Software Service Engineering: Tenets and Challenges

Willem-Jan van den Heuvel, Olaf Zimmermann[2], Frank Leymann[3], Patricia Lago[4], Ina Schieferdecker[5], Uwe Zdun[6], and Paris Avgeriou[7]

[1]Tilburg University, [2]IBM Zurich Research Lab, [3]Stuttgart University, [4]VU University Amsterdam, [5]Fraunhofer Institute, [6]Vienna University of Technology, [7]University of Groningen

Abstract

Service-Oriented Architecture (SOA) constitutes an important, standards-based and technology-independent distributed enterprise-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 engineers are confined by the limits of traditional software engineering. In this article, we scrutinize the fundamental tenets underpinning the development and maintenance of SOA systems. In particular, we introduce software service engineering as an emerging discipline that entails a departure from traditional software engineering disciplines such as component-based development, embracing the 'open world assumption'. Lastly, this article surveys research challenges.

1. Introduction

Service Oriented Architecture (SOA) is rapidly emerging as the premier distributed computing paradigm for developing, integrating, and evolving enterprise applications [8]. Many organizations are now in their early use of SOA, and assume that to engineer services they can simply apply principles and techniques from pre-existing software engineering paradigms such as Object Orientation (OO, [5]) or Component-Based Development (CBD, [2]), or the traditional architecting approaches (views like components and connectors) which are too generic for SOA. However, while SOA-enabled applications are operating in highly complex, distributed, unpredictable and heterogeneous execution environments, SOA engineers quickly encounter the limits of such traditional software engineering paradigms. Moreover, and probably more problematic, SOA confines itself to prescribing a rather rudimentary reference model for application development and deployment, defining basic roles such as service consumer (requester), service provider and service broker (repository). It is left up to the discretion of the engineers how to engineer software service applications within this model.

Our ultimate objective is to scrutinize the viability of existing engineering paradigms for developing and maintaining software service-based applications, including CBD and OO, and to explore their shortcomings. In particular, in this paper we investigate and further explore the distinguishing characteristics of a new engineering discipline for SOA-enabled applications, which we call *Software Service Engineering (SSE)*. We define the key SSE tenets. Lastly, this article aims at landscaping the key challenges for establishing SSE as a discipline.

The research that is presented herein has been conducted adopting a hybrid research approach, combining our background from literature surveys, case studies and best practices, and brainstorming sessions with key representatives of several communities – including key researchers and practitioners from the domain of software engineering, software patterns, SOA, and method engineering.

2. Background: SOA Principles and Patterns and SOA Design

Service-oriented architecture (SOA) as an architectural style based on common principles and patterns (such as Business Process Orchestration/Choreography and Enterprise Service Bus, [6]) allows service engineers to effectively (re)organize and (re)deploy business processes, functional components, and information assets as business-aligned, loosely-coupled and autonomous software services. SOA is unique in that it aims at combining various related, yet up to now largely isolated disciplines such as business process management, distributed computing, enterprise application integration, software architecture, and systems management.

Software architects on SOA projects are responsible for defining the architecturally-significant requirements (ASRs) during architectural analysis, such as use case and business process models, but also software quality attributes. Subsequently they propose design decisions to satisfy the ASRs during architectural synthesis resulting in designing the different aspects of the system by choosing and populating a number of architectural views iteratively and incrementally. The architects must ensure that the decisions made during synthesis are an optimal match against the ASRs defined through analysis, during the activity of architectural evaluation [3]. Finally, architects lead project teams via coaching and review activities, and manage the relationships with external stakeholders on the technical level. All these activities and interactions influence each other.

At the early elaboration stages, the conceptual architectures of SOA-based systems are straightforward to define: they are variations of logically layered two- or three-tier client-server architectures, which use message passing patterns to let *service consumers* and *service providers* communicate as well as workflow patterns to compose atomic services. A *service registry* serves as design time or runtime directory of service providers available to respond to service consumer requests.

During SOA design, though, architects are also concerned with the design, installation, and configuration of middleware components such as *Enterprise Service Buses* (responsible for service request routing, adaptation, and mediation), *business process orchestration engines* (performing service composition), and service registries (performing service provider lookup). Individual service consumers and providers of various types are designed, developed, and then deployed into such *SOA infrastructures*. The design solutions for these issues and numerous others have been successfully codified into design and architecture patterns by the software patterns community. However the use of such patterns in the daily practice of SOA architects and designers has not been particularly successful to date.

3. SSE Tenets

Software service engineers cannot be expected to embark on large-scale and complex SOA-development and maintenance projects without relying on sound principles and tenets underpinning the methods, techniques, and tools offered by SSE. Without sound SSE tenets, we cannot guarantee that the usage of SOA-compliant methods and tools results in software applications that meet the basic SOA criteria ensuring that services are *loosely coupled*, *self-contained*, and have a *clean interface* that is geared towards (re-) composition.

During a Schloss Dagstuhl seminar[1] organized in January 2009, we have gathered the key distinguishing SSE tenets. This was achieved by organizing two half-day (brainstorm) sessions in two groups of approximately 25 participants. During the first session candidate SSE tenets were identified and analyzed, while the second session was a plenary session during which the two proposed lists were correlated, integrated and consolidated. Note that due to reasons of space, we have not included transcripts of these discussions. They may be found in [1]. The discussions were kick-started by offering a list of potential tenets that were distilled on the basis of a literature survey that analyzed fundamental tenets underpinning OO and CBD (including seminal works such as [9], [10], [11], and [12]), but also input from other fields such as telecommunication services [13], networking [14] and testing [15].

The following list of seven clusters of SSE tenets were identified and defined:

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 run-time platforms, including mobile devices, computing clouds, and legacy systems, and have been developed in various programming paradigms – including, but not limited to, OO and CBD.
2. *Dynamism*. A key tenet of SSE is dynamism regarding both the services that are aggregated into dynamic service compositions – also referred

[1]www.dagstuhl.de/de/programm/kalender/semhp/?semnr=09021

to as 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 both to technical issues such as performance and security, as well as business 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) to the composition layer (e.g. by BPEL and BPEL^{light}). Late binding and loose coupling constitute two key principles for increasing the adaptability of service applications, accommodating dynamic (re-)composition and (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, 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 [16]. An intrinsic part of the service contract entails the service interface that clearly specifies the messages a service understands and the service end-points 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, there is a clear need to align service contracts with *Service Level Agreements* between service clients and providers. Finally SSE can use the sound principles of *built-in testing* allowing for services to contain their own test specification and enabling their run-time verification [18].

5. *Heterogeneity*. Any SSE concept, method or tool has 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 current practice, no assumptions can be made about the system's programming, execution, and management context before, during or after deployment.

6. *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 perform 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 and monitoring and adaptation. This implies that SSE concepts, models, methods are integrated and tools are interoperable, adhering to open standards and offering integrated support for several stakeholders.

4. Key SSE Research Challenges

To derive research and industry development challenges from the defining tenets and characteristics, a crowd-sourcing game has been conducted in Dagstuhl. The participants were asked to write a short answer to the following question: “What is the most important challenge of SSE?” 32 participants submitted an answer; the highest possible score was 20 points (result of four iterations of evaluating the answers, each round yielding a maximum score of 5). Result of this voting game was the following consolidated list of answers, ordered by total scored points:

1. Address the “open-world” assumption: unforeseen clients, execution context, usage (16 points)
2. Bridging a modeling chasm: design/develop and delivery/execution (15 points)
3. Open world assumption: uncertainty (15 points)
4. IT business alignment, adaptability (15 points)
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. Integration of programming models and runtime (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 a.k.a. dealing with heterogeneity (13)
12. Making the leap from business service to the right technical service design (11)
13. Alignment of business and technical level in SSE (12)
14. Composability (11)
15. Testing (11)

Clearly, these research challenges are closely related to the SSE tenets. Table 1 loosely correlates research challenges to the SSE tenets. Note that SSE tenet 7 pertains to all research challenges and has therefore not been included in this table. From this initial and informal cross-correlation we may carefully draw some very preliminary conclusions.

Firstly, 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- whilst other research challenges address specific problems such as service composability and service testing.
Research challenges relating to SSE tenet “Technical Federation” include the design of service applications without any knowledge about the context in which they will be executed. This research challenge is critical in open and agile service networks, with many unpredictable interactions between service participants. In addition, there is a need for novel approaches to integrate programming models and platforms while processes –some of which may in fact be transactional in nature- in service networks are executed. The high level of change in service networks also demands services to be dependable.

Because of the ‘open-world assumption’ and the dynamisms of service-based applications, traditional test methods for system development and deployment are not enough: 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 or run-time auditors and supervisors are first developments in this direction.

Table 1 Correlation of SSE Tenets and Challenges

SSE Tenet	*Description*	*Challenge ID*
1	Technical Federation	7, 8, 9, 14, 15
2	Dynamism	1, 3, 6, 15
3	Organizational Federation	1, 3, 7
4	Boundaries	10, 12
5	Heterogeneity	11
6	Alignment	2, 4, 5, 13, 15

The ‘open world assumption’ renders 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 the open world requires. Furthermore the traditional architecture-business cycle [19] that expresses the bidirectional influence between the technical system and the business organization cannot be managed using traditional architecting methods, because of the high dynamism and heterogeneity. Therefore the architecting dimension of SSE needs to be thoroughly re-considered, potentially leading to a new architecting paradigm.

5. Syntheses and Outlook

SOA-enabled applications cannot be simply developed and evolved by applying aging software engineering paradigms, notably CBD and OO. 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 [17].

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 systems – operating in highly complex, distributed, unpredictable, and heterogeneous execution environments. In addition, the service networks that are designed based on this assumption need to be continuously (re-)aligned with business processes, and vice versa. Adoption of the open-world assumption is reflected in the seven SSE tenets, which are thus strongly influenced by the underlying distributed computing paradigm: SOA.

Based on the research reported in this article, we can now come up with an initial definition of SSE as *the science and application of concepts, models, methods, and tools to design, develop/source, deploy, test, provision, 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 her elderly parent software engineering; however, we herein argue that aging computing model specific principles and practices, e.g., distributed component technology, clearly need revision given the exclusive nature of SOA.

In our view, SSE will be based on standards and will be frequently realized with Web services. In fact, languages such as SOAP, WSDL, BPEL, WS-Policy, WS-Agreement already constitute the first step to realize the technical aspects in some of the SSE tenets, including tenets 1, 2, 4 and 5. However, it is evident that research is needed 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 presented in this article are core results in nature. Further work is required in several directions. Firstly, the list of seven tenets has to be validated and possibly refined further. Indeed, the presented list is derived from a literature survey, and, 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, analysis of more case studies is critical in further validating this initial list. The ICSE workshop will serve as a first step to achieve this. In addition, the research challenges will be consolidated in a future roadmap for SSE.

References

[1] Report on SSE Dagstuhl seminar, 90021. (To appear)
[2] F. Bachmann et al. “Technical Concepts of Component-Based Software Engineering”, Technical Report, Carnegie-Mellon Univ., CMU/SEI-2000-TR-008 ESC-TR-2000-007, 2nd Edition, May 2000
[3] C. Hofmeister, P. Kruchten, R. Nord, H. Obbink, A. Ran, P. America. A general model of software architecture design derived from five industrial approaches. J. Syst. Softw., Elsevier Science Inc., 2007, 80, 106-126
[4] G. Alonso and F. Casati and H. Kuno and V. Machiraju, Web Services: Concepts, Architectures and Applications, Springer, Heidelberg, 2004
[5] B. Meyer, Object-oriented Software Construction, 2nd Edition. Prentice Hall, 2000
[6] G. Hohpe, SOA Patterns – New Insights or Recycled Knowledge? Available from www.eaipatterns.com/docs/SoaPatterns.pdf
[7] M.P. Papazoglou, and W. van den Heuvel. Service-oriented design and development methodology. Int. J. Web Eng. Technol. Vol. 2(4), Jul. 2006
[8] M.P. Papazoglou, W. van den Heuvel. Service oriented architectures: approaches, technologies and research issues. VLDB J. 16(3): 389-415, 2007
[9] C. Szyperski. Component technology: what, where, and how? In: Proceedings of the 25th international Conference on Software Engineering International Conference on Software Engineering. IEEE, 684-693, 2003
[10] P. Herzum, O. Sims “Business component Factory”, J. Wiley & Sons Inc., 2000
[11] G. Booch. Object-Oriented Analysis and Design with Applications (2nd Ed.). Benjamin-Cummings Publishing, 1994
[12] I. Jacobson. Object-Oriented Software Engineering. ACM, 1992
[13] Popescu-Zeletin, R.; Arbanowski, St.; Fikouras, I.; Gasbarrone, G.; Gebler, M.; Henning, H.; van Kranenburg,

H.; Portschy, H.; Postmann, E.; Raatikainen, K.: Service Architectures for the Wireless World. Computer Communications, Vol. 26, No. 1, January 2003, pp. 19 – 25
[14] B. Sarikaya. Principles of protocol engineering and conformance testing, Ellis Horwood, Series in Computer Communications and Networking, 1993
[15] I. Schieferdecker, J. Grabowski: Advances in Test Automation, STTT Special Issue, Springer Berlin / Heidelberg, ISSN1433-2779, Apr. 2008
[16] B. Meyer, Object-oriented software construction (2nd ed.), Prentice-Hall, Inc., Upper Saddle River, NJ, 1997
[17] B. Boehm. 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
[18] C. Atkinson, D. Brenner, G. Falcone, M. Juhasz. Specifying High-Assurance Services. *Computer* 41, 8 (Aug. 2008), 64-71
[19] L. Bass, P. Clements, R. Kazman. Software Architecture in Practice, 2nd Edition, Addison Wesley, 2003
[20] O. Zimmermann, P. Krogdahl, C.Gee, Elements of Service-Oriented Analysis and Design, IBM developerWorks, 2004