simple Object Access Protocol), and REST (Representational State Transfer), which can be integrated transparently with JIAC.

A. Multi-agent Framework

The execution environment is based on *JIAC V* (www.jiac.de), a multi-agent framework also incorporating many aspects of service-oriented architectures [37]. The agents are situated on agent nodes (runtime containers).

Complementary to message-based communication, one of the core mechanics of JIAC agents is to expose *actions*. Depending on its scope, an action can be found and used by other components of the same agent, by other agents on the same node, by any agent on the network, or exposed as a webservice to be used by different applications. Each JIAC agent node has a directory of known agents and actions, both on the same node as well as on other nodes, that can be used for querying and finding specific agents and actions according to templates. Given just the name, or the inputs and outputs of an action, the directory will find and return an action that matches that template (if such an action exists), which can then be used for creating an accordant intention.

Each agent's behaviours and capabilities are defined in several agent beans, providing different general and application-specific functions (see Figure 9).

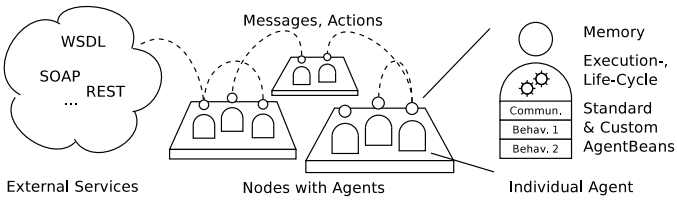

Figure 9. Components of a JIAC multi-agent system and individual agents (adapted from [49]).

Besides providing actions for others to use, agent beans can also implement periodic behaviour, or behaviours to be executed when the state of the agent changes (e.g., when it is starting or stopping). Also, they can attach observers to the agent's memory to react, e.g., to incoming messages or to changes in the environment. Finally, several application-independent beans can be added to the agent or the agent node as a whole, to provide certain functionalities, such as communication, persistence, migration, or reactive behaviour.

Integrating the semantic service matcher into JIAC was very natural and straightforward. The $SeMa^2$ itself has been wrapped into a JIAC agent node bean, i.e., there is one instance of the matcher for each individual node, shared by all agents on that node, hooking into the *directory* running on that node. Whenever a semantic service template (as opposed to a plain JIAC action template) is passed to the directory for service lookup, the directory will delegate it to the semantic service matcher bean, which will return the best matching service. To the agent invoking the service, it is fully transparent whether the found capability is a standard JIAC action or a semantic service.

In order to utilise the service matching and planning functionalities within the JIAC environment it was necessary to extend the existing action model for agents by means of a semantic service description model. The model is oriented towards the OWL-S standard dividing information into Profile, Process and Grounding parts. The latter can either reference JIAC action information or it can define WSDL or REST attributes.

B. WSDL and REST Web Service Integration

For interfacing with other services, the WSDL- and REST-services integration beans can be used. Those two components do both have the following two effects:

- all the JIAC actions accessible via the directory that have the 'webservice' scope will be exposed to the outside world as accordant WSDL or REST services, respectively,

- additional JIAC actions will be created and exposed, representing each of the WSDL and REST services known to those beans.

The respective input and output data types (e.g., XML schemas in the case of WSDL web services) are mapped to corresponding Java classes, and vice versa. The created web services are hosted by the same agent node using an integrated Jetty server. Thus, JIAC agents can seamlessly and transparently be integrated with both, REST and WSDL services.

C. Semantic Service Repository

Each of the semantic services is associated with a URI resource, holding their actual semantic descriptions in the form of an OWL-S document. While in theory each of the agents (or corresponding entities in a different runtime) could host their respective service descriptions themselves, this approach is not optimal, as the URI might change depending on where the agent is running. Instead, a central *Semantic Service Repository* is used for hosting the different service descriptions and their relevant ontologies, each being identified by a unique and invariant URI.

The service repository has been realised as a JIAC agent node, encapsulating a Jetty web server and providing a number of actions for deploying, searching, and fetching service descriptions. It also supports multiple filters, e.g., for only showing services that are currently running. Service descriptions can be deployed to the repository either statically, using the SSM tool, or dynamically whenever an agent providing the respective service starts. The service repository automatically parses the service descriptions and adapts all the internal URI references, e.g., to the descriptions of the services' inputs and outputs within the same document, to its current server address, where those resources are stored.

Each JIAC service, that is backed by a semantic service description, has an attribute `semanticURI` referring to the corresponding OWL-S resources. When the $SeMa^2$ matcher is invoked from within JIAC, the runtime will fetch the service descriptions, pass those to the matcher, receive the result, and finally return the action whose `semanticURI` corresponds to that matching result.

Currently, we are working towards distributing the service repository, to improve scalability for large numbers of services and service requests, as well as the ability to automatically assess the Quality-of-Service (QoS) of the invoked services, e.g., latency and time-to-complete.

D. JIAC based BPMN Interpreter

One of several application-independent components for JIAC agents is the process interpreter bean, enabling the agent to interpret and execute BPMN processes created with the VSDT [42].

The process interpreter bean is composed of three layers (Figure 10): First, the *process interpreter bean* itself provides actions for adding processes to be interpreted and for managing already running processes. Also, it acts as an interface to the agents, providing functionality for sending and receiving messages and invoking other actions from within the BPMN processes. Finally, it exposes all the processes (that have an accordant start event) as actions so they can be used by other agents.

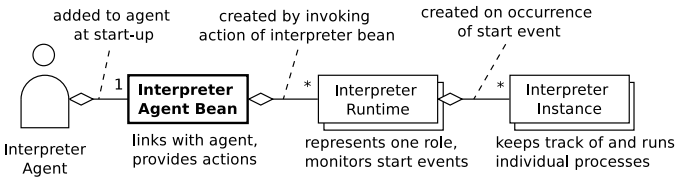

Figure 10. Layered architecture of BPMN Interpreter Bean. [42]

Whenever a BPMN diagram is added to the process engine bean for interpretation, an *interpreter runtime* is created, which is responsible for each process spawned from this diagram. It keeps track of start events and creates a new instance of that process whenever an event corresponding to the respective process start events occurs. Several volatile *process instances* are responsible for running the individual processes spawned by the runtime, evaluating conditions and assignments, executing the different activities, and keeping track of the current state of the process, i.e., which activities are ready for execution, as well as the values of the different process variables. Depending on the type of the activities, different actions are taken, e.g., sending or receiving a message, invoking another service, executing some short script, or interacting with the user.

After one or more processes have been deployed to the interpreter – either at start-up or using the above-mentioned actions – in each step of the *interpreter bean*'s execution cycle, each *interpreter runtime* will advance each of its associated *interpreter runtimes* by one step, which in turn each execute each activity that is currently in a *ready* or *active* state.

Employing JIAC's communication and service infrastructure, the interpreted processes can discover and make use of other JIAC actions, and – if the respective proxy beans are present – of WSDL and REST services. If the semantic service matcher is installed in the node, it is automatically used for finding services according to the templates used in the processes. The current state of the interpreter bean – the active runtimes, their respective process instances, and their internal states – can be monitored using a simple UI, also providing an interface for manually starting processes and for the processes to interact with the user, e.g., for BPMN *user tasks*, or for querying missing service parameters.

While this UI is intended for developers, a similar generic UI can also be used for invoking the services and interacting with the user in a more end-user friendly way, as we will show in the following.

E. Smart Personal Assistant and UI Renderer

The *Smart Personal Assistant* (SPA) is a UI framework for quickly developing adaptive, multimodal user interfaces for services [50], and is used in a number of research projects. While the usual SPA UI is manually created for the service at hand and styled to fit the design of the respective project, a special *Renderer UI* has been created, allowing to start any service, and also providing callbacks for user interaction triggered by the invoked service. Similar to the interpreter monitoring UI, input fields for querying the service parameters and for displaying the output are automatically derived from the respective classes, using the Java Reflection API.

While those generic, automatically generated UIs do not look as polished as the manually crafted ones, they allow

for quickly prototyping new complex services with rich user interaction and for integrating them into the user's workflow.

All of the here described components are open source and available for download from www.jiac.de.

VI. Methodology

In the following, we will sketch a process of how the different components introduced in the last three sections are used together to form a methodology of semantic service engineering. At its base, the method is similar to other software- and service engineering methodologies, but combines those with requirements for and contributions of semantic services. An overview of the methodology is shown in Figure 11, using the BPMN notation, and highlighting how the different components are used in the stages of the process. In the following, we will describe the different steps in more detail.

A. Ontology Engineering

The first step in creating semantic services is to model the ontology that will be used for describing the service's inputs, outputs, precondition and effect, if any. This is particularly important, since one of the main motivations for semantic services is for those services to be easily findable, reusable, and composable with other services; thus, whenever possible it should be the aim to reuse, or, if necessary, extend existing ontologies, instead of creating new ones. This step is also concerned with mapping the ontological concepts, for example described in OWL, to a representation that is closer to the service implementation, e.g., Java classes (or vice versa, starting with Java classes and generating according OWL ontologies).

The new or modified ontologies are then uploaded to a server hosting a repository of known ontologies, so they can be used in the next step, as well as in other services. There is no specific tool for this step in our method. Ontologies can be created, e.g., with Protégé [51], or generated from existing Java classes or EMF models [52].

B. Creating Semantic Service Description

Next is the creation of the semantic service description itself, defining the "contract" of the service. Of course, this step is not particular for semantic services, but is a common practice for all of service- and software engineering. The major difference is that besides name, textual description, input and output parameter, also the preconditions and effects of a service can be defined. Especially the latter, which in our approach can be described with the semantic rule language SWRL, extend the attributes of a service in a way that matching or planning processes can deduce its purpose and its formal prerequisites. However, as describing semantic terms can be challenging, we paid attention to provide a user-friendly editor with syntax-highlighting, auto-completion and validation parser. Currently missing, but contemplated is the integration of several QoS attributes, making the selection of services also sensitive to non-functional aspects.

The new service description is uploaded to a service repository, adding it to the list of services usable by the semantic service matcher and planner. In our method, the SSM tool is used for creating the service descriptions using OWL-S. Existing ontologies can be browsed (but not edited) for selecting concepts for input and output, while preconditions and effects are specified using SWRL. The finalized service description can then be deployed to the repository and an accordant stub for the service implementation can be generated.

C. Service- and Process Engineering

The bulk of the service development process is occupied with engineering the service's implementation. While the service's method declaration can be generated from the semantic service description, its body has to be implemented by a developer. Here, we can differentiate two main activities: Identifying and integrating existing services, and developing the logic that combines those services to a new service, or process, with added value.

There are three ways how services can be searched, identified, and imported into the currently developed process, using the SSM tool:

- The service can be searched for, using a semantic service template, and the service best matching the template is integrated into the current service.
- In case no single service satisfies the template, the semantic service planner can be used to automatically find a service composition that, as a whole, matches the template; the individual services of that composition are then integrated into the current service in the appropriate sequence.
- Instead of searching services at design time, the template that would be used for matching the service can itself be integrated into the current service, deferring the search and matching process to runtime.

Of course, there is also a fourth case: That no service or service composition can be found that fulfils the template. In this case, a new service has to be created, thus starting a new instance of the service development process.

The service logic can be created in two ways: Either in the form of a Java method, or, using the VSDT, as a BPMN process, which is later either mapped to Java (JIAC agent beans) or interpreted directly. Which one to choose mainly depends on the ratio of service reuse to "original" service logic: In case the new service is mainly a composition of existing basic services, they can very well be modelled visually as business processes, but if they contain complex calculations or make extensive use of third-party libraries (that are not available as services), then implementing the services in plain Java is the better choice. As a middle way, it is also possible to integrate short snippets of Java code into a BPMN process, using *script* tasks.

D. Testing the Implementation against the Specification

The last step before deployment is testing, to ensure that the services' implementations comply with their semantic descriptions. Of course, testing plays a well-established role in software engineering and is not particular to semantic service development. However, the presence of formal semantic descriptions impose both an obligation and an opportunity for (automated) unit testing.

On the one hand, while even a regular function or service that does not comply with its documentation is always a nuisance, a semantic service that violates its stated effect could threaten the functionality of the entire system it is embedded in, as automated planners will rely on that information. On

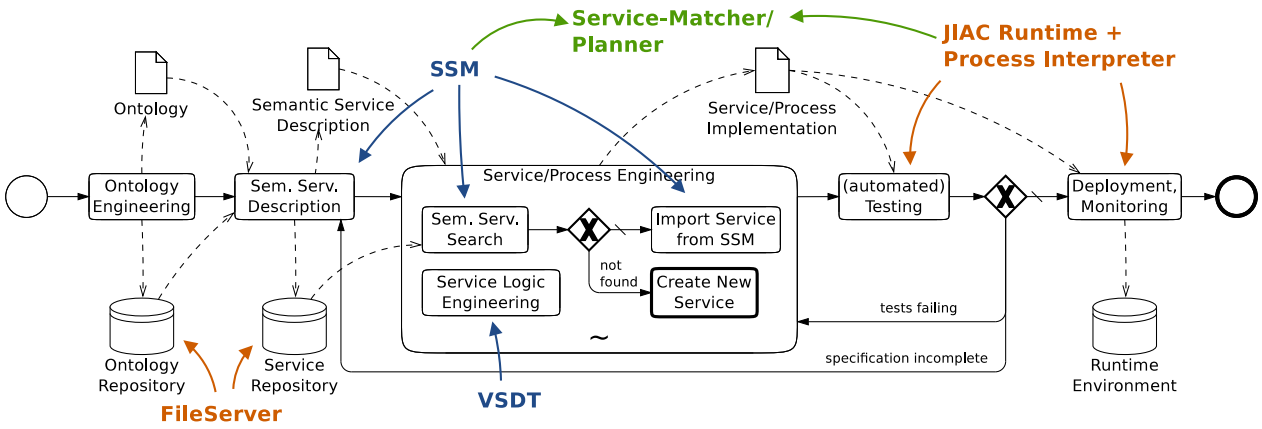

Figure 11. Semantic Service Management and Development Process, as a BPMN process, and associated components: Green: Semantic Service Core; Blue: Development Tools; Red: Execution Environment. The central activity is an *ad-hoc* subprocess, executing the embedded tasks as needed, in no fixed order.

the other hand, since the intended behaviour of the service has already been specified in its precondition and effect, writing the actual tests becomes very straightforward.

While this is currently not implemented in our approach, it would also be possible to automatically generate unit tests from the semantic service description, particularly the service's pre-conditions and according effects. For this, the input parameters can be generated, setting all attributes that are not specified in the precondition randomly; then, the expected output can be inferred from the service's effect, thus testing the actual result of the service invocation against the expected value.

In case the service does not comply with the tests (i.e., with its stated preconditions and effects), the usual course of action is, of course, to fix the service. However, in some cases this may also expose flaws in the service's input, output, precondition and effect (IOPE) descriptions. In this case, the process has to backtrack and update the semantic service description and adapt or extend the service's implementation accordingly.

E. Deployment and Runtime Monitoring

The final step is to deploy the new service to the runtime environment. Depending on whether the service has been implemented directly as a Java class (e.g., a JIAC agent bean exposing an accordant action), or in the form of a BPMN process diagram orchestrating different existing services, the deployment process is slightly different.

- In case the service has been implemented directly in Java and is meant to be a basic service to be used as a building block for other services, it is best to create a new agent exclusively for that service and to deploy it to the runtime server.
- In case of a service composition created as a BPMN process, the process diagram can be deployed to an already running process interpreter agent. This way, deployment and undeployment is very dynamic, and the interpreter also provides basic capabilities for runtime monitoring and user interaction. Alternatively, the process can also be automatically translated to Java code and deployed as in the above mentioned case.

In both cases, the services are deployed to the JIAC runtime environment and can be invoked as actions, and searched for using the semantic service matcher. Using the WSDL and REST integration beans, the services will also be exposed as WSDL or REST services, respectively, and can transparently use other services available in those formats.

VII. Semantic Service Management in Practice

In this section, we will explore how the semantic service engineering method discussed in this paper can be applied in practice. For this, we will have a look at two scenarios: First, we describe how the service matcher and the development tools have been used in a recently completed research project in the e-mobility domain. Then, we continue to describe one of our current projects, in which we are extending our approach for the *augmented reality* domain.

A. Semantic Services for E-Mobility

In the project EMD (Extendable and adaptive E-Mobility Services), a use case within the transportation domain was constructed to demonstrate the use of the developed tools, the methodology, and the basic services, as seen in Figure 7. The process was created using the VSDT editor, orchestrating services from the SSM. The finished process is deployed to the JIAC BPMN interpreter for execution and the SeMa^2 is used for service matching at runtime.

Our first scenario combines different basic services for searching for charging stations, reserving parking spaces and charging slots, as well as access control to the same. First, the process queries the user's information, particularly w.r.t. her current location, and available subscriptions for car sharing and parking space providers. It then uses the location to find charging stations that are close by using services from charging station provides for whom the user has a subscription. Those charging stations are then presented in a list for the user to choose from, using the UI renderer, and the user is asked for the time of reservation. The corresponding charging station reservation service is matched and the booking is made, if that time slot is not already taken, or the user is asked again. Finally, as soon as the user signals that she arrived at the location, another service is matched and invoked to handle the access control, if any.

In this example, the SeMa^2 can be used for finding relevant services for searching and reserving charging stations, depending on the user's subscriptions. For this, the `User` object is

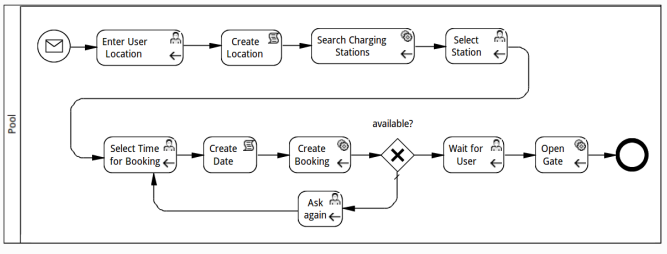

Figure 12. Service orchestration in the e-mobility use-case.

passed as an input parameter or as a context object to the service matcher, and the different services, having a precondition like `hasSubscription(?user, "Provider X")`, can then be selected by the semantic service matcher. This matching can be either be done a-priori, at design time, using the SSM for finding a set of matching services for one specific provider, or it can be deferred to runtime, making the selected services depending on the current user.

Of course, this required the relevant services not only to be semantically annotated, but to be so using the same domain model, or ontology, as used in the rest of the system. Since this is usually not the case, particularly when dealing with service provided by third parties (such as e-mobility providers) some of those services have to be wrapped accordingly.

B. Semantic Service Management for Augmented Reality

Another technical domain where semantic service management has the potential to lead to a boost of development is the area of augmented reality (AR) services. In the project AcRoSS (Augmented-Reality-based Product-Service Systems, more information available at: www.across-ar.de) we aim to set up a library for AR-services that can be used together with the software components presented in this paper in order to develop problem tailored solutions for small and medium-sized enterprises, for whom it is currently extremely hard and expensive to develop such specific solutions.

However, services for AR-glasses do have very strict requirements that have to be met. For instance, the hardware on the devices is currently still limited. Therefore, there has to be a very efficient concept for adaptive service processes. Furthermore, in many scenarios the glasses that will be used will be offline, meaning that the matching procedure and the services have to be located on the device itself. These issues as well as service specific aspects, like the quality of object recognition services, lead to the need for a service management concept that takes Quality-of-Service aspects into account.

One scenario within the AcRoSS project is about maintenance and repair of exposure machines, which are used for the creation of printing plates for the newspaper industry. The maintenance task includes the cleaning of exposure rolls, which means that they have to be taken out of the machine, maintained and correctly set into place again afterwards. Although this task sounds simple in first place, it is quite error-prone, since the rolls can also be set into place with the wrong direction or at the wrong place within the machine. The same challenge holds for even more complicated repair tasks. Currently, very experienced employees have to do these tasks at the client side, which is expensive. Using AR, the employee will be supported by services that recognize each part of the machine, search for related manuals in the backend, guide the employee what to do with the component and also check and display the machine's status. In order not to redevelop each process again and again for every machine, the process will be designed in an adaptive way, meaning that specific services like the retrieval of manuals will be selected dynamically as well as the request for the machine's status. Furthermore, at design time the developer will be able to select object recognition services via given Quality-of-Service parameters.

At the time this paper has been written the project was in the specification phase. A thorough evaluation will be done and published at a later point in time.

VIII. Conclusion

In this article, we presented an approach for semantically matching services and for combining those services to complex plans, both at design-time and at runtime, as well as a set of development tools and an accordant runtime environment for generating adaptive and flexible systems in service-oriented environments. Those planning components, development tools and runtime have been integrated into a methodology for semantic service management and engineering, covering all phases from semantic service description and service development up to testing and runtime monitoring. In this approach, semantic services are orchestrated in adaptive business processes, based on BPMN, where service templates can be specified within the process and dynamically matched to concrete services at runtime. This method has successfully been applied, among others, in a research project in the e-mobility sector. Currently, the same approach is adapted and extended for services in the augmented reality domain.

In the future, we plan to address a number of challenges related to automated service composition and planning [32]. Among others, we want to extend service matching and planning by taking quality-of-service aspects into account. Also, we want to investigate the use of heuristics for more efficient planning, to foster the usefulness of service planning in real-world applications.

Acknowledgements

This work is based on projects funded by the German Federal Ministry of Economic Affairs and Energy under the funding reference numbers 16SBB007A and 01MD16016F.

References

[1] J. Fähndrich, T. Küster, and N. Masuch, "Semantic service management for enabling adaptive and evolving processes," in Proc. of 11th Int. Conf. on Internet and Web Applications and Services (ICIW 2016), Valencia, Spain, May 22–26 2016, pp. 46–53.

[2] M. Klusch, U. Küster, A. Leger, D. Martin, and M. Paolucci, "5^{th} International Semantic Service Selection Contest - Performance Evaluation of Semantic Service Matchmakers," Nov. 2012, last access: 2016/11/28. [Online]. Available: http://www-ags.dfki.uni-sb.de/~klusch/s3/s3c-2012-summary-report.pdf

[3] J. Fähndrich, N. Masuch, H. Yildirim, and S. Albayrak, "Towards automated service matchmaking and planning for multi-agent systems with OWL-S – approach and challenges," in Service-Oriented Computing - ICSOC 2013 Workshops, ser. Lecture Notes in Computer Science, A. Lomuscio, S. Nepal, F. Patrizi, B. Benatallah, and I. Brandi, Eds. Springer International Publishing, 2014, vol. 8377, pp. 240–247.

[4] J. Fähndrich, S. Weber, and S. Ahrndt, "Design and Use of a Semantic Similarity Measure for Interoperability Among Agents," in Multiagent System Technologies. Springer International Publishing, 2016, pp. 41–57.

[5] P. A. Morris, "Combining expert judgments: A Bayesian approach," Management Science, vol. 23, no. 7, 1977, pp. 679–693.

[6] M. Stone, "The opinion pool," The Annals of Mathematical Statistics, vol. 32, no. 4, 1961, pp. 1339–1342.

[7] C. Genest, "Pooling operators with the marginalization property," The Canadian Journal of Statistics/La Revue Canadienne de Statistique, vol. 12, no. 2, 1984, pp. 153–163.

[8] D. Dyer. Watchmaker Framework. Last access: 2016/05/11. [Online]. Available: http://watchmaker.uncommons.org/ (2006)

[9] W. L. Goffe, G. D. Ferrier, and J. Rogers, "Global optimization of statistical functions with simulated annealing," Journal of Econometrics, vol. 60, no. 1-2, Jan. 1994, pp. 65–99.

[10] M. Klusch and P. Kapahnke. The Semantic Web Service Matchmaker Evaluation Environment (SME2). Last access: 2016/11/28. [Online]. Available: http://projects.semwebcentral.org/projects/sme2/ (2008)

[11] M. Klusch, P. Kapahnke, and I. Zinnikus, "SAWSDL-MX2: A machine-learning approach for integrating semantic web service matchmaking variants," in 2009 IEEE International Conference on Web Services (ICWS), IEEE Computer Society. IEEE, 2009, pp. 335–342.

[12] M. Klusch and P. Kapahnke, "The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection," vol. 15, Sep. 2012, pp. 1–14.

[13] M. Klusch, B. Fries, and K. Sycara, "OWLS-MX: A hybrid semantic web service matchmaker for OWL-S services," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 7, no. 2, Apr. 2009, pp. 121–133.

[14] F. E. Gmati, N. Y. Ayadi, A. Bahri, S. Chakhar, and A. Ishizaka, "A framework for parameterized semantic matchmaking and ranking of web services," in Proc. of 12th Int. Conf. on Web Information Systems and Technologies, 2016, pp. 54–65.

[15] M. Ghallab, D. S. Nau, and P. Traverso, Automated Planning: Theory & Practice, D. E. M. Penrose, Ed. Morgan Kaufmann, 2008.

[16] H. Saboohi and S. A. Kareem, "A resemblance study of test collections for world-altering semantic web services," in Int. MultiConf. of Engineers and Computer Scientists (IMECS), vol. I, 2011, pp. 716–720.

[17] P. Doherty, W. Lukaszewicz, and A. Szalas, "Efficient reasoning using the local closed-world assumption," in Agents and Computational Autonomy. Springer Berlin Heidelberg, Jan. 2003, pp. 49–58.

[18] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," Web Semant., vol. 5, no. 2, Jun. 2007, pp. 51–53.

[19] G. Rodríguez, Á. Soria, and M. Campo, "Artificial intelligence in service-oriented software design," Engineering Applications of Artificial Intelligence, vol. 53, no. C, Aug. 2016, pp. 86–104.

[20] G. Markou and I. Refanidis, "Non-deterministic planning methods for automated web service composition," Artif. Intell. Research, vol. 5, no. 1, 2016, p. 14.

[21] G. Zou, Y. Gan, Y. Chen, and B. Zhang, "Dynamic composition of Web services using efficient planners in large-scale service repository," Knowledge-Based Systems, vol. 62, May 2014, pp. 98–112.

[22] G. Zou et al., "QoS-aware dynamic composition of Web services using numerical temporal planning," 2012.

[23] G. Zou, Y. Chen, Y. Xu, R. Huang, and Y. Xiang, "Towards automated choreographing of web services using planning," AAAI, 2012.

[24] P. Rodriguez-Mier, M. Mucientes, and M. Lama, "Automatic web service composition with a heuristic-based search algorithm," in 2011 IEEE International Conference on Web Services (ICWS). IEEE, 2011, pp. 81–88.

[25] H. Meyer and M. Weske, "Automated Service Composition Using Heuristic Search," in Agents and Computational Autonomy. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 81–96.

[26] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," Journal of Artificial Intelligence Research, vol. 14, no. 1, 2001.

[27] M. Klusch and A. Gerber, "Fast Composition Planning of OWL-S Services and Application," in Proc. of European Conference on Web Services (ECOWS '06). IEEE, 2006, pp. 181–190.

[28] B. Bonet and H. Geffner, "Planning as heuristic search," Artificial Intelligence, vol. 129, no. 1-2, Jun. 2001, pp. 5–33.

[29] A. Mediratta and B. Srivastava, "Applying planning in composition of web services with a user-driven contingent planner," IBM Research, 2006.

[30] Y.-Y. FanJiang and Y. Syu, "Semantic-based automatic service composition with functional and non-functional requirements in design time: A genetic algorithm approach," Information and Software Technology, vol. 56, no. 3, Mar. 2014, pp. 352–373.

[31] F. Lécué, A. Léger, and A. Delteil, "DL Reasoning and AI Planning for Web Service Composition," in 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. IEEE, 2008, pp. 445–453.

[32] M. Lützenberger, T. Küster, N. Masuch, and J. Fähndrich, "Multi-agent systems in practice – when research meets reality," in Proc. of 15th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, and S. Marsella, Eds. Singapore: IFAAMAS, May 2016, pp. 796–805.

[33] Eclipse Foundation. Eclipse. Last access: 2016/11/28. [Online]. Available: http://www.eclipse.org/ (2016)

[34] L. de Silva, S. Sardiña, and L. Padgham, "First Principles Planning in BDI Systems," in Proccedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), C. Sierra, K. S. Decker, and J. S. Sichman, Eds. Budapest, Hungary: IFAAMAS, May 2009, pp. 1105–1112.

[35] D. Martin et al., "OWL-S: Semantic Markup for Web Services," Website, Tech. Rep., Nov. 2004. [Online]. Available: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

[36] N. Masuch, C. Kuster, and S. Albayrak, "Semantic service manager–enabling semantic web technologies in multi-agent systems," in Proceedings of the Joint Workshops on Semantic Web and Big Data Technologies, INFORMATIK 2014, Stuttgart, Germany, 2014, pp. 499–510.

[37] M. Lützenberger, T. Konnerth, and T. Küster, "Programming of multi-agent applications with JIAC," in Industrial Agents – Emerging Applications of Software Agents in Industry, P. Leitão and S. Karnouskos, Eds. Elsevier, 2015, pp. 381–400.

[38] D. Elenius et al., "The OWL-S editor – a development tool for semantic web services," in The Semantic Web: Research and Applications. Springer, 2005, pp. 78–92.

[39] N. Srinivasan, M. Paolucci, and K. Sycara, "Semantic web service discovery in the OWL-S IDE," in Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 06, ser. HICSS '06. Washington, DC, USA: IEEE Computer Society, 2006.

[40] OMG, "Business process model and notation (BPMN) version 2.0," Object Management Group, Specification formal/2011-01-03, 2011.

[41] T. Küster and A. Heßler, "Towards transformations from BPMN to heterogeneous systems," in Business Process Management Workshops, ser. LNBIP, D. Ardagna, M. Mecella, and J. Yang, Eds. Springer Berlin Heidelberg, 2009, vol. 17, pp. 200–211.

[42] T. Küster, A. Heßler, and S. Albayrak, "Process-oriented modelling, creation, and interpretation of multi-agent systems," International Journal of Agent-Oriented Software Engineering, 2016, to appear.

[43] T. Küster, M. Lützenberger, and S. Albayrak, "A formal description of a mapping from business processes to agents," in Engineering Multi-Agent Systems, ser. LNAI, M. Baldoni, L. Baresi, and M. Dastani, Eds. Springer International Publishing, 2015, vol. 9318, pp. 153–170.

[44] C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Mendling, "From business process models to process-oriented software systems," ACM Transactions on Software Engineering and Methodology, vol. 19, no. 1, August 2009, pp. 1–37.

[45] J. Mendling, K. B. Lassen, and U. Zdun, "On the transformation of control flow between block-oriented and graph-oriented process modelling languages," International Journal of Business Process Integration and Management (IJBPIM), vol. 3, no. 2, 2008, pp. 96–108.

[46] K. Jander, L. Braubach, A. Pokahr, W. Lamersdorf, and K. Wack, "Goal-oriented processes with GPMN," International Journal on Artificial Intelligence Tools, vol. 20, no. 6, 2011, pp. 1021–1041.

[47] F. Bergenti, G. Caire, and D. Gotta, "Interactive workflows with WADE," 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, vol. 0, 2012, pp. 10–15.

[48] N. Barnickel, J. Böttcher, and A. Paschke, "Semantic mediation of information flow in cross-organizational business process modeling," in Proc. of 5th Int. Workshop on Semantic Business Process Management SBPM 2010, held in conjunction with the European Semantic Web Conference (ESWC 2010), Heraklion, Greece, May 2010, pp. 21–28.

[49] T. Küster, A. Heßler, and S. Albayrak, "Towards process-oriented modelling and creation of multi-agent systems," in Engineering Multi-Agent Systems, ser. LNAI, F. Dalpiaz, J. Dix, and M. B. van Riemsdijk, Eds. Springer International Publishing, 2014, vol. 8758, pp. 163–180.

[50] N. Braun, R. Cissée, and S. Albayrak, "An agent-based approach to user-initiated semantic service interconnection," in Service-Oriented Computing: Agents, Semantics, and Engineering: AAMAS 2007 International Workshop, SOCASE 2007, Honolulu, HI, USA, May 14, 2007. Proceedings, J. Huang et al., Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 49–62.

[51] Stanford. Protégé. Last access: 2016/11/28. [Online]. Available: http://protege.stanford.edu/ (2016)

[52] Eclipse Foundation. Eclipse Modeling Framework (EMF). Last access: 2016/11/28. [Online]. Available: https://eclipse.org/modeling/emf/ (2016)

www.iariajournals.org

International Journal On Advances in Intelligent Systems
issn: 1942-2679

International Journal On Advances in Internet Technology
issn: 1942-2652

International Journal On Advances in Life Sciences
issn: 1942-2660

International Journal On Advances in Networks and Services
issn: 1942-2644

International Journal On Advances in Security
issn: 1942-2636

International Journal On Advances in Software
issn: 1942-2628

International Journal On Advances in Systems and Measurements
issn: 1942-261x

International Journal On Advances in Telecommunications
issn: 1942-2601