Report from Dagstuhl Seminar 13171

Customizing Service Platforms

Edited by

Luciano Baresi[1], Andreas Rummler[2], and Klaus Schmid[3]

1 **Politecnico di Milano, IT,** luciano.baresi@polimi.it
2 **SAP Research Center – Dresden, DE,** andreas.rummler@sap.com
3 **Universität Hildesheim, DE,** schmid@sse.uni-hildesheim.de

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 13171 "Customizing Service Platforms". The aim of the seminar was to bring together researchers from different areas of academia and industry that are related to the seminar topic and typically do not intensively interact with each other. These communities are Product Line Engineering, Software Architecture, Service Engineering, and Cloud Computing.

The ambition of the seminar was to work on the topic of "Customization of Service Platforms", which is related to all of these areas, in a synergistic and cooperative way to identify new research challenges and solution approaches. As part of the seminar, we identified a number of key areas which provided the basis for highly interactive working groups.

Seminar 21.–26. April, 2013 – www.dagstuhl.de/13171
1998 ACM Subject Classification D.2.2 Design Tools and Techniques, D.2.11 Software Architectures, D.2.13 Reusable Software
Keywords and phrases Service-Oriented Architectures, Service Platforms / Cloud Computing, Product Line Engineering, Variability Management
Digital Object Identifier 10.4230/DagRep.3.4.114

1 Executive Summary

Luciano Baresi
Andreas Rummler
Klaus Schmid

Background

Service-orientation has become a major trend in computer science over the last decade. More recently cloud computing is leading into the same direction: a virtualization of resources and service offerings. Especially cloud computing is getting very significant attention by companies. While the initial idea in service orientation was to have the relevant services standardized and distributed across the internet, we also see that an increasing amount of customization must be done to really meet customer needs. As in traditional system development, one size fits all is not enough.

This seminar focused on the notion of service platforms, a concept including, but not limited to, cloud computing. A service platform is a combination of technical infrastructure along with domain-specific or business-specific services built according to the service-oriented

development paradigm. Especially the latter in practice often requires significant customization in order to be practically useful. Providing such customizations on a massive scale cost-effectively is an extremely demanding task. This is a lesson that has been learned hard by a number of companies in traditional software engineering. As a consequence the concept of product line engineering was conceived.

The focus of this seminar was to explore the range of different approaches towards customized service offerings in current —- and future —- service-based environments. In particular, it was a goal to address the potential for a combination of service-orientation with product line engineering ideas. In this regard, this seminar was the first of its kind.

Diversity of Topics

The expected diversity of inputs that was desired for the seminar was well achieved. This is shown by the diversity of individual presentations summarized in chapter 3. Also the working groups that were established had participants from multiple communities. These working groups discussed the following topics:

Quality Assurance and Validation in the Context of Customization: Here, a broad range of different problems and techniques could be identified, related both to problems of varying of the object of the quality assurance as well as to the variation of the expections (qualities).

Mobility Devices and Customization: This working group focused particularly on the difficulties that arise from a mobile context with a lot of variation over time and limited resources.

Architecting for Platform Customization: Architectures are fundamental to any software system, so this group addressed what architectural techniques are important to create customizable platforms.

Energy-Aware Customization: Here, the focus was on the issue of energy-awareness and, in particular, energy-efficiency, which is particularly relevant to mobile platforms. By adequate customization, this can be improved for a platform.

Customizing Service Platforms for Cloud Computing: Modern cloud computing environments pose new challenges and provide new opportunities for customizing service platforms. It turned out that the cloud context provides a number of very special problems and technologies for addressing them.

Customizing Service Platforms for Agile Networked Organizations: The organizational context of service platform needs to be taken into account as well as a platform needs to fit to the relevant business context. Hence customization needs to be done on both levels in a synchronized manner.

Binding time aspects of service platform customization: This working group focused on when (i.e., in which lifecycle phase) the customization is done, as this has significant impact on the details of the technologies that can be used.

Reflections on the Format

A main goal of the seminar was to have a significant portion of the time for discussion. In order to achieve this, we decided to not require presentations from everyone associated with

a long introduction round. Rather, we decided to ask everyone for a poster to present her- or himself and describe the personal interest and relation to the topic. Overall this novel approach was well received by the participants. The poster walls were set up in the coffee break area outside the room. (Thanks to everyone at Dagstuhl for their support.) This allowed for a casual browsing of the posters in every coffee break during the seminar. Each poster also had a picture of the participant, this also helped to get to know each other.

2 Table of Contents

3 Overview of Talks

3.1 Imperative versus Declarative Process Variability: Why Choose?

Marco Aiello (University of Groningen, NL)

Joint work of Aiello, Marco; Groefsema, Heerko; Bulanov, Pavel

Variability is a powerful abstraction in software engineering that allows managing product lines and business processes requiring great deals of change, customization and adaptation. In the field of Business Process Management (BPM) the increasing deployment of workflow engines having to handle an increasing number of instances has prompted for the strong need for variability techniques.

The idea is that parts of a business process remain either open to change, or not fully defined, in order to support several versions of the same process depending on the intended use or execution context. The goal is to support two major challenges for BPM: re-usability and flexibility. Existing approaches are broadly categorized as Imperative or Declarative. We propose Process Variability through Declarative and Imperative techniques (PVDI), a variability framework which utilizes temporal logic to represent the basic structure of a process, leaving other choices open for later customization and adaptation. We show how both approaches to variability excel for different aspects of the modeling and we highlight PVDI's ability to take the best of both worlds. Furthermore, by enriching the process modeling environment with graphical elements, the complications of temporal logic are hidden from the user. To show the practical viability of PVDI, we present tooling supporting the full PVDI lifecycle and test its feasibility in the form of a performance evaluation.

3.2 My View on Customizing Service Platforms

Luciano Baresi (Polytechnic University of Milano, IT)

A service platform is a set of related services supplied by the same provider under a common umbrella and together with some shared qualities of service. Platforms as a service are a special class of the more general concept.

My interests in the customization of service platforms come from different motivations. Since I have been working on services at application level for years, moving to platforms provides a nice complement. The work done on eliciting the requirements for (self-)adaptive service applications easily fits the customization problem since it helps one understand what the platform is supposed to provide, and how it should be tailored to the different needs and situations. Similarly, the work done on adapting service compositions may provide interesting insights towards the definition of suitable customization means for the different service platforms. In these days, I am also interested in mobile applications and in the frameworks (platforms) that provide the bases to implement them. Since the resources of these devices are still limited, the customization of these platforms may help preserve them, and thus it may help the user keep the device (e.g., a smartphone) alive longer.

I am still a bit concerned, or confused, about the use of many different terms, like customization, adaptation, evolution, and maintenance, to mean similar and possibly related

concepts, but I am very interested in how the diverse services, and the infrastructure that operates them, can evolve in the different phases of the platform's life-cycle. Run-time changes, and the correctness of the new platform, are also particularly intriguing.

3.3 Dynamic Product Lines using the HATS framework

Karina Barreto Villela (Fraunhofer IESE, DE)

Typical Software Product Lines (SPL) approaches do not focus on dynamic aspects, and the reconfiguration of products occurs mainly statically at development time. Dynamic Software Product Lines (DSPL) enable a product to be reconfigured dynamically at runtime, which can be understood as the transformation of a product into another valid product without any kind of interruption in its execution. The reconfiguration, in this context, takes place without the need to halt the system, recompile and redeploy. From a technical perspective, dynamic reconfiguration is a challenging task due to reasons such as ensuring that dynamically updated systems will behave correctly or ensuring that no state data is lost. Moreover, from the Product Line (PL) perspective, not all technically possible changes in a running system are valid and make sense. In order to preserve the consistency of the PL products when reconfigured at runtime, there must be a way to restrict the adaptations that can be performed at runtime.

Fraunhofer IESE has added support for runtime product reconfiguration to ABS (an abstract but executable modeling language developed in the HATS project), by adding a dynamic representation of the possible product reconfigurations at runtime and a state update element responsible for data transfer, and by using the metaABS tool developed by the University of Leuven, which allows deltas to be applied at runtime.

3.4 Quality-Aware Product Configuration

Karina Barreto Villela (Fraunhofer IESE, DE)

The configuration of concrete products from a product line infrastructure is the process of resolving the variability captured in the product line according to a company's market strategy or specific customer requirements. Several aspects influence the configuration of a product, such as dependencies and constraints between features, the different stakeholders involved in the process, the desired degree of quality, and cost constraints. Fraunhofer IESE has developed a quality-aware configurator in which the user specifies the key product features and its quality concerns and cost constraints, and the configurator gives close to optimal configurations based on the user's input. The configurator is based on the assumption that the selection of a feature has an impact in the quality attributes of the final product, as well as the interaction among the selected features. This work included the integration of COSTABS (a static performance analyzer developed by the University of Madrid) to provide performance annotations to features and the first steps towards a reusable security feature model, which includes security feature implementations in ABS (an abstract but executable modeling language developed in the HATS project).

3.5 Customization of existing industrial plants to achieve modernization

Deepak Dhungana (Siemens AG-Wien, AT)

Industrial plants are complex and costly software-intensive systems that are operated over long periods of time. The modernization of plants with new technologies can significantly increase productivity while at the same time reducing energy consumption and environmental impact. Unfortunately, it is a daunting task to find out which new technologies are appropriate for an existing plant and to calculate the modernization costs and time. This process in practice today relies mainly on the experience and knowledge of key employees and is not well defined. Currently, there is no standardized method or tool for systematically eliciting customer requirements and plant data. In our ongoing work, we are developing methods and tools to support planning the modernization of complex industrial plants, which need to be adapted to meet new customer requirements and environmental constraints. In this project we first analyzed the current modernization process based on concrete scenarios and examples (e.g., improvement of a cooling system in a steel plant, in order to reduce the water consumption) to clearly define the requirements for tool development. The next step was to develop tools supporting the modeling of expert knowledge, the definition and formalization of modernization goals, as well as the definition of available resources. The optimization with regard to global constraints and objectives like productivity, quality, and economic impact is a complex task. We thus develop tools for capturing and modeling expert knowledge with the goal to assist in the selection of modernization packages based on customer requirements and in the creation of sales offers. The tools further support optimizing selected modernizations, for example, by comparing different modernization scenarios. The tools are flexible to allow their use within industrial plants in various domains.

3.6 Forward Recovery for Web Service Environments

Peter Dolog (Aalborg University, DK)

In web service environments there are web services which are often long running. In this situation, typical properties known from transactional management in databases, such as atomicity or isolation, are relaxed. This impacts on the transactions so that when some of the participants fail, they cannot easily undo outcomes of the web services participating in such transactions. We have studied this problem and designed an environment where we allow for forward recovery which means we allow for replacing failed web services with different ones which can deliver the work required to finish the transactions. The candidate web services are selected based on features which have been defined similarly as in product lines methodology, with mandatory and optional features. We compare and rank suitability of services according to matching between required feature model and those provided. The score is higher if there are more optional features satisfied with provided candidate service.

3.7 Customizing Service Platforms

Holger Eichelberger (University of Hildesheim, GE)

Customization of service-based systems is current practice in industry to meet the needs of customers in a qualitative and timely manner, e.g., to introduce novel functionality, to optimize the quality of service (QoS), or to realize integrations with existing systems. While many customizations can be implemented using service-oriented mechanisms such as (late) service bindings, several situations require the customization of existing services or the underlying service platforms, e.g., to develop domain-specific platforms. Here, systematic customization of services and service platforms can lower development effort and increase reusability.

Software Product Line Engineering (SPLE) is an industry best practice to achieve systematic customization in software systems. The key idea of SPLE is to focus on the differences (called variabilities) among similar systems. However, the existing methods and techniques for describing and realizing variabilities must be refined or extended to provide adequate support for service-based systems, including heterogeneity, open-world scenarios and runtime dynamicity which are common in service orientation. Thus, current challenges in customizing services and service platforms are: a) variability modeling for heterogeneous environments and, moreover, for entire software ecosystems, b) unified approaches to variability realization in service-based systems (in contrast to current individual and unrelated techniques), and for both, variability modeling and instantiation support for c) openness and extensibility and d) runtime variability.

Currently, we work on methods and techniques for addressing the challenges sketched above, in particular on

- Large-scale variability modeling, in particular in terms of the INDENICA variability modeling language (IVML), a textual language which provides concepts for variability modeling, runtime variabilities, openness, extensibility and QoS constraints.
- Generalizing and unifying the implementation of variabilities. Currently, we work on designing and realizing a common Variability Implementation Language (VIL).
- Increasing the flexibility of variability instantiations by separating the binding of variabilities and their functional code so that even the binding can vary according to properties of variabilities (meta-variability), e.g., to flexibly shift the binding time (currently applied in physical manufacturing of embedded systems).
- Observing the resource consumption of individual software parts at runtime, including services, components and variabilities. Our approach called SPASS-meter is designed for quality assurance for SPLE and, in particular, for supporting and simplifying the development of resource-adaptive software systems.

Future work is planned in particular on a) dynamic software product lines based on resource measurements and enhanced meta-variability, b) quality and resource aware variability modeling and c) large-scale variability modeling as well as supporting techniques.

3.8 SPASS-Meter – Monitoring Resource Consumption of Services and Service Platforms

Holger Eichelberger (University of Hildesheim, GE)

Monitoring the resource consumption of a system supports the operationalization of quality requirements, supports quality assurance, provides a basis for the estimation of energy consumption and supports the realization of (resource-aware) adaptive systems. Currently, resource consumption is typically measured on application level, on operating system level or, in contrast, on the level of individual classes. As also expressed in discussions during this seminar, there is a clear need to provide such measurements also for units within programs such as individual application services, for the underlying service platform or for technical services within the service platform.

In this talk, we present SPASS-meter, a novel monitoring approach, which enables the observation of resource consumptions for user-specified semantic units of software systems such as services, components or variabilities. In SPASS-meter, these semantic units are defined in the so-called monitoring scope specification, including the individual resources to be monitored as well as the monitoring depth, i.e., whether dependent functionality in related services, the service platform or in libraries shall be considered or not. SPASS-meter aggregates the resources consumption of these semantic units at runtime and allows comparing the consumptions with those on application and system level. Currently, SPASS-meter supports the monitoring of Java applications and, in particular, of Android Apps. As monitoring tools such as SPASS-meter execute additional code for probing and analysis, they cause a certain memory overhead. We conclude that the monitoring overhead created by SPASS-meter is reasonably small compared to the overhead of recent tools such as OpenCore or Kieker, in particular regarding the provided flexibility and functionality of SPASS-meter.

3.9 On-the-Fly Computing – Individualized IT Services in Dynamic Markets

Gregor Engels (University of Paderborn, GE)

Due to a steadily increasing market and budget pressure, the development and maintenance of IT systems have to become more efficient and more effective in the future. The traditional approach of software procurement by employing expensive and inflexible standard IT solutions or by purchasing individually developed software systems is obviously not a solution in the future. The new approach of cloud-based services allows an on-demand usage of software solutions and might be a first step in the direction of a more efficient and effective procurement of IT solutions. Combining this with the paradigm of service-oriented architectures, individualized IT services might be composed and used to fulfill certain business demands.

This service-oriented paradigm combined with the idea of deploying services in the cloud was one of the motivating starting points of the Collaborative Research Center

(CRC) 901 On-The-Fly Computing (OTF Computing), which is funded by the Deutsche Forschungsgemeinschaft (DFG) and conducted at the University of Paderborn since 2011.

The objective of CRC 901 – On-The-Fly Computing (OTF Computing) – is to develop techniques and processes for automatic on-the-fly configuration and provision of individual IT services out of base services that are available on world-wide markets. In addition to the configuration by special OTF service providers and the provision by what are called OTF Compute Centers, this involves developing methods for quality assurance and the protection of participating clients and providers, methods for the target-oriented further development of markets, and methods to support the interaction of the participants in dynamically changing markets.

In order to reach these objectives, computer scientists from different areas like software engineering, algorithms, artificial intelligence, distributed systems, networks, and security cooperate with scientists from the economics department, who are experts in organizing world-wide markets.

The CRC 901 is structurally divided into three scientific project areas: Project area A is devoted to the algorithmic and economic basic principles for the organization of large, dynamic markets. It concerns on the one hand the algorithmic procedures for the organization of large nets in general and for the interaction from participants in nets in particular; and on the other hand the economic concepts for incentive systems to the control of participants in markets.

Project area B investigates procedures for the modeling, composition and quality analysis of services and service configurations with the goal of an on-the-fly development of high-quality IT services.

Project area C develops reliable execution environments for the On-The-Fly Computing, and is concerned with questions of the robustness and security of markets, the organization of high-grade heterogeneous OTF Compute Centers and the execution of configured services by such Centers. In addition, there is an integrated application project which is concerned with optimization systems for supply and logistics networks and acts on a long-term basis as an application domain for the work of the SFB.

More detailed information about the CRC 901 can be found at http://sfb901.uni-paderborn.de/sfb-901.

3.10 Multi-level Service Management

Sam Guinea (Politecnico di Milano, IT)

Due to the growing pervasiveness of the service paradigm, modern systems are now often built as Software as a Service, and tend to exploit underlying platforms (Platform as a Service) and virtualized resources (Infrastructure as a Service). Managing such systems requires that we are aware of the behaviors of all the different layers, and of the strong dependencies that exist between them. This way we will be able to perform run-time customization and ensure that the functional and non-functional aspects of the overall system are always preserved, even in the wake of profound changes in the stakeholders' requirements and in the context of execution.

To accomplish this we are studying how to apply the traditional MAPE-K (Monitoring

– Analysis – Planning – Execution) control loop to such multi-level systems. We advocate that this will require novel data collection, analysis, planning, and execution mechanisms. Indeed we will need to collect runtime data from multiple levels at the same time, and be able to correlate them to build more detailed information of what is actually occurring inside the system. To this extent we have developed the Multi-layer Collection and Constraint Language. It allows us to define how to collect, aggregate, and analyze runtime data in a multi-layered system. We also present ECoWare, a framework for event correlation and aggregation that supports the Multi-layer Collection and Constraint Language, and provides a dashboard for on line and off-line drill-down analyses of collected data. Our initial empirical assessment shows that the impact of the approach on runtime performance is negligible.

In the future we will further pursue this research by evaluating our results in concrete real-world examples, through the collaboration with key cloud-based industrial partners. We will also study how the understanding that we gather of the system at runtime can be used to plan coordinated recovery actions at multiple levels. Indeed, we expect that the most cost-effective customization solutions would require coordinated intervention at multiple levels.

3.11 Customizable Reliability and Security for Data-Centric Applications in the Cloud

Waldemar Hummer (TU Vienna, AT)

Service-oriented computing (SOC) has become a prevalent paradigm for creating loosely coupled distributed applications and workflows. In parallel to SOC, Event-Based Systems (EBS) in various fashions (e.g., data stream processing) are gaining considerable momentum as a means for encoding complex business logic on the basis of correlated, temporally decoupled event messages. More recently, advanced virtualization and resource allocation techniques advocated by Cloud computing have further shaped the implementation possibilities of SOC and EBS. Clouds have proven to be an ideal environment for flexible and elastic applications which provide scalability, resource optimization, and built-in support for multi-tenancy. Ongoing trends in the area of Data-as-a-Service (DaaS) have spurred further research efforts towards robust data processing services, leveraging the benefits of the Cloud.

Distributed computing systems in general, and applications in the Cloud in particular, are often burdened with stringent requirements concerning reliability and security, dictated by business objectives (e.g., cost-benefit tradeoffs), contractual agreements (e.g., service level agreements, SLAs), or laws. Customized support for reliability and security in service platforms is a key issue. One approach to reliability is software testing, which attempts to identify and avoid software-induced faults in the first place. A second important aspect of reliability is adaptability and fault-tolerance, which involves different runtime challenges such as fault detection, isolation, or recovery. Additionally, due to the multi-tenancy inherently encountered in Cloud environments, security and access control play a crucial role for application provisioning. Consideration of these aspects in the software development and validation process requires precise knowledge about the type and nature of potential threats to reliability.

Within our work we tackle the aforementioned challenges and present novel techniques for reliable and secure provisioning of data-centric service platforms and applications in the Cloud. We strive for a robust, scalable, and secure execution environment for applications to integrate services and data from a plurality of sources, generating added value for service consumers. During the development phase, applications are systematically tested for incompatibilities and integration issues. At runtime, platforms should leverage Cloud virtualization to ensure reliability and efficiency (elastic scaling, minimal resource allocation, optimized load distribution). Moreover, customized security policies need to be enforced to assure responsibilities and avoid unauthorized access.

3.12 Adaptation in complex service ecosystems

Christian Inzinger (TU Vienna, AT)

Our current research deals with customization through adaptation of complex service ecosystems operating in dynamic environments such as cloud computing systems. Based on our work on fault detection and identification we model monitoring and adaptation behavior of complex applications in a unified manner to allow for optimized deployment of necessary control infrastructure.

3.13 A Library for Green Knowledge

Patricia Lago (VU University Amsterdam, NL)

Joint work of Lago, Patricia; Gu, Qing

In spite of the investments in green ICT, industry and research both lack reusable green practices including operational actions to re-green ICT, metrics, and examples of achieved results. Such green action can include optimizations in customized cloud provisioning, but also reusable patterns for engineering software exploiting service oriented principles.

Another problem is the lack of alignment between economic impact and environmental effect in green practices. If green practices do not lead to an explicit (and significant) reduction of costs (hence increase in revenues) they are nice but not part of the business strategy of the company.

To address these two problems, in this project an online-library for green practices has been built. This library provides a collection of 258 reusable green ICT practices with explicitly documented environmental effects and economic impacts, based on which companies are able to select and justify green ICT practices that fit best their needs.

While green practices so far mainly focus on non-software related actions, research is maturing toward energy efficient and environmental sustainable software service engineering. Future optimizations (green actions) will hopefully focus on how to achieve green services and how to combine them in greener service-based applications.

3.14 Cloud Computing as a Service Platform for Mobile Systems

Grace Lewis (SEI, USA)

Cloud computing infrastructures are used by organizations to provide access to large public data sets such as maps and images from mobile devices, and to host mobile applications outside of the enterprise to support front-line employees such as sales personnel. An additional use case that is at the intersection of mobile and cloud computing is to use the cloud to perform computation-intensive activities on behalf of mobile devices such as is currently done by Apple Siri, Google Glass, and the coming soon Apple iWatch. The latter use case is the one that is of interest from the perspective of customizing service platforms. This presentation discusses cloud computing as a service platform for mobile systems in the context of cyber-foraging – the leverage of external, nearby resource-rich surrogates to augment the capabilities of resource-limited mobile devices. It presents two types of cyber-foraging – code/computation offload and data staging – as well as the challenges of customizing surrogates as service platforms.

3.15 Cloudlet-Based Cyber-Foraging

Grace Lewis (SEI, USA)

Cloudlet-Based Cyber-Foraging is a strategy for extending the computation power of mobile devices by offloading resource-intensive computation to cloudlets – discoverable, generic servers located in single-hop proximity of mobile devices. We present the basic of cloudlet-based cyber-foraging in addition to future work in this area to address system-wide quality attributes beyond energy, performance and fidelity of results.

3.16 Customizing Platforms by Higher-Order Process Modeling: Product-Lining, Variability Modeling and Beyond

Tiziana Margaria (University of Potsdam, GE)

Joint work of Margaria, Tiziana; Steffen, Bernhard; Neubauer, Johannes

(Business) Process modeling languages like BPMN2 are static in the sense that they determine at modeling time which activities may be invoked at runtime and where. We overcome this limitation by presenting a graphical and dynamic framework for binding and execution of (business) process models. This framework is tailored to integrate

1. ad hoc processes modeled graphically,
2. third party services discovered in the (Inter)net, and
3. (dynamically) synthesized process chains that solve situation-specific tasks, with the synthesis taking place not only at design time, but also at runtime.

Key to our approach is the introduction of type-safe stacked second-order execution contexts, that allow for higher-order process modeling. Tamed by our underlying strict service-oriented notion of abstraction, this approach is tailored also to be used by application experts with little technical knowledge: users can select, modify, construct and then pass (component) processes during process execution as if they were data. The approach has been applied to a concrete, realistic (business) process modeling scenario: the development of Springer's browser-based Online Conference Service (OCS).

The most advanced feature of our new framework allows one to combine online synthesis with the integration of the synthesized process into the running application. This ability leads to a particularly flexible way of implementing self-adaption, and to a particularly concise and powerful way of achieving (re-)configuration via variability not only at design time, but also at runtime.

3.17 Platform Architectures

Nenad Medvidovic (USC – Los Angeles, USA)

The talk explores different views of service platform from the perspective of architectural style and architectural building blocks (specifically, connectors). An argument is made that a platform in this context is a middleware platform or a framework. Customization, then, boils down to customizing the middleware or framework. These are software systems in their own right and suffer from many architectural problems common to software systems. Grid service platforms are presented as an example case study. A number of open issues are identified as research challenges.

3.18 Variability Modeling & Management

Nanjangud C. Narendra (IBM India – Bangalore, IN)

Our work is motivated by the need to improve productivity of software development solutions, in particular, SOA-based solutions, in the IT services industry. Traditional approaches have involved the development of solutions from scratch in every customer engagement. To that end, we have developed the Variation Oriented Engineering (VOE) approach towards developing reusable SOA-based solutions, by modeling variations in those solutions as first-class entities. Currently our work has spanned the following topics:

- Variation Oriented Service Design for deriving variants from Business Process specifications
- Automated change impact propagation
- Variability Modeling for determining legal variants
- Variant and Version Management in Business Process Repositories

We foresee the following challenges in Variability Management:

- Formalizing Variability via algebraic approaches
- Integration with Business Process Management
- Lifecycle-based approach towards Variability Management
- Variability Management at runtime

Our future work will comprise (but not be limited to) the following:

- Variability Algebra
- Integration with adaptive workflow
- Variability at runtime
- Integrating variability into service ecosystems

3.19 Customized Mashups with Natural Language Composition

Cesare Pautasso (University of Lugano, CH)

End-User Development (EUD) is an emerging research area aiming at empowering non-technical users to somehow create or design software artifacts. Web mashups provide a high potential for EUD activities on the Web. Users on the Web can tap into a vast resource of off-the- shelf components in order to rapidly compose new, custom-made, lightweight software applications called mashups. In this presentation we have demonstrated JOpera (http://www.jopera.org) a visual service composition tool for Eclipse and NaturalMash a natural mashup composition tool that combines WYSIWYG, programming by demonstration and constrained natural language within a live programming environment that lets end users interactively specify the behavior of custom-made mashups that are built on-the-fly.

More information:

- S. Aghaee, C. Pautasso, Live Mashup Tools: Challenges and Opportunities, accepted at the First ICSE International Workshop on Live Programming (LIVE 2013), San Francisco, USA, May 2013.
- S. Aghaee, C. Pautasso, EnglishMash: usability design for a natural mashup composition environment, 4th International Workshop on Lightweight Integration on the Web (ComposableWeb2012) at ICWE 2012, Berlin, Germany, July 2012

3.20 Challenges of offering customizable domain-specific business processes as a service

Manuel Resinas Arias de Reyna (University of Sevilla, ES)

Joint work of Resinas Arias de Reyna, Manuel; Ruiz Cortés, Antonio

The growing demand of business–driven IT systems as well as the rise of Software as a Service (SaaS) has led to the creation of a category of SaaS known as Business Process as a Service (BPaaS). In them, service users can access a set of domain-specific processes, customize them according to their needs and enact them in the cloud. In this scenario, several challenges arise. On the one hand, current mechanisms to manage the variability in business processes should be extended to allow the customization not only of the control flow, but also of other perspectives of the process such as the organizational or the performance perspective. On the other hand, compliance with regulations, best practices and internal policies is a key aspect in organizations nowadays and may vary significantly from one organization to another.

Therefore, BPaaS must provide their users with mechanisms to ensure their processes are customized and enacted according to the regulations that are relevant for the users. Our current work focus on facing these challenges leveraging the work we have done on business process compliance management systems and on models and techniques for the management of process performance indicators and human resources

3.21 Customization of Large, Complex Systems

Klaus Schmid (University of Hildesheim, GE)

The major thrust of our work is on the customization of large, complex systems and in particular software ecosystems. We are in particular using product line engineering technologies to perform the necessary kinds of customizations. A particular challenge in the service platform is the need to support a range of very different artifacts and the need also to go to later binding times like initialization time or runtime. This requires on the hand a complex coordination among individual customizations to support the integrated customization. On the other hand it requires different techniques to address the later binding times.

A further challenge is the overall size and complexity of the platforms, which may often give rise to many thousand variation points.

3.22 Service Networks for Development Communities

Damian Andrew Tamburri (VU University Amsterdam, NL)

Communities of developers have rapidly become global, encompassing multiple timezones and cultures alike. In previous work we investigated the possible shapes of communities for software development. In addition, we explored mechanisms to uncover communities emerging during development. However, we barely scratched the surface. We found that development communities yield properties of dynamic change and organic evolution. Much work is still needed to support such communities with mechanisms able to proactively react to community dynamism. We argue that service-networks can be used to deliver this support. Service-networks are sets of people and information brought together by the internet.

The missing keystone is to support social communities with an innovative and pro-active mechanism operating through services. The research hypothesis that drives the work in this paper is quite simple and equally intriguing: social communities of developers can be supported by a global network of software and socio-technical services, spanning different organisations, sites, timezones and cultures. The result is a service-network that blends the internet of services with large-scale, adaptable choreographies to deliver a powerful and scalable solution that adapts to the changes of a community. On one hand, software services are pieces of software operating under a service-dominant logic. These pieces of software collaborate together across the web using standard protocols, to deliver complex, adaptable functionality (e.g. cloud-based functionalities such as GoogleDocs). Much literature in service

sciences provide ways to identify, monitor and adapt software services On the other hand, socio-technical services are hybrid human and software services, i.e. services that explicitly mediate the collaborative work of people within a social community, e.g. by fostering relevant community aspects or by increasing situation awareness of community members.

http://www.dagstuhl.de/mat/Files/13/13171/13171.TamburriDamianAndrew.Paper.pdf

3.23 Customizing Science Gateway Platforms via SaaS Approach

Wenjun Wu (Beihang University – Beijing, CN)

A Science Gateway is a computational web portal that includes a community-developed set of tools, applications, and data customized to enable scientists to run scientific simulations, data analysis, and visualization through their web browsers. Because scientists always have different requirements for their data processing pipeline, science gateway developers have to cope with the customization of GUI, workflow, applications and runtime environment. So the research problem is how to effectively support multi-tenant customization in science gateway platforms.

The talk introduces a SaaS framework to enable customization of life science gateway through four levels: GUI, workflow, bio-application and workspace.

It allows users to rapidly generate Web GUI and deploy their pipelines in heterogeneous environments

3.24 Service-based Platform Integration and Customization

Uwe Zdun (University of Vienna, AT)

In service-based integration, platform customization, and similar areas, our research group addresses the following challenges: understand and support architecture and design decision making; link architectures, designs, and implementations; automate recurring tasks; base these solutions on time-proven architectural knowledge; provide empirical evidence. Our work and interests in this area concerns

- reusable decision models and corresponding tools
- design patterns; model-driven techniques (MDD) to bridge between architectural decisions and designs
- view-based architecture for service platform
- MDD generators
- full traceability
- empirical studies

3.25 Customizing Service Platforms — new or have we seen this before?

Frank van der Linden (Philips Medical Systems – Best, NL)

I have the feeling that, although the problems are new, I have seen this before. Over the year people have struggled with customization or variability at higher levels of abstraction. The initial programming languages tamed the variability into a few constructs: if, case, while, ... and goto. When the programs became complex, functions and subroutines were introduced. This added parameters and recursion to the palette of variability. Separate compilation added IFDEF. Again systems became complex, and again variability needed to be tamed. Functions were grouped into object classes, and related data into objects. This added inheritance to the mechanisms variability. A next step added configurations of object classes into components.

Each time, the new concept reduced the choices of how variability can be used. Special configurations were supported others were not. Sometimes a new mechanism was introduced, but it also kept the programs comprehensible, because the mechanism provided abstraction and hiding of internal details. Presently configurations of components are combined into services. This provides, again, abstraction and hiding of internal details. The situation is somewhat different because now it is apparent that services can and will be provided by different providers. This was also the case for previous mechanisms, as there are third party libraries, object frameworks, and COTS. However, the services structure is getting complex, and we cannot track, control or trust all the code will be executed. As in previous times we have to apply the mechanisms we have used before – variability management, negotiation, service configuration modeling, reduction of configurations that will be allowed to those for which the trust can be assessed. This still needs partially to be done, and that is the goal of this seminar.

4 Working Groups

4.1 Quality Assurance and Validation in Customizable Service Platforms

Deepak Dhungana (Siemens, AT)

Participants: Deepak Dhungana, Waldemar Hummer, Georg Leyh, Frank van der Linden, Antonio Ruiz Cortés

4.1.1 Introduction

With the increasing importance of service oriented applications in many businesses and new application fields such as cloud computing, many new challenges are arising in this area. The initial effort required for customization of a service platform or associated services is already very high, but at the same time the nature of these applications requires them to consider customizations at runtime, too. Apart from the technical difficulties related to customization

of the functionality, we see serious needs to discuss the impact of developing and deploying customizable services on the quality of the overall system.

4.1.1.1 Quality Assurance

Software quality assurance is often defined as a means of monitoring the software engineering processes and methods used to ensure quality. Quality assurance therefore needs to consider both how such services can be developed or deployed and the runtime monitoring to ensure required quality.

Service based applications (SBA) are described by functional and non-functional properties. Non-functional properties include some QoS dimensions such as accuracy, coverage, network-related QoS, performance, reliability, robustness, and scalability. Our discussion in this section is focused on the relationship between customizability of applications and the impact on the quality attributes.

4.1.1.2 Open Questions

We summarize the key issues related to customization of SBA and quality of the overall system.

- What is the effect of customizability (having a flexible and customizable platform) on the quality of the system? What kind of activities is additionally (as opposed to fixed/monolithic environments) needed during development and runtime to ensure required quality?
- What support can be provided to different stakeholders (which roles do they have?) in customizing a service based application to achieve the required quality?
- How can variability of the quality attributes of a SBA be defined, so that the customization processes and tools can deal with them?

4.1.2 Case Studies

In oder to demonstrate the industrial need for "quality-aware" service based applications, we present two case studies.

4.1.2.1 Case Study 1 – Image Processing Service Platform

The first case study is about an image processing service platform at Philips medical systems. The platform provides image processing services to clients that are, in general, hospitals. Dependent of the hospital infrastructure, but also to the terminal capabilities, more or less processing can be done at the client side. A solution for this is to split the service platform in several abstraction layers and provide services in between the abstraction layers. Dependent on the situation, the client can take one or more top level abstraction layers and the server provides the remainder. Note, that the client platforms are not in control of the service provider and can be very diverse. The hospital pays for the services provided. It is the business model that determines how this is done, and will not be part of this description.

This all needs to be done in an environment where several quality concerns are important. These are related to performance, latency, throughput, but also to security, privacy and legal rules. All these quality requirements may vary dependent on the customer. In addition, the user expects personalization.

In summary, the customization deals with personalization, quality requirements, and the level of abstraction that will be provided.

4.1.2.2 Case Study 2 – Logistics Management Platform

The second case study is about a platform for logistics, e.g., Warehouse Management solutions or solutions for spare parts logistics. Usually, the platform vendor provides (delivers) many variants of the platform based on domain specific service platforms. The goal is to reuse the service platforms for different customers, however, different users usually have a different prioritization of qualities (e.g., a small warehouse may prefer a very cost-efficient solution, which may be based on open source software, while a big warehouse needs a solution that is available 24/7). Therefore, we need to customize the prioritization of qualities.

In spare parts logistics, the platform usually makes heavy use of existing legacy systems installed at the end-users site. Sometimes, different legacy systems are deployed e.g. in different countries. Here we need a uniform customization, even if the services use different legacy platforms / data.

4.1.3 Identified Challenges

Based on the case studies, we have identified the following challenges related to quality assurance in customizable service platforms.

4.1.3.1 Quality Monitoring

Many service platforms use resources from existing legacy systems. Those systems usually have no formal SLAs. To model qualities of systems that include legacy systems, at least the quality monitoring for legacy systems is necessary. The monitoring information can be used instead of a formal SLA, e.g. to calculate the overall availability of the SBA.

4.1.3.2 Quality(-Driven) Adaptation

As seen in case study 2, for reusing domain specific service platforms it is necessary to adapt the prioritization of different qualities. Since the qualities of the SBA need to be adjusted, a check whether these qualities can be achieved with the existing base services needs to be made. If the base services cannot be used to achieve the necessary quality, other services must be checked for conformance.

4.1.3.3 Quality Assurance in the Development Lifecycle

Quality assurance has to be performed at several stages in the development lifecycle. Already at development time software engineering has to provide the right mechanisms to do quality assurance. In addition, certain development patterns may be used to ensure a level of quality assurance upfront. A candidate service needs to be tested. For instance, it has to become clear whether it performs according to its SLA (see below). This action is similar to acceptance tests for COTS integration in the software. However, run-time testing needs to be added, as the service provision may change unexpectedly. In this case a fault tolerance mechanism needs to be added. In the case that a service fails to perform with the right quality, this should be repaired. There are several options for this: renegotiation, adding a similar service to the configuration, or escalating the fault to the higher level – i.e. announce on-conformance to the own SLA to the client.

4.1.3.4 SLA Modeling

An SLA (Service Level Agreement) is the interface of a service towards its clients. Based on the SLA a client decides whether to use the service. An SLA describes the function that is provided; in addition it describes the level of relevant qualities the function is delivered. Today a SLA is usually static, which means that for each quality a certain quality range is given. Note, that this implies that there is a metric to measure the quality level. Sometimes a few configurations are offered that the client might select.

At the client side the situation is less simple. Quality levels might be related by conditions, such as: if quality A is less than level U then quality B must be more than level V. Also qualities might have priorities: If one of the quality levels needs to be dropped than C is the last one to choose from. The client also might have quality requirements that cannot be measured, e.g. data quality (such as: does the picture show the right features).

This all asks for a model to incorporate all the qualities that are relevant for the client and this needs to be used for SLA management. A model will describe a region in a multi-dimensional space, where qualities are dimensions. The region is the level of acceptable quality. Note that in this model cost is just one of the quality dimensions.

4.1.3.5 SLA Management

SLA management deals with negotiating and monitoring the services provided. Negotiation deals with finding a service configuration that provides the service fitting in the acceptable region defined by the model. Note that we may need a configuration, as it might be the case that a single service cannot provide the right SLA; e.g. if two services both have an availability of 98%, using them both for 50% of the time we can get an availability level of 99%. Note that finding the right configuration can be posed as an optimization problem. In the case that a standard modeling language exists for SLA, it might be expected that service providers might offer more complex service offerings, which makes negotiation more complex as well.

The importance of temporal awareness is rising in SOA solutions. Temporal awareness refers to managing service demands and offers which are subject to validity periods, i.e. their evaluation depends not only on QoS values, but also on time. For example, the QoS of some web services can be considered critical in working hours (9:00 to 17:00 from Monday to Friday) and irrelevant at any other moment. Until now, the expressiveness of such temporal–aware specifications has been quite limited. This issue also makes negotiation and monitoring more complex.

After negotiation the SLA management is not finished. At run-time SLA monitors the services for several reasons:

- Does the service work according to its SLA? If not, then fault management needs to be incorporated to manage the fault, which might lead to re-negotiation, changing the service configuration, or escalation to the client.
- Establish levels of qualities that are not mentioned in the SLA, or that might be difficult to measure

4.1.4 Possible Solutions

Some possible solutions to deal with the identified challenges could be summarized as follows. However, these are rather open issues, that need to be elaborated further and are topics for future research.

4.1.4.1 SLA Management Facilities

There are a number of activities that may be performed one or more times during the SLA management lifecycle and the operation of a SLA-aware service platform such as:

- Prior to advertising an offer (quality guaranteed by the service provider) and issuing a demand (customer quality requirements), they both should be checked for consistency, i.e. to check that they do not have any internal contradictions.
- Checking whether an offer fulfills a given demand, i.e. checking them for conformance a.k.a compliance.
- Finding the optimal offer out of a set of offers conformant to a given demand, mandatory if we want to automatically create the optimum SLA.
- Checking whether an SLA has been violated.
- Finding out all the SLAs which are 'outside the law' defined by a set of governance policies.
- Finding out the set of SLAs that may be violated during a given time window with a cost below a given amount.
- Giving explanations about why an offer or demand is not consistent, why there is no possibility to reach an agreement, why the SLA has been violated, etc.

The degree of difficulty of implementing these facilities depends on the degree of expressiveness of the SLA model used. Furthermore, implementing some of these facilities may lead to NP-hard problems, especially if features such as conditional terms, temporal-awareness, non-linear selection criteria are allowed.

These activities could be organized by using a catalogue of common operations a.k.a facilities (this approach has been widely accepted in the Automated Analysis of Software Product Lines). In this catalogue it would be possible to distinguish between basic and composite operations (only if it is possible to define it as a combination of basic operations) as well as to provide a reference implementation.

4.1.4.2 Support during development

The issue of customization and service quality assurance influences all phases of the service engineering lifecycle, including design, implementation, validation, deployment, and runtime. We categorize this lifecycle into development phases (design, implementation, validation) and runtime phases (deployment, execution time). To assure quality in service environments, it needs to be clearly understood which quality parameters are influenced by different parts of the lifecycle.

During the development phase, all aspects related to quality assurance need to be collected and encoded in a multi-dimensional quality model. The quality model should capture the requirements, goals, and risks associated with different quality assurance scenarios. The quality model then serves as the basis to derive measurable quality metrics as well as potential actions to take in case of quality issues (e.g., SLA violations). We distinguish between static consistency checks and dynamic conformance checks. Static consistency checks are required to determine whether the quality model can generally satisfy important properties such as soundness, completeness or satisfiability. Dynamic conformance checks are employed to determine, for concrete instantiations of service platforms, whether the current state corresponds to the target quality requirements and goals.

The development phase is particularly important for quality assurance, for two main reasons. Firstly, certain quality characteristics like correctness, availability, consistency, or fault tolerance need to be modeled and systematically tested for. Secondly, all capabilities

required to assure quality at runtime (e.g., monitoring, optimization, adaptation) need to be accounted for during the development phase.

4.1.4.3 Runtime Support

During runtime, one of the key concerns is to monitor the conformance of the service platform to customized quality metrics, in order to timely react to occurring SLA violations or potential upcoming quality issues. In recent years, event-based monitoring based on the complex event processing (CEP) paradigm has emerged as the key technology to support loosely coupled monitoring infrastructures. CEP leverages the concept of complex events, which aggregate and correlate streams of raw events to provide higher-level knowledge about the current quality and health status of a system, i.e., service platform. Efficiency and non-intrusiveness are among the core research questions related to monitoring of service quality.

Certain quality problems allow timely correction by adapting the system within the current phase of the provisioning lifecycle. For instance, a service platform which is configured to react to load bursts should be able to acquire new computing resources as soon as a quality degradation is measured at runtime due to high request load. However, if elasticity is not correctly implemented and runtime monitoring detects that the acquired resources are not released once the load decreases, the problem needs to be escalated and fixed in previous phases, re-iterating through the development phases of design/implementation/validation. As of today, the research community still lacks a deep understanding of how to support this type of escalation by systematically modeling the connections between quality metrics and different development lifecycle phases.

4.1.4.4 Domain-Specific Solutions

In recent years, research and industry have experienced the emergence of new paradigms related to service platforms and service-oriented computing.

Arguably one of the most important trends is Cloud Computing, which introduces advanced virtualization and resource allocation techniques, providing for a new class of applications with a high degree of dynamism, scalability, and elasticity. These elastic applications often require non-trivial re-configurations, because once the elasticity state is changed (i.e., a resource gets added or removed), the connections and dependencies on potentially many other resources need to be updated and re-configured. To ensure that the transitions between elasticity states function properly, comprehensive testing and verification efforts need to be undertaken. Moreover, quality agreements in Cloud Computing are inherently related to multiple tenants. To avoid redundant detection and enforcement efforts, quality assurance mechanisms should be efficiently tailored to multiple tenants, for instance by grouping together tenants with similar or overlapping quality requirements.

A second important trend is the increasing integration of humans in the service delivery process. Novel concepts like crowd-sourcing or human-provided services (HPS) are finding adoption in service-based applications. Since humans operate distinctly different from machines, advanced quality characteristics and metrics such as trust, skills, or reputation need to be taken into account.

4.1.5 Open Issues

Some issues need further discussions and more industrial cases to support their practical relevance.

4.1.5.1 Variability in SLA Definitions

Currently, it is not possible to describe variability in SLAs. In addition, it is not possible to check SLAs that contain variability information for conformance with other SLAs. E.g., a variable SLA may state that it has a const / availability variation point. 99% availability with 0.01 EUR per call for low budget, 99.99% availability with 0.05 EUR per call for high availability customers. Three services are available: Service A with 99.999% availability and 0,03 EUR / call, Service B with 99.9% availability and 0,005 EUR per call, Service C with 99,99% availability and 0,01 EUR per call.

Service A would partially comply (only high availability customers), Service B would partially comply (only low budget customers), Service C would fully comply to the variable SLA. A formal language and calculus for this kind of problems is currently missing.

4.1.5.2 Systematic Decision Support

Quality assurance, in customizable service platforms, requires decision making in complex multi-dimensional spaces. This implies that automatic decision support is requested. It is needed in static and run-time SLA management during negotiation and configuration selection to decide which offering fits best to the request. Decision support needs optimization algorithms to execute its ask.

The decision making has to find matches of regions in multi-dimensional spaces, where certain dimensions are described as ranges, others are described as a probability distribution, and others are even not really measurable, but are based on human "expert" based decisions. In addition there are relationships between different dimensions, and there are dimensions that have higher priority than others. Optimization algorithms exist, but they often are restricted to certain kinds of input only.

The main issue here is to find algorithms that deal with the complexity above. In addition, it needs to be made clear in which form the regions in the spaces can be described. This latter point is related to SLA modeling, but modeling needs to address the issue of decidability as well.

The output of decision support will be the best offer that fits the requirement, but it should also indicate margins between the solution and the best fit. This margin should be expressed in such a way that it can be understood by the human client.

4.1.5.3 Integrated Tooling for SLA Languages and Models

Probably it makes no sense to design a universal (domain-independent) language to describe SLAs. In fact, the WS-Agreement specification identifies up to nine regions where for each and every region a DSL (Domain Specific Language) must be provided by the SLA editor. Thus, it is possible to have an unbounded number of different WS-Agreement compliant DSLs. Probably, this circumstance may explain at a given extent why the WS-Agreement has not been (widely) used.

However, it does not seem reasonable to implement the SLA management facilities (see above) from scratch to the management of the SLAs of each SBA, especially for very expressive SLAs. Therefore, there is an important room for improvement in this issue.

4.1.5.4 Bootstrapping

In this document we have discussed various aspects of quality assurance, including modeling, monitoring, adaptation, decision support, and more. An additional complexity dimension is

the question of how quality requirements can be introduced into an (existing) platform in the first place. Assume that the current state of quality provided by a service platform is expressed as X, and that the provider plans to achieve the customized quality level Y. First, the delta between X and Y needs to be identified. In order to enact the required changes, a semi-automated procedure can be employed to identify the required steps to be taken. For high-availability applications the key concern is that these quality adaptation steps should be enforced with the shortest possible downtime. Moreover, since these steps may involve non-trivial re-configurations and adjustments within the platform, a critical aspect of the adaptation procedure is to maintain correctness and not to introduce any new issues (e.g., bugs, misconfigurations).

4.1.5.5 Maturity Model for Customizability

In general terms, a maturity model can be viewed as a set of structured levels that describe how well the behaviors, practices and processes of an organization can reliably and sustainably produce required outcomes. In this sense, a maturity model can be used as a benchmark for comparison and as an aid to understanding – for example, for comparative assessment of different organizations where there is something in common that can be used as a basis for comparison. In the case of the CMM, for example, the basis for comparison would be the organizations' software development processes. (from Wikipedia)

Assuming that service platforms have a considerable number of customization points that may crosscut different perspectives implies that the customization degree may vary significantly among service platforms. Moreover, the existing dependencies among customization degree and other quality factors such as maintainability increase the complexity of this issue.

In these circumstances having a maturity model to assess the service platform's customization capabilities could be really useful for both customers and providers.

4.2 Mobility and Service Platform Customization

Grace Lewis (SEI, USA)

Participants: Luciano Baresi, Schahram Dustdar, Sam Guinea, Grace Lewis, Tiziana Margaria, Andreas Rummler, Karina Villela, Wenjun Wu, Uwe Zdun

Mobile computing is transforming the way in which people interact with the world and with each other far beyond the simple use of a smartphone as a communication device. In recent years, there has also been a rapid explosion of mobile devices and sensors that are not only pervasive but often interconnected. Mobility and ubiquity therefore create a potential for mobile devices to (1) become service platforms for local applications as well as service platforms for other nearby mobile devices and (2) extend their computational capabilities by taking advantage of service platforms in the cloud. This working group explores both options.

4.2.1 Mobile Devices and Customization

We define mobile device as any device that is battery-powered, has wireless capabilities, and a small form factor. Examples of mobile devices therefore include smartphones, tablets, wearable devices (e.g., Google Glass, iWatch), devices built on top of small single-board computers (e.g., RaspberryPi), sensors, drones, sports devices (e.g., FitBit), medical devices, and navigation devices.

Examples of design-time and runtime customization for service platforms deployed on mobile devices to serve local applications as well as other mobile devices include:

- Sensors to enable/disable
- Exploitation of context-awareness to adjust sensor sampling rates to drive energy efficiency or data quality/precision
- Data consumed and provided
- Definition of swarming behavior for groups of mobile devices
- Location of deployed sensors
- User interaction (e.g., touch, gesture) or action patterns based on specific applications or content
- Add/remove computation or change algorithms
- Communication mechanisms to enable/disable (e.g., WiFi, Bluetooth)

As can be seen in the previous examples, it is difficult to differentiate between services that are provided by the operating system, services that are provided as part of a more traditional service platform, and the applications themselves. This is probably due to the small form factor but also because mobile applications are tied to a specific platform/OS and typically make use of sensors that are part of the platform.

4.2.2 Mobility and Service Platforms

There are two areas related to mobility and service platforms that are interesting from a customization perspective:

1. Mobile device as a service platform: These are mobile devices that act as mobile limited-resource service platforms that can exploit on-board sensors and humans for data collection or collections of devices combining to form a single platform.
2. Cloud computing as a service platform for mobile systems: Mobile devices can use service platforms in the cloud in multiple ways:
   - Mobile device as a data collection platform that is uploaded to a service in a surrogate or the cloud
   - Surrogates as a service platform for multiple mobile platforms
   - Mobile devices connecting directly to services in the cloud

4.2.3 Sample Scenario: First Responders

There are multiple scenarios that would benefit from customizable mobile service platforms:

First Responders: Personnel operating in emergency situations can use a variety of mobile devices to support dynamic mission activities such as situational awareness, exploration of unsafe areas and medical assistance.

Asset tracking: Mobile devices can be attached to any asset to determine location, usage patterns, or environment characteristics. An example of an asset is a container that can be tracked from origin to destination, varying the data sampling and rate according to location.

Smart homes/cities: Data collected and potentially pre-processed by mobile devices that are spread throughout homes or cities can help in task automation, emergency detection and response, surveillance,

Remote locations: Mobile devices can be located or dispatched to locations where it is difficult, impossible or dangerous for a human to go to.

The first responder scenario is of particular interest from a customization perspective because of the dynamic nature of the environment as a disaster or emergency situation evolves from panic to medical attention to supplies to connecting people.

Imagine a scenario in which a bomb detonates in a very public place leaving lots of people trapped and hurt. In addition, the bomb damages the communication network. In this situation, first responders are equipped with smartphones and tablets with sensors and communication mechanisms that are particular to the type of emergency and network situation and can receive and install services on-the-fly. Disposable surrogates that can execute expensive computations and have access to the cloud are deployed in strategic locations. Robots with mounted cameras are sent in to explore the damaged areas and throwable cameras are used to create a picture of the damaged areas. Surveillance cameras in the area are automatically configured to capture high-resolution video of the affected areas.

As the scenario unfolds, first responder mobile devices use contextual information to adjust sensor sampling rates to extend battery life. Nearby mobile devices create an adhoc network to deal with the damaged network and combine to form a single platform where each devices performs certain tasks according to their capabilities. These mobile devices also reconfigure (manually or automatically) as deployed devices gather information about the context.

4.2.4 Open Challenges

Satisfying the scenario that was just described requires addressing a number of open challenges related to the customization of mobile service platforms:

- Challenges introduced by mobility: The main challenge is ephemerality — they don't last long and you don't know when they will fail. Because of limited resources, platforms and applications have to be very efficient in terms of energy and bandwidth usage and have to deal with unstable connection.
- Privacy, Security and Trust: Mobile devices that act as customizable service platforms have many issues related to data privacy and trust in new features.
- Multiple stakeholders have different concerns that could be in conflict — users, mobile peers, providers (platform, apps, network, storage), government agencies, and certification organizations.
- Multiple Roles: In a multi-platform scenario, devices may play multiple roles at different points in time — consumer, provider, host. Switching between roles requires, in addition, continuous discovery and coordination.
- Business models: Creating a business model that motivates users to add services on-the-fly is a challenge. There may be the need for third-party certifiers that certify that services do what they say they do.

4.3 Architecting for Platform Customization

Damian Andrew Tamburri (VU University Amsterdam, NL)

Participants: Florian Rosenberg, Cesare Pautasso, Damian Andrew Tamburri, Leonardo Passos, Nenad Medvidovic, Manuel Resinas Arias de Reyna, Patricia Lago, Peter Dolog, Gregor Engels, Nanjangud C. Narendra, Klaus Schmid[1]

The group discussed the architectural implications and underpinning of platform customization problems. First, the group explored the architecture decision to adopt certain technologies as opposed to others, for adaptation and architecture flexibility. These technologies include: REST vs JSON/RPC vs SOAP/RPC vs MQ. These decisions are revisited when needs arise for (re-)integration of platforms for networked organizations. These scenarios require a customization cutting across service definitions. The following discussions rotated around the following research questions:

RQ 1: What is customization?
- Depends on the context (service consumer, provider, platform)
- From consumer perspective it is: "Service selection, service configuration, platform configuration/constraint"
- From consumer/provider perspective it is: "Negotiable, flexible billing/accounting model"
- From platform: "Resource allocation, platform service selection"

RQ 2: How is customization expressed at the interface/service abstraction level?
- Request/job submission metadata/constraints
- Platform feature selection, activate/deactivate services for which you will be charged
- Product lines?

RQ 3: What are the platform mechanisms to allow customization?

Strategy pattern: one abstract interface with multiple implementations
Extension point: plugin additional implementations
Architectural patterns for customization *such as?*
Tuning/controller component for self-adaptive architectures

RQ 4: How can you design an architectural style for service-oriented architectures that facilitates customization?
- Loose coupling -> easier to customize
- Granularity of services: small -> easy to recompose
- Formalize customization with a customization algebra

RQ 5: How are existing services/platforms customizable? How are customization done today? — How is this unique to services/service platforms? How are different "features" of the platform customized and exposed?

- Separate branches in the code, then compile and deploy separate branches to enable different customizations
- One branch with feature toggles and turn toggles on and off at runtime through configuration

[1] Further information can be found at http://ep.sonyx.net:9000/dagstuhl

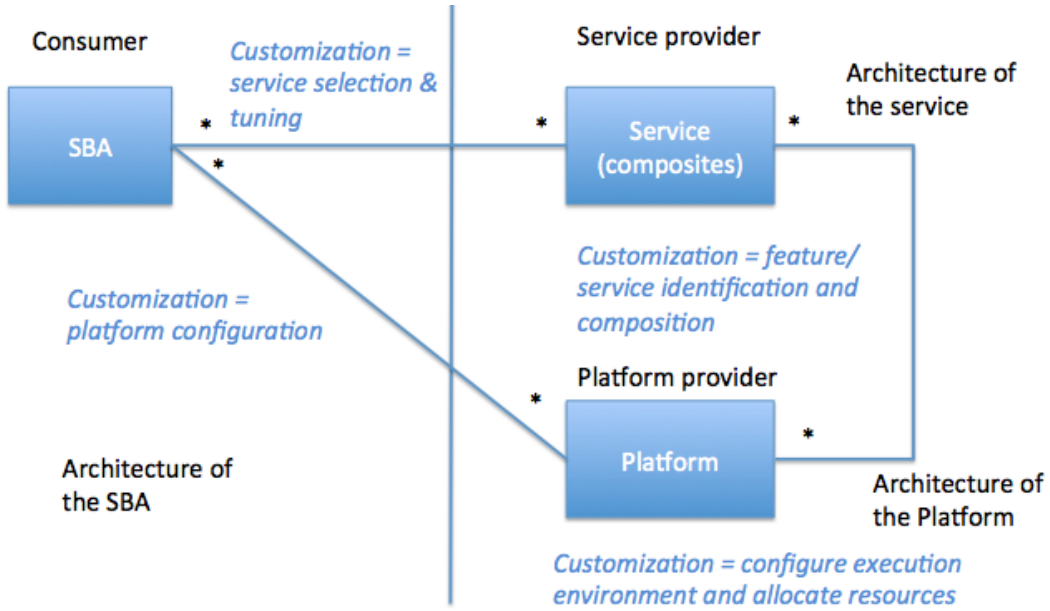

Figure 1 Architecture of the SBA.

- UI-level composition of widgets
- #ifdef
- if (config)
- Interface o = Class.new(config)
- Dynamic discovery and composition as a form of customization
- AOP for Devops languages

What are typical binding times:

- Very early (design time, select-integrate-test)
- Early (compile, deploy)
- Late (deploy, runtime)
- Very late (runtime after failure)

It is important to include the customization context as shown in Figure 1.

4.4 Energy-Aware Customization

Patricia Lago (VU University Amsterdam, NL)

Participants: Patricia Lago, Luciano Baresi, Sam Guinea, Grace Lewis, Marco Aiello, Holger Eichelberger, Nenad Medvidovic, Antonio Ruiz Cortez, Jacek Serafinski, Wenjun Wu

The group discussed what energy-aware customizations from the platform level up to the application level should entail. Requirements include measure, platform self-optimization, and mapping of the elements that belong to an energy context, both within and across levels. The unanimous conclusion was that:

1. past research in optimizations driven by scarcity of resources could be partially applicable for reducing energy consumption, *and that*
2. new research is needed due to the complexity of the current IT contexts, and due to the fact that energy efficiency requires tradeoffs between optimization costs and energy savings.

While promising, this area needs much more research in the future.

4.5 Customizing Service Platforms for Cloud Computing

Cesare Pautasso (University of Lugano, CH)

Participants: Florian Rosenberg, Waldemar Hummer, Christian Inzinger, Cesare Pautasso, Manuel Resinas, Peter Dolog, Klaus Schmid

Service Platforms for Cloud Computing are highly customizable. In this working group we have analyzed the variability points for Infrastructure-as-a-Service (IaaS) offerings and performed a practical comparison of concrete public cloud platforms (Amazon EC2, Microsoft Azure, Google Compute) and also the OpenStack framework for private clouds.

The variability points concern:

- the mechanism used for the compute, storage, and network abstraction offered by the platform
- the possibility of configuring image flavours (and how these flavours are defined)
- the way images are managed and whether it is possible to bring-your-own customized images for deployment
- the possibility to customize images upon their activation
- the structure of the metering and billing model of the provider (whereas most providers differ in the way they charge for using their infrastructure, it is not always possible to negotiate customized billing agreements with a provider)
- the mechanism used for offering elastic scalability (and whether it is possible to customize it with specific policies)
- the set of security mechanisms that can be enabled and the corresponding policies
- the presence of explicit "geographic regions" and how these are represented.

4.6 Customizing Service Platforms for Agile Networked Organizations

Damian Andrew Tamburri (VU University Amsterdam, NL)

Participants: Uwe Zdun, Georg Leyh, Karina Villela, Gregor Engels, Tiziana Margaria, Andreas Rummler, Deepak Dhungana, Nanjangud Narendra, Frank van der Linden, Damian Andrew Tamburri

The group explored the challenge of understanding how to customise service platforms to allow IT-intensive organizations to network with each other. Organizational networking

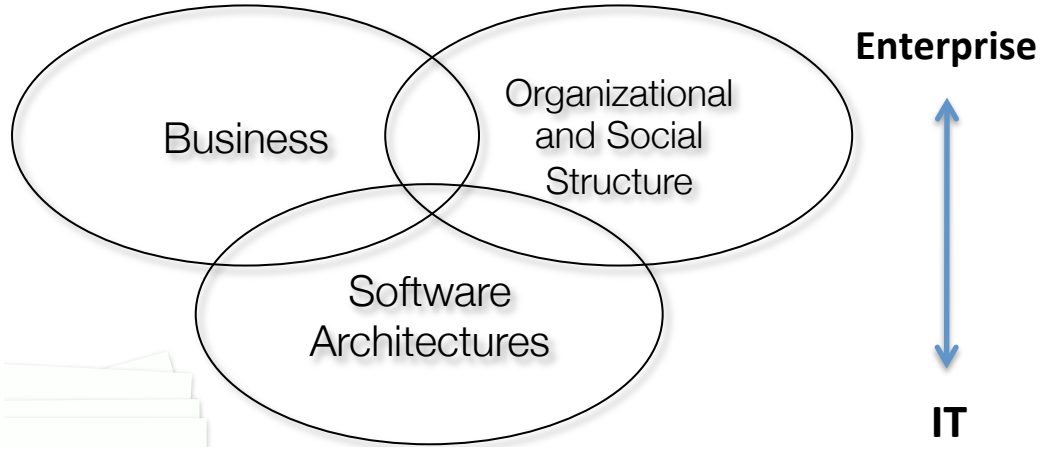

Figure 2 Relationship of Business, Organization, and Architecture in Networked Organizations.

scenarios include any situation in which a set of organizaitons partner up to achieve shared business goals. Similarly, organizations need to network when one organization subsumes or merges with another one. Making this transition into an "agile", i.e., adaptable, smooth, and painless transition is still a big challenge.

The group agrees that the customization problem at hand is a "Business-IT alignment" problem, since the agile networked organization stems to align business drivers with IT-Architecture and vice versa. Also, the problem is strongly affected by market speed, key driver for organizational dynamism. The organizational artefact, which is being adapted in the process of organizational networking, is the organizational and social structure emerging between networked organizations.

To evaluate strategies and approaches currently adopted by companies to tackle this problem, the group evaluated industrial scenarios of networked organizations during companies merging.

SCENARIO 1: when Philips Corp. acquires companies, architects from both companies use software architecture as a brokering artefact to decide which IT infrastructure in either company needs adaptation to the other one. The "best" architecture between the two companies is used as a basis for integration/customization to support the networked organization. This process also requires to "decorate" the software architecture with business (e.g. business processes, business requirements, etc.) and organizational/social structure information (roles, responsibilities, locations, governance guidelines, etc.). Reference frameworks exist within Philips to drive this process but are currently an industrial secret. *NEED: research into networked organizations creation and governance.*

SCENARIO 2: when IBM acquires a company, they integrate the company's software products into IBM's product portfolio via a process known as "blue washing". This makes the creation of the networked organization much simpler and smoother, but has the potential to create social and organizational integration issues. *NEED: research into understanding and mitigation of social aspects for networked organizations.*

The group agrees that additional requirements come from the intense and increasing presence of service-based and cloud-based technologies. The problem of networked organizations in the cloud is still an open problem. Also, architecting for networked organizations is still an open problem. Both problems require the definition of architectural viewpoints and reference frameworks to specify and analyse software architectures from four perspectives:

1. Business
2. IT
3. Organization
4. Organizational Process

4.7 Binding time aspects of service platform customization

Customizing Service Platforms - Development time vs. Compile time vs. Runtime

Holger Eichelberger (Universität Hildesheim, DE)

Participants: Marco Aiello, Christian Inzinger, Jacek Serafinski, Holger Eichelberger

This group discussed temporal aspects of the customization of service platforms, in particular, the role of (self-)adaptation as a customization technique. This section summarizes the main findings of the group discussions in terms of (agreed) terminology, examples and scenarios for customizations at different points in time as well as research challenges.

4.7.1 Main Questions and Terminology

Several questions and terminology issues were discussed:

- What is the system being configured, i.e., what shall be subject to configuration? In this discussion, the group considered traditional systems (software product lines), service platforms, as well as service ecosystems.
- What is the semantics of time with respect to a customization? The group discussed two alternatives:
  - **Application time:** This denotes the point during the software production process when the customization is (actually) applied.
  - **Binding time:** The latest point in time when the decision for a customization must be made. This complies with the typical notion of binding time in Software Product Lines. In particular, a customization may be applicable at multiple binding times.

  During the discussion, the group adopted the binding time semantics.
- Which types of customizations are relevant and can those types be (partially) ordered? The group identified the following types:
  - **Configuration:** Simple as well as sophisticated mechanisms to configure a system, its settings or its code. Although pre-runtime configuration is frequently applied in traditional software product lines, configuration techniques may also be applied at runtime.
  - **Adaptation:** An external mechanism defines the configuration of the system at runtime. This does not imply that the adaptation mechanism itself can also be configured (through appropriate configuration approaches).
  - **Self-adaptation:** The system itself determines and applies appropriate configurations during its runtime. Thus, the adaptation mechanism itself may be subject to configurations within the same system.

  The group agreed that the given sequence expresses a (partial) order of increasing flexibility (and typically also complexity) of the underlying configuration mechansisms. While some work, e.g., [1, 2, 4, 7], make the distinction between adaptive and self-adaptive

systems, the group finally decided to focus only on two types, namely *Configuration* and *Self-adaptation*.
- Can a combination of customization type and the time aspect be used to classify existing systems and their applied customization mechanisms, in particular with respect to the timing aspects discussed by the group? The implied space can be considered as a coordinate-system with two axis, namely customization time and binding time. As discussed, the customization type depends on the (binding) time aspect, i.e., (self-)adaptation can only be applied during runtime of the system, e.g., for startup, initialization, or runtime customizations.

4.7.2 Examples / Scenarios

The group identified several examples and scenarios for applying customizations at different (binding) times. In particular, the group focused on service-based systems, service platforms, and service ecosystems. These examples were classified as shown below:
- **Configuration at development time:** Decisions in architecture, manual customizations in code, the build process, etc. The group considered this as a typical approach to the customization of systems and did not discuss specific scenarios.
- **Configuration at compile time:** This is most frequently applied in traditional software product lines. However, customization at compile time is not uncommon in service-oriented systems as well [3]. In particular, the group discussed the approach of the SAS-LeG project (Software As Service for the varying needs of Local eGovernments)[2], where (static) customization at compile time is applied to customize services for the needs of several municipalities in the Netherlands.
- **Configuration at deployment time:** In the INDENICA warehouse management case study [5], customizations are applied prior to or at deployment time so that the customized service platform and the services become active at deployment.
- **Configuration at startup time:** Software in this category reads configuration files and binds the customization with specific values at that point in time, i.e., during early runtime. Examples for traditional software systems are Apache `httpd`[3] or relational database management systems such as `MySQL`[4]. Further, all three service-based case studies in the INDENICA-project [5], namely the warehouse management system, the yard management system or the remote management system also rely on startup time mechanisms.
- **Configuration at runtime:** One particular example is the INDENICA warehouse management case study [5], where the user may determine and change the actual binding of customizations at runtime (although these capabilities are introduced at compile time).
- **Adaptation at runtime:** Here, in particular semi-automated or automated mechanisms such as adaptivity managers may change the actual customization of a system at runtime. This is for example applied in the SM4ALL project (Smart Homes for All)[5] on an embedded middleware for pervasive and immersive environments in order to enable a continuous adaptation of sensors, devices, services and appliances to user context and habits. Further, the virtual service platform researched in the INDENICA-project[6], which

[2] http://www.sas-leg.net/web/
[3] http://httpd.apache.org/
[4] http://www.mysql.com
[5] http://http://www.sm4all-project.eu/
[6] http://www.indenica.eu

integrates the INDENICA case studies mentioned above, contains an adaptation manager [6], which affects the configurations of the integrated customizable platforms, in particular the remote management system.

- **Customization at late runtime:** This class encompasses systems, which enable (open) customizations during runtime, which may even extend the system. Basically, this class includes multi-tenant systems or concepts relized by user-programmable systems such as `Cloud9`[7] or the `Heroku`[8] ecosystem.

The group assigned the examples and scenarios listed in this section to the two-dimensional coordinate system sketeched above. All examples and scenarios in this section are located below the bisecting line, which separates static from dynamic customizations. According to our findings, there is a clear relationship between binding time and flexibility, i.e., the later the binding time, the more flexibility is supported by the customizations, while extremely dynamic and flexible customizations are not applied at early binding times such as design or compile time.

4.7.3 Challenges

During the discussion we identified the following challenges:

- **Quantify the trade-offs among different (binding) times** in order to determine (relative) benefits, impacts or even risks. This quantification may happen in terms of metrics such as costs, revenue or downtime. Such tradeoffs enable the objective selection among available binding times for an individual customization opportunity, to support (risk) mitigation strategies or even to support temporal relationships across applied customizations, e.g., to trace failures to (combinations of) customizations applied at different points in time.
- **Support understandability of managing customizations at different points in time**: Are the actual capabilities sufficient for modeling and managing customizations which apply at different points in time? How can the different roles in the software development process be supported in understanding the effect (and the impact) of temporal customization aspects, in particular in the dynamic and open environment of service platforms?
- **Ensure semantically meaningful configurations when temporal aspects become customization alternatives.** This includes consistency issues (avoid selecting the wrong service, the wrong service interface or wrong data) or means to show the correctness of configurations.
- **Combine platform evolution and service platform lifecycle management with temporal configuration aspects.** How can upgrades of services and service platforms with temporal customizations be upgraded? How can staged upgrades be supported? How can integrity and consistency of service bindings and data be guaranteed?
- **Analyze the mutual influence of temporal customization aspects on the openness of service platforms.** By construction, service platforms support open-world scenarios, e.g., services can be discovered and bound at runtime. Do temporal aspects introduce another dimension of openness? How can openness be considered in the challenges state above, for example, how do customizations interrelate with openness and

[7] https://c9.io
[8] https://www.heroku.com

extensibility, e.g., in terms of customizable extension bundles with own development lifecycle?

- **Analyze the impacts of temporal aspects in customizing multi-tenant environments**, e.g., with respect to tenant-specific isolation mechanisms (regarding resource usage, data and tenant-specific functions) or mapping of functionality or resources to physical infrastructures. Shall (temporal) customization be available for tenant-specific extensions (e.g., as part of a development introduce new temporal customization aspects such as a replication of binding times (a "development" time as part of runtime) or can this be considered as views (the system is still at runtime while the tenant has its own time scale).

References

1 N. Abbas. Towards autonomic software product lines. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, pages 44:1–44:8, New York, NY, USA, 2011. ACM.

2 N. Abbas, J. Andersson, and D. Weyns. Knowledge evolution in autonomic software product lines. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC'11, pages 36:1–36:8, New York, NY, USA, 2011. ACM.

3 H. Eichelberger, C. Kröher, and K. Schmid. Variability in Service-Oriented Systems: An Analysis of Existing Approaches. In *Conf. on Service-Oriented Computing (ICSOC'12)*, pages 516–524, 2012.

4 D. Garlan, B. Schmerl, and S.-W. Cheng. Software architecture-based self-adaptation. In Y. Zhang, L. T. Yang, and M. K. Denko, editors, *Autonomic Computing and Networking*, pages 31–55. Springer US, 2009.

5 INDENICA project consortium. Description of Feasible Case Studies. Technical Report Deliverable D5.1, 2011. http://www.indenica.eu [validated: April 2013].

6 INDENICA project consortium. Report Describing a Framework for Deployment, Monitoring & Controlling of Virtual Service Platforms. Technical Report Deliverable D4.1, 2012. http://www.indenica.eu [validated: April 2013].

7 M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42, May 2009.

5 Open Problems

Open problems were described throughout the previous sections, in particular, in the working group summaries.

Participants

- Marco Aiello
University of Groningen, NL
- Luciano Baresi
Polytechnic Univ. of Milan, IT
- Karina Barreto Villela
Fraunhofer IESE – Kaiserslautern, DE
- Deepak Dhungana
Siemens AG – Wien, AT
- Peter Dolog
Aalborg University, DK
- Schahram Dustdar
TU Wien, AT
- Holger Eichelberger
Universität Hildesheim, DE
- Gregor Engels
Universität Paderborn, DE
- Sam Guinea
Politecnico di Milano, IT
- Waldemar Hummer
TU Wien, AT
- Christian Inzinger
TU Wien, AT
- Patricia Lago
Free Univ. of Amsterdam, NL
- Grace A. Lewis
Carnegie Mellon University, US
- Georg Leyh
Siemens AG – Erlangen, DE
- Tiziana Margaria
Universität Potsdam, DE
- Nenad Medvidovic
USC – Los Angeles, US
- Nanjangud C. Narendra
IBM India – Bangalore, IN
- Leonardo Passos
University of Waterloo, CA
- Cesare Pautasso
University of Lugano, CH
- Manuel Resinas Arias de Reyna
University of Sevilla, ES
- Florian Rosenberg
IBM TJ Watson Res. Center – Yorktown Heights, US
- Antonio Ruiz Cortés
University of Sevilla, ES
- Andreas Rummler
SAP Research Center – Dresden, DE
- Klaus Schmid
Universität Hildesheim, DE
- Jacek Serafinski
NextDayLab Sp. z o.o. – Poznan, PL
- Damian Andrew Tamburri
VU – Amsterdam, NL
- Frank van der Linden
Philips Medical Systems – Best, NL
- Wenjun Wu
Beihang University – Beijing, CN
- Uwe Zdun
Universität Wien, AT