Dagstuhl Seminar 09021: Software Service Engineering Executive Summary

Willem-Jan van den Heuvel[1], Olaf Zimmermann[2], Frank Leymann[3], Tony Shan[4]

[1] European Research Institute on Services Science (ERISS), Tilburg University, Warandelaan 2, 5000 LE, Tilburg, The Netherlands, wjheuvel@uvt.nl

[2] IBM Research GmbH, Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland olz@zurich.ibm.com

[3] Universität Stuttgart, Institute of Architecture of Application Systems, Universitätsstraße 38, 70569 Stuttgart, Germany frank.leymann@iaas.uni-stuttgart.de

[4] SOACP (www.soacp.com), 708-1155 West Pender Vancouver, BC, Canada V6E 2P4, tonycshan@gmail.com

Abstract. Service-Oriented Architecture (SOA) constitutes an important, standards-based and technology-independent distributed computing paradigm and architectural style for discovering, binding, assembling, and publishing loosely-coupled and network-available software services. With SOA-enabled applications operating in highly complex, distributed, and heterogeneous execution environments, SOA practitioners encounter the limits of traditional software engineering. In this Dagstuhl seminar, we have discussed and explored the fundamental tenets underpinning the development and maintenance of SOA systems. As a result of our discussions, we position software service engineering as an evolving and converging discipline that embraces the open world assumption. Software service engineering entails a departure from traditional software engineering disciplines such as component-based development, supplementing them with techniques and patterns tailored to service enablement, composition, and management.

Keywords: Service engineering, software engineering, service-oriented computing, service-oriented analysis and design, SOA, systems engineering, Web engineering.

1 Seminar Topics and Objectives

Service-oriented architecture (SOA) [11] as an architectural style based on common principles and patterns such as Business Process Choreography and Enterprise Service Bus (ESB) allows service engineers to effectively (re-)organize and (re-) deploy executable business processes, functional components, and information assets as business-aligned and loosely-coupled software services. SOA is unique in that it aims at unifying various related, yet up to now largely isolated domains such as

Dagstuhl Seminar Proceedings 09021
Software Service Engineering
http://drops.dagstuhl.de/opus/volltexte/2009/2041

business process management, distributed computing, enterprise application integration, software architecture, and systems management.

Given the loosely-coupled, heterogeneous, and dispersed nature of SOA, several of the key assumptions underlying traditional approaches to software engineering are being challenged; consequently, conventional software engineering methods and tools have to be carefully reevaluated and possibly extended to be applicable to analysis, design, construction, and delivery of software services. Due to the continuing evolution of SOA, SOA research so far has been mostly focused on certain parts of the service lifecycle, such as runtime service infrastructure and middleware. There is a lack of comprehensive methods and tools consistently supporting all phases of software service engineering ranging from analysis to implementation and evolution. Such methods and tools must be grounded both in scientific foundations and in industrial best practices.

It was the overall goal of this seminar to assess existing methods, techniques, heuristics, and practices from related fields such as requirements engineering, software engineering, Object-Oriented Analysis and Design (OOAD), Component-Based Development (CBD), and method engineering to harness SOA methods and tools and to define a road map for the creation of a unified software service engineering method. More precisely, the first objective of the workshop was *to understand assumptions and impact of emerging runtime platform models on the engineering process*, e.g., SOA principles and patterns such as loose coupling and programming without a call stack, ESB and service composition, Software as a Service (SaaS) and cloud computing, Web 2.0 and mashups, as well as mass programming.

Based on the results of this analysis activity, the second objective of the workshop was *to define distinguishing characteristics of Software Service Engineering (SSE) and to assess the state of the art in SOA and service design methods.*

The third and last goal was to *identify focus areas for future work and a roadmap for SSE*. In particular, the following three questions were addressed: Are new methods and tools required? How can the existing body of knowledge in software engineering and SOA design be extended? Is method unification a la Unified Modeling Language (UML) [16] and Unified Process (UP) [12] desirable and feasible?

2 Seminar Organization

Participating communities. With this seminar we brought together researchers and practitioners from various industrial domains and research areas that work in the emerging field of software service engineering. In particular, we established linkages and enduring collaborations between the following three communities that operated in isolation before:

1. Requirements and software engineering including patterns.
2. SOA middleware and platform standards.
3. Industrial adopters of SOA.

41 participants from 10 countries attended the seminar; industry participation was in the range of 40% to 60% (depending on how industrial research labs are counted). Areas of interest and expertise varied from business process modeling to SOA design principles, patterns, and platform, but also method engineering, software architecture, testing, legacy system analysis, semantic Web, and software product lines.

Themes. To realize the seminar's objectives, and to streamline presentations and discussions, the seminar was organized around the following themes:

1. What are the novelties in Software Service Engineering (SSE)?
2. Top-down SSE starting from business architecture and mapping to Information Technology (IT) architecture.
3. Bottom-up SSE service composition and service enablement of existing (legacy) applications.
4. Recurring design issues in meet-in-the-middle service realization and patterns for them.

Agenda. We organized this Dagstuhl seminar into three general and four topic-specific sessions that used various formats. The two and a half seminar days comprised a series of interactive presentations as well as plenary discussions and breakout groups. They were accompanied by an open space session [9] and two panels held in the evenings. The panels focused on the impact of cloud computing on SSE and on SOA patterns, respectively.

Day 1 was devoted to introduce the topic and to establish a joint understanding of SSE aspects such as process, notation, platform, principles, patterns, and tools. Two general sessions featured four widely recognized speakers from the SOA platform, pattern, and practitioner communities. The open space session gave all participants the opportunity to propose additional topics to be discussed or presented in breakouts. Seven sessions were held; topics included "how to model services in UML", "from enterprise architecture to SOA", "the role of domain-specific languages in SSE", and "RESTful service composition".

Day 2 was devoted to finding answers to the questions we had posted for the seminar objectives (see Section 1). One session per topic was held (see beginning of this section): each session comprised two presentations and a 30 minute discussion slot. Session chairs and scribes captured the results in the seminar wiki.

Day 3 had the objective to recapitulate and reflect upon the sessions of the previous two days, working towards a research roadmap or manifesto. To do so, we first discussed the format in the plenum and jointly decided to divide into two moderated breakout groups that both investigated defining characteristics of SSE. The results of the breakout sessions were presented in the plenum (see next section). After that, a particular form of crowd and voting game was facilitated to give all participants the opportunity to propose their number one challenge for SSE and come up with a ranking that was not dictated by any of the organizers, session facilitators, or other opinion leaders. This voting game turned out to be highly efficient; the technique scaled well and ensures that less vocal participants could provide input to the decision making. In the following Section 3 we present the results.

3 Seminar Results

We structure this section of the seminar report into initial analysis, conclusions regarding the four SSE topics from Section 2, and a synthesis of defining characteristics and related challenges in industry and academia.

3.1 Initial Analysis (Day 1)

SSE must avoid the pitfalls of an uncontrollable maze of services and provide a solid foundation for service development. It must allow for professional, highly distributed peer development and service management. The following general issues emerged:

1. Support for opportunistic SSE projects (a.k.a. situational development).
2. Bridge a modeling chasm: design/development and delivery/execution.
3. Bridge another analysis and modeling chasm to find a top-level *model, analysis, develop, deliver* SSE process loop that integrates a *{service, policy, data, message, user}* artifact set and supports the iterative and incremental decomposition and refinement of such artifacts.
4. Existing programming models allow constructing service-oriented solutions, but they do so in a sub-optimal way both from a build time developer experience and runtime quality attribute point of view.

Existing assets from the industry include Service Lifecycle Process [18], Service-Oriented Modeling Framework [3], and Mainstream SOA Methodology (MSOAM) [6]. Service-Oriented Modeling and Architecture (SOMA) [1] is the IBM methodology for service-oriented analysis and design. Vendors such as SAP and BEA, and firms providing analysis and design consulting services, e.g., Cap Gemini, sd&m [5], and Everwhere-CBDI [8] provide additional methodologies. Some of the existing assets from the industry are proprietary and for company internal use, or are only partially published. In academia, we have witnessed proposals such as Service Development Lifecycle Methodology (SDLM) [15], Service Centric System Engineering (SECSE) Methodology [17], and Architectural Framework for Service Definition and Realization (SOAF) [7]. Some of the assets from academia have not been proven in practice yet.

Open space session summaries. The open space sessions were summarized in the seminar wiki; the main results served as input to the general session on day 3 which produced the SSE characteristics and challenges we will present in Section 3.3.

3.2 Presentations and Discussions (Day 2)

In the following, we will summarize the key conclusions from the presentations and discussions revolving around the seminar's four themes that were introduced in

Section 2. The presentations themselves are available for download from the website of the seminar (http://www.dagstuhl.de/09021).

1. What is new in Software Service Engineering (SSE), including programming without a call stack? While the foundations (i.e., models, techniques, and concepts) of SSE are available and relatively mature, they unfortunately fall short in designing service-based systems that operate in open, dynamic, and uncontrolled environments. Therefore, SSE involves combining various programming models and development paradigms, including event-driven and asynchronous programming, declarative programming, process modeling, and protocol design.

2. Top-down SSE starting from business architecture and mapping to Information Technology (IT) architecture. Currently, many uncorrelated approaches and tools are available for developing SOAs on the one hand and for facilitating business (process) modeling on the other hand. It remains a challenge, however, how these methods and supporting toolsets can be seamlessly integrated to formulate a holistic approach, so that business-aligned software services can be identified, specified, realized, tested, deployed, managed, and evolved in a consistent and standardized fashion.

3. Bottom-up SSE service composition and service enablement of existing (legacy) applications. We are quickly moving from the current Internet of services to the ubiquitous Internet of services and things, which combines software services with mobile devices, sensor networks, and social networks. This leads towards the concept of *services everywhere*. In this novel service engineering concept, software services residing on the Internet are fuelled by various very heterogeneous building blocks. These building blocks include mashups, complex cloud computing stacks, and already existing external services. Clearly, the service everywhere concept is strongly influenced by social computing, service-oriented computing, and cloud computing. Already existing enterprise systems will continue to provide important sources for services; there is a growing need for integrated approaches that cater for redeployment of legacy resources as services on clouds.

4. Architectural decision models and tools for meet-in-the-middle service realization. In this session, we investigated how service design knowledge can be made reusable and how to identify, make, and enforce architectural decisions during SOA design. Pattern languages can go a long way in supporting SSE practitioners; however, they do not provide everything that is required to make tacit software service design rationale explicit. Architectural decision models complement pattern languages with domain-specific guidance and technology and asset-level refinements.

3.3 Seminar Result Synthesis (Day 3)

We distilled the observations from day 1 and the conclusions from day 2 into a set of defining SSE tenets, which fall in three complementary dimensions: *architectural (platform), process, and engineering* (see Figure 1).

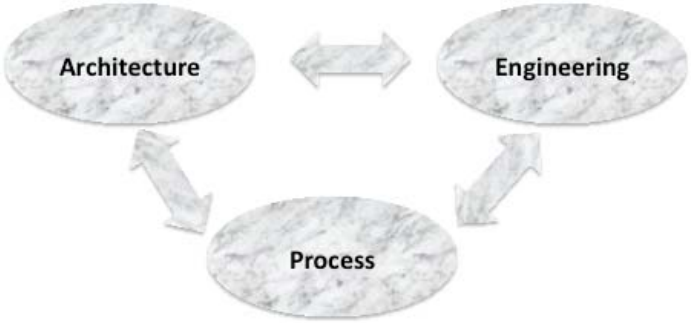

Fig. 1. Dimensions of SSE Tenets

The **architectural (platform) tenets** were compiled as follows:[1]

- *Service* is the key design and runtime concept; a service is described by its *contract*.
- The services described in a contract are provided by *endpoints* (providers) and invoked by service requesters.
- *Composition* of services is facilitated by the service contract.
- Services communicate via *document* exchange using *messaging* technology.
- The service communication leverages *protocols*, which may support request coordination and support for conversations.
- *Service virtualization* supports location transparency.

From a process perspective, we identified the following **process tenets**:

- *Scoping (application boundaries)/context*. Services should be designed in such a way that they routinely support business processes. This implies a scoping of processes so that their supporting software services are logically cohesive and loosely coupled, minimizing message, protocol, and context dependencies.
- *Lifecycle, ownership, and versioning*. The objective of SOA is to manage the lifecycle of a service starting from business goals over service definition, through deployment, execution, measurement, analysis, change, and redeployment. Specifically, during their life services are subject to two broad classes of changes: low-impact changes versus intrusive changes. Intrusive changes include operational behavior changes and policy-induced changes, while low-impact changes demand a comprehensive service versioning strategy that may cater for forward and backward compatibility. A key issue regarding version management entails ownership of service data, logic, and transactions, especially in the context of processes that cross organizational boundaries.

[1] The tenets can be expressed as principles or patterns (or both). At present, there is no industry consensus on these principles and patterns; each book author follows his/her own approach.

- *Reuse and variability*. To cater for reuse in various process contexts, services should be designed as differentiated services that allow for multiple levels of service, depending on service request(er). Functional variability may also be built in by parameterization, delegation, or specialization/generalization.
- *Governance and roles.* Successful implementation and management of service-enabled processes is directly dependent on a strict service governance framework that clearly defines chains of responsibility, measurements to gauge efficacy, and controls to check on compliance.

Finally, the following seven clusters of **engineering tenets** were established:

1. Technical federation. SSE has to cater for service-enabled software applications that are highly distributed in nature with many asynchronous interactions between services. In addition, SSE has to deal with services that may be deployed on various runtime platforms, including mobile devices, computing clouds, and legacy systems, and have been developed in various programming paradigms – including, but not limited to, OOAD and CBD.

2. Dynamism. A key tenet of SSE is dynamism regarding both the services that are aggregated into dynamic service compositions via late binding – possibly into agile service networks – as well as the highly volatile context in which they operate. Firstly, dynamism implies that SSE methods, techniques, and tools have to deal with emergent properties and behavior of complex service networks, which may in fact be comprised of thousands of independent – yet cooperating – services. In fact, emergent behaviors pertain to system-level issues such as performance and security as well as to business-level issues including profitability, return-on-investment, and indices of value-creation. This signifies that software applications that have been designed in accordance with SSE typically exhibit unpredictable, non-linear and non-deterministic behavior. Dynamism puts requirements on virtually all layers of the typical SOA stack, ranging from the network layer (often SOAP messages transmitted over synchronous HTTP or asynchronous messaging protocols) to the composition layer (e.g., BPEL). Late binding and loose coupling constitute two key principles for increasing the adaptability of service applications and for accommodating dynamic (re-)composition as well as (re-)configuration of services in a network. In addition, SSE has to accommodate various styles of composition, fostering user-friendly enterprise service mash-ups as well as heavy-weight compositions of industry-strength enterprise applications by service development professionals.

3. Organizational federation. SSE should be shaped around the doctrine stating that development and maintenance (operations) be typically achieved in highly distributed organizational environments, involving multiple departments, units, enterprises, and governmental organizations. Typically, development and maintenance of applications will be a collaborative effort, implying that in fact design, coding, deployment etc. will occur in networks of collaborative service clients and providers. Organizational federation requires sound distributed governance policies and mechanisms, accommodating individual needs of various stakeholders and constraints stemming from organization-specific policies or governmental rules and legislations. Organizational federation may adopt a range of coordination mechanisms, ranging

from a classical central control system to a decentralized control approach, relying on mechanisms such as service markets and contracts.

4. Boundaries. Services developed with SSE methods or tools have to be endowed with clear and explicit boundaries. In particular, SSE has to respect service contracts that capture goals and constraints (pre- and post-conditions and invariants), capitalizing Bertrand Meyer's classical design-by-contract principle [13]. An intrinsic part of the service contract entails the service interface that clearly specifies the messages a service understands and the service endpoints that are available. Enriching the service interfaces with additional semantic information such as scenarios or behaviors allows a more robust and stable service composition. In addition, given the highly distributed and volatile nature of service applications, service contracts have to be aligned with service level agreements between service clients and providers.

5. Heterogeneity. Any SSE concept, method, and tool have to embrace heterogeneity of the service application and the context in which it operates. Just like dynamism, heterogeneity impacts all phases of the service development lifecycle, posing restrictions on how software service systems can be designed, developed, deployed, and evolved over time. Note that in contrast to before, no assumptions can be made about the system's programming, execution, and management context before, during and after deployment.

6. Business-IT alignment. SSE embraces a new style of development assuming that software service applications can be systemically and routinely (re-)mapped to the business processes they realize, and vice versa. This in fact points towards the need for unification of concepts, models, methods, and techniques from Business Process Management (BPM) to ensure that these applications do not only meet system-level Quality of Service (QoS) criteria, but also conform to given process-level business performance indicators.

7. Holistic approach. A key distinguishing "meta" characteristic of SSE refers to its holistic nature. More than ever before, SSE demands an interdisciplinary approach towards the analysis and rationalization of business processes, design of supporting software service systems, their realization, deployment, provisioning, monitoring, and adaptation. This implies that SSE concepts, models, methods, and tools be integrated and unified, adhering to open standards and offering integrated support for multiple stakeholders.

Research challenges. To derive research and industry development challenges from the defining tenets and characteristics, a crowd-sourcing and -scoring game was conducted. First, the participants were asked to briefly answer the question:

What is the most important challenge of SSE?

32 participants submitted an answer. Next, these unedited answers served as input to a scoring game without any upfront discussion clarification; duplicates were not eliminated. Pairs of answers were scored against each other in four iterations (the pairs were built randomly; the facilitator of the game only was responsible for the time management). The maximum score per iteration was five points. Hence, the

highest possible score was 20 points. The result of this sourcing and scoring game is the following consolidated list of answers, ordered by points scored:

1. Address the 'open-world' assumption: unforeseen clients, execution context, usage (16 points)
2. Bridging a modeling chasm: design/develop and delivery/execution (15)
3. 'Open world assumption': uncertainty (15)
4. IT-business alignment, adaptability (15)
5. Alignment of technical and business engineering for services (14)
6. New models and abstractions to represent and handle SOA dynamics (14)
7. To develop software without knowing in which context it is used (14)
8. Programming models and runtime integration (14)
9. Service resilience, system level (robustness) (13)
10. The mapping from requirements to services fulfilling them (13)
11. How to architect SOA with respect to the heterogeneous nature; dealing with heterogeneity (13)
12. Making the leap from business service to the right technical service design (11)
13. Alignment of business and technical SSE level (12)
14. Composability (11)
15. Testing (11)

Not surprisingly, many of these research challenges are closely related to the SSE tenets. Table 1 loosely correlates the 15 research challenges to the engineering tenets. Note that engineering tenet 7 (holistic approach) pertains to all research challenges and is therefore not included in Table 1.

Table 1. Correlation of SSE Tenets and Challenges

SSE Tenet	Description	Challenge ID
1	Technical federation	7, 8, 9, 14, 15
2	Dynamism (virtualization)	1, 3, 6, 15
3	Organizational federation	1, 3, 7
4	Explicit boundaries (contracts)	10, 12
5	Heterogeneity	11
6	Business-IT alignment	2, 4, 5, 13, 15

From this informal cross-correlation we may carefully draw first conclusions. It should be noted that the level of granularity of the research varies; some challenges are very generic in nature – including challenge 1 and 3 – while other challenges address specific problems such as service composability and testing.

The number of challenges correlated to an SSE tenet indicates how the participants of the game perceive the tenet. The same holds true for the score of the challenge, which is expressed by the challenge ID: a small number indicates high importance.

The research challenges relating to tenet "technical federation" include the design of service-based applications without any knowledge about the context in which these applications will be executed. This research challenge is critical in open and agile service networks, with many interactions between service participants, which are not known at design time. In addition, there is a need for novel approaches to integrate programming models and platforms while processes in service networks are executed.

The high level of change in service networks also demands services to be robust and reliable. Challenge 8 points out that the traditional boundary between application development and integration on the one hand and application maintenance and change management on the other hand becomes blurred in SSE. In response, continuous integration, a term from agile development, may be projected into the operations and maintenance phase of the service development lifecycle to support *continuous evolution*. Backward and forward compatibility issues have to be addressed here.

The 'open world assumption' makes the current architecting methods obsolete to a large extent, as they are largely based upon a predefined organizational and technical context. Some flexibility is taken into account, but not nearly as much as required when designing under the 'open world assumption'. Furthermore, the traditional architecture-business cycle that expresses the bidirectional influence between the technical system and the business organization cannot be managed using traditional architecting methods in SSE because of the high dynamism and heterogeneity put forward by the SOA style. Therefore the architecting dimension of SSE needs to be thoroughly re-considered, possibly leading to a new architecting paradigm. Architecture knowledge management with its focus on architectural decisions and their rationale is an emerging sub-discipline of software architecture that we expect to contribute solutions to this new architecting paradigm [20].

Because of the 'open world assumption' and the dynamisms of service-based applications, traditional test methods for system development and deployment are no longer sufficient: As not all usage contexts and configurations can be predetermined in pre-deployment tests setups, tests have to be extended into the operation and maintenance of these applications. Contract-oriented build-in tests, active online tests, and runtime auditors and supervisors are first developments in this direction.

We investigated the possibility to come up with an "SSE manifesto", but concluded that such an undertaking would be premature and bound to fail at this stage. Initial ideas for statements in such manifesto were to value sustainable benefits over tactical gains, to prioritize adaptability over reusability, to prefer business-driven change to technology-driven change, to favor rapid team formation over formal structures, and to favor meet-in-the-middle service design techniques over dogmatic unidirectionality.

4 Conclusion and Outlook

SOA-enabled applications can be developed and evolved by applying aging software engineering paradigms, notably CBD and OO; however, the key advantages of SOA cannot be fully exploited when doing so. The main reason for this is that conventional software engineering paradigms typically adopt the closed world assumption, hypothesizing that applications have clear boundaries, and will be executed in fully controlled, relatively homogeneous, predictable and stable execution environments. This thesis is backed up by conclusions drawn from a decade-to-decade analysis of software engineering by Barry Boehm [3][4].

Instead, we claim that for SOA to be applied successfully, SSE has to embrace the 'open-world assumption', in which software services are composed in agile and

highly fluid service networks – that are in fact systems of software-intensive systems – operating in highly complex, distributed, and heterogeneous execution environments. In addition, the service networks that are designed based on this assumption have to be continuously (re-)aligned with business processes, and vice versa. Adoption of the 'open-world assumption' is reflected in the three types of SSE tenets: architecture, process, and engineering.

Based on the research reported, we came up with an initial definition of SSE as:

Software service engineering entails the science and application of concepts, models, methods, and tools to define, design, develop/source, integrate, test, deploy, provision, operate, and evolve business-aligned and SOA-enabled software systems in a disciplined and routinely manner.

Clearly, SSE will benefit from timeless generic principles and lessons learned from its elderly parent software engineering; however, we herein argue that aging computing model specific principles and practices, e.g., distributed component technology, are in clear need for revision given the specific nature of SOA.

In our view, SSE will be based on standards and will be frequently realized with Web services. Specifications such as SOAP, WSDL, BPEL, WS-Policy, and WS-Agreement already constitute the first step to realize the technical aspects in some of the SSE tenets, including engineering tenets 1, 2, 4, and 5. Other architectural styles and technology paradigms can also be used to realize software services. However, further research is required to more effectively satisfy the open-world assumption. This has also been reflected in the outcome of the brainstorm on the key open research challenges.

The results of this seminar are core results in nature. During the seminar it became clear that the discipline of software service engineering is still in its embryonic phase, and further work is required in several directions. Firstly, the list of tenets has to be further explored, validated, and potentially refined or expanded. The presented list is derived from a literature survey, as well as expertise and experience from real-world SOA projects and discussions with leading industry experts and renowned researchers in the field of software engineering, software patterns and SOA; however, more case studies have to be analyzed critically to further validate this initial list.

As a follow-up activity, we have published the results of this seminar in an ICSE workshop paper [19]. The workshop paper extends the discussion in this executive summary and provides an example which illustrates the difference between SSE/SOA and traditional software engineering disciplines.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

[1] Arsanjani, A.: Service-Oriented Modeling and Architecture (SOMA), IBM developer-Works 2004, http://www.ibm.com/developerworks/webservices/library/ws-soa-design

[2] Bass, L., Clements P., Kazman R., Software Architecture in Practice, 2nd Edition, Addison Wesley, 2003.

[3] Bell, M., Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture, Wiley, 2008.

[4] Boehm B., A View of 20th and 21st Century Software Engineering. In: Proceedings of the 28th international Conference on Software Engineering ICSE, 12-29, ACM Press, 2006.

[5] Engels G., Hess A., Humm B., Juwig O., Lohmann M., Richter J. P., Voß M., Willkomm J., A Method for Engineering a True Service-Oriented Architecture. In: Proc. of ICEIS 2008.

[6] Erl T., Service-Oriented Architecture: Concepts, Technology & Design. Prentice Hall, 2005.

[7] Erradi A., Anand S., Kulkarni N., SOAF: An Architectural Framework for Service Definition and Realization. Proceedings of SCC'06, IEEE Computer Society, 2006, pp 151-158.

[8] Everware-CBDI Inc, CBDI Service Architecture & Engineering: A Framework and Methodology for Service-Oriented Architecture (SOA). CBDI Report, 2006.

[9] Fowler M., Open Space Sessions, http://www.martinfowler.com/bliki/OpenSpace.html

[10] Hohpe G., SOA Patterns – New Insights or Recycled Knowledge? http://www.eaipatterns.com/docs/SoaPatterns.pdf

[11] Krafzig D., Banke K., Slama D., Enterprise SOA, Prentice Hall, 2005.

[12] Kruchten P., The Rational Unified Process: An Introduction. Addison-Wesley, 2003.

[13] Meyer B., Object-Oriented Software Construction, 2nd Edition. Prentice Hall, 2000.

[14] Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. 2007. Service-Oriented Computing: State of the Art and Research Challenges. Computer 40, 11 (Nov. 2007), pp. 38-45.

[15] Papazoglou M., van den Heuvel W. J., Service-Oriented Design and Development Method, International Journal of Web Engineering and Technology (IJWET), Volume 2 No 4. Inderscience Enterprises, 2006, pp. 412-44.

[16] Rumbaugh, J., Jacobson, I., Booch, G., The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

[17] SECSE, Service-Centric System Engineering. http://secse.eng.it/pls/secse/secse.home, 2004-2008.

[18] SOA Practitioners' Guide Part 3: Introduction to Service Lifecyle. http://www.healthmgttech.com/editorial_whitepages/SOAPGPart3.pdf. 2006

[19] van den Heuvel W. J., Zimmermann O., Leymann F., Lago P., Schieferdecker I., Zdun U., Avgeriou P., Software Service Engineering: Tenets and Challenges. In: Proceedings of ICSE 2009 Workshop - Principles of Engineering Service Oriented Systems (PESOS), IEEE Computer Society, May 2009.

[20] Zimmermann O., Koehler J., Leymann F., Polley R., Schuster N., Managing Architectural Decision Models with Dependency Relations, Integrity Constraints, and Production Rules. Journal of Software and Services, Special Edition on Architectural Decisions, Elsevier, 2009.