Towards Self-Organizing Service-Oriented Architectures

Walter Binder, Daniele Bonetta, Cesare Pautasso, Achille Peternier
Faculty of Informatics
University of Lugano (USI)
Via G. Buffi 13
Lugano, Switzerland
Email: first.last@usi.ch

Diego Milano, Heiko Schuldt, Nenad Stojnić
University of Basel
Bernoullistrasse 16
Basel, Switzerland
Email: first.last@unibas.ch

Boi Faltings, Immanuel Trummer
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Artificial Intelligence Laboratory
IN Building Station 14
Lausanne, Switzerland
Email: first.last@epfl.ch

***Abstract*—Service-oriented architectures (SOAs) provide a successful model for structuring complex distributed software systems, as they reduce the cost of ownership and ease the creation of new applications by composing existing services. However, currently, the development of service-oriented applications requires many manual tasks and prevailing infrastructure is often based on centralized components that are central points of failure and easily become bottlenecks.**

In this paper, we promote self-organizing SOA as a new approach to overcome these limitations. Self-organizing SOA integrates research results in the areas of autonomic and service-oriented computing. We consider self-organizing features for the whole life-cycle of a service-oriented application, from the creation to the execution, optimization, and monitoring.

***Keywords*-SOA; Web services; service composition; self-* systems;**

I. Introduction

Service-oriented architectures (SOAs) offer many compelling opportunities to address pressing problems in large IT infrastructures and enterprise application integration. SOAs promote a specific approach for building distributed applications by composing reusable services thanks to well-defined interoperation semantics based on standard protocols. This emerging approach to software development based on reuse and composition promises many benefits, such as extensibility, ease of maintenance, and reduced development effort and cost. For these reasons, SOAs have recently attracted much attention in both academia and industry [23], [15], [34].

In this paper we describe the vision of *self-organizing SOAs* by exploring a novel, self-organizing approach to the design and life cycle support of next-generation SOAs. The main objective is to overcome the following limitations of current SOA [12]. Due to the success of SOA, the problem of managing large collections of services has become crucial. Also known as the SOA governance problem, the issue reflects that little is known about deploying, invoking, monitoring, and providing load balancing and fault tolerance over a highly dynamic and potentially very large collection of services each one possibly representing a complex application running on a heterogeneous cloud-based infrastructure [29].

Composite services still need to be designed by hand, requiring the software designer to manually search repositories of service advertisements for relevant services and explicitly define the logic to glue them together. Existing languages and tools offer little support for the automated evolution, repair, and tuning of composite services. Most SOA middleware implementations rely on centralized services (e.g., centralized repositories of service advertisements) that are single points of failure, easily become performance bottlenecks, and may limit the scalability of the overall architecture. In addition, the current standardized service abstraction does not consider advanced interaction patterns as it is limited to asynchronous message-based and synchronous request/response interactions.

In this paper we also present a roadmap to address the aforementioned limitations and investigate the design

and implementation of self-organizing SOAs. Overall, the idea is to explore scalable, decentralized solutions to the problems of current SOAs and investigate automation of important aspects of the whole lifecycle of a service-oriented application, including resource and service discovery, binding and composition, deployment and monitoring, RESTful and stream-based interaction, as well as fault detection and repair. The constraint is to avoid heavy-weight centralized solutions and to provide a decentralized, self-organizing, and light-weight infrastructure that can simplify most of the management and reduce the execution overhead of services.

Our proposed roadmap addresses the investigation of novel algorithms for automatically composing services based on high-level task specifications that can also be applied to repair existing compositions. This requires designing and experimenting with innovative middleware architectures that exhibit self-configuration, self-healing, and self-tuning capabilities to enable the dynamic evolution and optimization of composite services. For example, composite services will be automatically reconfigured such that they always make use of the best available services at each moment.

Furthermore, we suggest to compare both reactive and proactive solutions and study under which conditions non-functional properties associated with services can be guaranteed. In order to go beyond the current service abstraction, this requires an extension to services with the capability of generating and processing continuous streams of data of infinite length and by applying the notion of composition to RESTful services.

Finally, also as part of our roadmap, we put in practice and perform real experiences by defining application scenarios and specifying underlying assumptions. A software architecture with well-defined component interfaces eases the integration of components for an end-to-end evaluation of the resulting system.

The main contribution of this paper is to outline selected research opportunities in the context of self-organizing SOAs. These opportunities have been identified based on our collective experience in industry projects and by surveying the existing scientific literature. We do not claim completeness of the set of identified research challenges, as the goal of this paper is to foster discussion on a roadmap for future research in the area of SOAs and self-organizing, adaptive, and self-managing software.

This paper is structured as follows: In Section II we introduce basic concepts and state-of-the-art related to the topics of self-organizing SOAs. Section III presents a series of issues, research ideas, and potential contributions in our roadmap. Section IV exposes the methodology we adopt to validate our results. Finally, Section V contains our conclusion.

II. Background and Related Work

In this section we give a brief overview over *Service-Oriented Architectures* and *Self-Organizing Systems*, including references to some recently published surveys.

A. Service-Oriented Architectures

The service-oriented architectural style [40] changes the way we think about building integrated software systems as it brings forward a new abstraction: the service. In a SOA, software components with standardized interfaces are delivered as services on demand [34], [41]. Web services are the current technology which delivers the interoperability necessary to implement these architectures across distributed, autonomous, and heterogeneous systems [1], [42].

SOAs have recently appeared as the force driving the reduction of the total cost of ownership of enterprise applications [23]. Applications no longer have to be separately installed and maintained. Instead, their constituent software components are delivered as a service to the enterprise [19]. SOAs also intend to reduce the cost of integrating enterprise applications [9]. Thanks to the recent standardization efforts ensuring the interoperability of the underlying middleware tools and protocols [35], it has become much easier to compose new applications out of novel combinations of existing services having well-defined interfaces [30], [14].

The implications of this new paradigm, where software is delivered as a service, are not yet fully understood. The advantages [43], [26], [44] in terms of cost reduction, flexibility, reuse, and the possibility of outsourcing part of the IT infrastructure need to be placed in the context of some related problems that require further research [36], [37]:

1) New abstractions appear before tools are ready to support them. Existing tools (i.e., object-oriented programming languages or component frameworks and containers) are struggling to be extended to incorporate the technologies required by the new paradigm. Still, there is a mismatch between the new paradigm and the capabilities of available tools [16]. This is a great window of opportunity to try out innovative approaches for effectively building integrated systems using the service-oriented computing paradigm.

2) Abstractions can hide but cannot remove the distributed nature of the interactions in SOAs [17]. One of the challenges in this context is how to build reliable applications out of services that are outside the control of their users and may suffer from temporary unavailability. Reliable message delivery techniques help in this regard, but change the familiar programming metaphor based on synchronous calls into asynchronous event-driven programming [7].

3) A loosely coupled system implemented using a SOA is meant to be flexible and easier to evolve. Although this is indeed the case for some kinds of changes (e.g., when moving the network location of a service provider), paradoxically, a loosely coupled system makes it even more

difficult to asses the global impact of other changes (i.e., modifications to service interfaces) [37]. We believe that this concern can only be addressed through a sound foundational approach that leverages modern advances in the engineering of self-organizing and autonomic middleware systems.

B. Self-Organizing Systems

Self-organization defines the capability of systems to feature emergent properties through the collective behavior of their components acting independently and in a decentralized fashion [3]. Whereas self-organization is a common property of living systems, it remains a major challenge to design artificial systems with an architecture capable of achieving a similar degree of self-organization.

Autonomic computing is an emerging research field addressing the challenge of building computing systems capable of managing themselves in accordance with high-level goals, policies, and objectives specified by humans [25], [22]. Given the growing size and scale of distributed information systems in general and modern service-oriented architectures in particular, the importance of providing solutions to effectively deal with the complexity of such systems has now been widely recognized [24]. A recent survey can be found in [21], while [31] proposes a reference architecture for self-organizing SOA-based computing.

In the context of autonomic computing, self-organization has been proposed as a viable design alternative to centralized solutions [20] for automatically managing the configuration, the composition, the performance, and handling failures within such systems [2], [18], [4], [13]. Self-organizing systems present so-called self-* properties, which in the context of SOAs can be interpreted as follows [37]:

Self-Configuration: refers to the ability of services to automatically configure themselves to adapt to different deployment environments.

Self-Tuning: is the ability of automatically tune the performance of services based on online monitoring of workload conditions and control of resource allocation and utilization.

Self-Healing: services can detect faults, diagnose failures, and react to disruptions in order to automatically maintain the overall system in operation.

Self-Protection: assuming that the communication between services is secure, services proactively detect intrusions and can resist denial-of-service attacks.

III. Towards Self-Organizing SOAs: a Research Roadmap

The intersection of service-oriented computing and autonomic computing is a rich source of problems, which needs to be studied in order to achieve building of self-organizing SOAs.

In general, the objective of a self-organizing service-oriented system is to effectively and automatically *deal with change*. Change can happen at all levels (i.e., communication topology, middleware infrastructure, availability of service providers, and application environment) and needs to be properly accounted for within the "service" abstraction we are proposing in this roadmap.

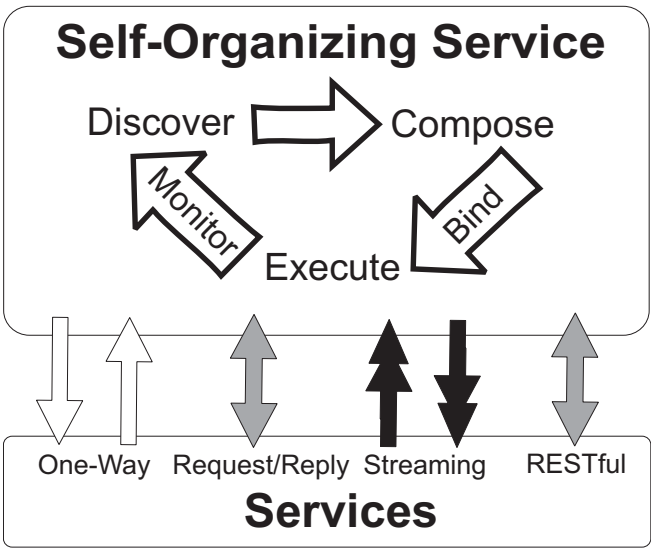

Figure 1. Self-organizing services.

We broadly classify the research topics in our roadmap following the most important aspects of the lifecycle of a service-oriented application, including *discovery*, *composition*, *binding*, *execution*, *streaming invocation*, *monitoring*, and *management*. Figure 1 gives a conceptual view of a self-organizing service that manages its lifecyle in an autonomic way. The self-organizing service integrates a set of services (which themselves can be self-organizing services, though this is not a requirement) and interacts with these services either through one-way communication, request/reply invocation, streaming data transfer, or RESTful interaction [38]. A self-organizing service is created by automated composition that relies on discovery to find relevant services. Before execution, binding selects the optimal concrete communication end-points among a set of alternative service providers. The self-organizing service monitors its execution and may dynamically change its structure and service dependencies so as to provide self-configuration, self-tuning, and self-healing features. This is illustrated in Figure 1 by the closed loop linking the various phases of the lifecycle of a self-organizing service. In some cases, depending on the characteristics of the service, some phases may be skipped. For example, rebinding to a newly discovered service provider may not always require to change the structure of a composition.

The challenge is to explore how to design a distributed infrastructure with a small system footprint for the different phases of the self-organizing service lifecycle. This will allow for a high degree of scalability which, in turn, is

one of the main preconditions for applying the idea of self-organizing SOA in realistic settings.

In the following text we consider the aforementioned phases of the self-organizing service lifecycle, discuss how change impacts each of these phases, and argue how self-organization helps dealing with change. For each topic, we highlight the open issues and research challenges targeted by our roadmap.

A. Discovery

Many current SOAs suffer from a centralized approach to service discovery. While a centralized service repository is able to process complex matchmaking queries (e.g., [27]) since all the possibly relevant service data is stored in a single place, it represents a central point of failure and performance bottleneck, reducing the scalability and dependability of the overall system.

Distributed service discovery in current peer-to-peer (P2P) systems can avoid the problems of a centralized service repository, but often imposes strong restrictions on the type of queries that are supported, or may suffer from reduced precision and recall, or can cause high communication overhead if too many nodes in the P2P system need to be contacted.

To this end, we believe it is important to explore new approaches to provide scalable and dependable service discovery. To address this issue, techniques used in centralized databases, such as multidimensional index structures and multiversioning need to be combined with distribution and replication algorithms using P2P middleware. Similarly, parallelized query execution, for example by exploiting recent multicore hardware, needs to be taken into account.

B. Composition

A major topic on the roadmap for self-organizing SOA concerns the development of new techniques for automated service composition. In particular, an interesting problem is how to design and develop techniques that provide the possibility to gradually trade optimality of the result for the composition against effort required to compute it, thus allowing both to address highly dynamic short-term needs as well as optimization of compositions that will be used in the longer term.

Another important issue is related to constraints on computation and communication resources without a central authority. Such constraints are essential in applications such as sensing and monitoring, where large quantities of data must be manipulated under strong resource constraints. They create a web of dependencies between different composition processes that must be handled in a distributed algorithm.

Automated service composition can be seen as a generalization of service discovery: if a single matching service can not be found, composition tries to integrate a set of services in order to meet given requirements. Hence, automated service composition relies on service discovery and requires particular features of the discovery component, such as the support for partial matches when processing queries.

C. Execution

As a fundamental design constraint, the middleware for the execution of composite services comes without any central component, i.e., it is itself inherently distributed. This allows for a scale-out at the middleware level and guarantees a high degree of scalability, both in terms of the number of basic services that are available and in terms of the number of composite services to be executed concurrently.

The challenge is to apply self-tuning techniques to automatically optimize the performance of this decentralized architecture. Based on an open set of metrics to observe system performance, reconfiguration policies and actions will be applied to dynamically allocate resources and migrate components between nodes taking into account the expected benefits of the reconfiguration weighted against the cost of applying changes to the running system.

Reliable execution of composite services by making use of advanced self-healing mechanisms is also another major challenge on the way to self-organizing SOA. This task will be strongly facilitated by making use of the type associated with individual services (i.e., maintain the state associated with a service invocation between requests whenever possible, in order to derive the information necessary for compensation). This also includes alternative execution paths (either statically or dynamically defined) for exploring forward failure handling techniques.

D. Binding

Flexibility in the binding of services to compositions is an essential pre-requisite for self-healing and self-tuning of a service-oriented application. As a first step, it is important to enumerate a set of dynamic binding patterns that will bring the results of service discovery closer to the composition in a parametric way. This initial work will become the foundation for the design and evaluation of a spectrum of different alternatives ranging from advance resource reservation (e.g., to reserve all resources needed for a composition at the time its execution starts) up to advanced runtime load balancing mechanisms (e.g., in case several instances of the same service type are available, the one with least expected load, lowest prize, optimal quality of service guarantees, or, in general, based on a predefined measure of "proximity" can be chosen).

E. Streaming Invocation

In addition to message-based synchronous and asynchronous interactions supported by current SOAs, an extension of the current notion of service to support stream-based interactions is needed for a variety of applications. This will not only broaden the domain of application of

self-organizing SOA results, but also allow to apply service-orientation to pervasive and ubiquitous applications where the need for self-organization has recently become apparent.

Streaming services extend the existing discrete service abstraction to bring continuous data streams of software or hardware sensors into a SOA. This requires to tackle the challenge of correlating the flow of stream elements among multiple sources and maintaining the state of a stream-based interaction in a reliable way. Also, both streaming and non-streaming services need to be jointly considered in the same application, for example by using a form of pipelined processing or by triggering the execution of a (non-streaming) composite service if certain application-specific characteristics are detected within a certain stream.

F. Monitoring and Management

An infrastructure to support self-organizing SOA will allow a user to specify desired quality-of-service (QoS) characteristics at the level of composite services. These QoS characteristics will be automatically broken down into requirements for individual services. By establishing service-level agreements with the respective service providers, the infrastructure will support a novel approach based on advance resource reservation protocols that will support re-binding and integrate well with the decentralized approach to service discovery.

IV. Methodology and Scenario

The path to self-organizing SOAs passes through theoretical contributions on fundamental algorithms and designs for self-organizing services, including their evaluation on a collection of real-world use cases and application scenarios. According to our research roadmap, we propose the following methodology.

Discovery: scalable discovery algorithms and services will be developed which are key for automated service composition and for the automated evolution and self-healing of composite services. In the design of discovery algorithms, we therefore consider the requirements and constraints coming from automated service composition and from optimized service selection at execution time.

Expertise in the design of service matchmakers, as well as on indexing and concurrency control in service repositories [10], [11], [5], [6], still needs further leveraging to explore and develop new approaches for scalable service discovery.

Composition: we suggest to address the composition problem using recently developed algorithms for distributed constraint optimization in a framework that fits with the planning framework developed above. The results of these novel automatic composition algorithms will need to be visualized to be inspected and monitored by users. The algorithms will therefore be integrated with existing service composition environments (e.g., JOpera[1]) that offer a strong complementary approach to automatic service composition.

Execution: the execution of service compositions is one of the main topics at the center of our research focus. We earned a significant expertise in the design of distributed service execution engines: both the OSIRIS [39], [32] and JOpera engines feature a scalable approach to workflow execution with a cluster-based, multicore-based, and a P2P architecture. Thus, experimentation based on these research platforms is promising as we already have an underlying architecture for service composition featuring the necessary mechanisms that enable self-organization properties.

Binding: we propose the investigation of *service re-binding*, where a binding is dynamically changed at runtime in case of exceptional conditions (e.g., when services are not available [33] or when advance reservations [28] are invalidated). This self-healing feature requires the definition of customizable metrics to characterize individual and composite service instances and also to derive the characterization of composite services out of the included basic services and the structural characteristics of the composition.

Streaming invocation. Applying re-binding to streaming services is particularly challenge because it requires proper session management and state transfer for the reliable hand-off of an ongoing streaming interaction [8]. We propose the application of the notion of streaming service in the context of RESTful services, for example, to study how to efficiently compose Web data feeds. This will require the development of new techniques to transparently manage pipeline buffers and data caches.

Monitoring and management. Research in this area will make intensive use of the provenance features of our middleware, where execution of composite services will be thoroughly logged. On the basis of these logs, the autonomic management policies can be evaluated *a posteriori* and the procedure to decide about the self-configuration of service-level agreements can be properly adapted.

Experimentation and benchmarking. Due to the heterogenous and distributed nature of any SOA-based deployment, we privilege an experimental approach to research, where empirical validation of novel algorithms and techniques is preferred to results obtained by modeling and simulation. This is an appropriate methodology as it is applied to our domain, where it is hard to establish realistic modeling assumptions. Theoretical considerations, such as regarding the runtime complexity of algorithms, will complement the empirical assessment. A good experimental scenario candidate consists of a self-organizing *grid of services* for large-scale computations using huge amounts of data, such as in astronomy or in meteorology. This scenario involves distributed data sources and data processing services that need to be composed according to high-level goal specifica-

[1] http://www.jopera.org

tions. Self-organization will be achieved by a combination of automated composition integrated with service discovery, advance resource reservation, dynamic service rebinding or re-composition in case of service failure or when better performing services become available, as well as stream processing for handling large data sets. Additionally, the self-configuration techniques developed can be evaluated on real-world benchmarks and developed into open-source implementations publicly available[2].

V. Conclusion

The research challenges identified in this paper are not only of high scientific relevance, but they also address serious limitations in the way current SOAs are designed and maintained. Solving these challenges will significantly advance the state-of-art in SOAs.

Self-organizing SOAs will drastically reduce the costs of both software reuse and integration and, most important, also simplify the maintenance of complex software systems. More specifically, automated service composition significantly reduces the effort needed to create service-oriented applications. The integration of automated composition with dynamic service discovery ensures higher quality of the applications that are composed leveraging the best available services that can be found over time. The advanced execution features (self-configuration, self-optimization, self-tuning, and self-protection) that define an infrastructure for self-organizing SOA promise to take the burden of manually updating, correcting, and tuning the performance of composite services away from the software producer. When combined with the results on RESTful service composition and data streaming, these will also provide the key capabilities to address a rich set of challenging application scenarios, in addition to traditional SOA-related ones, that can be used to demonstrate and benchmark the developed techniques.

Acknowledgments

The work presented in this paper has been funded by the Swiss National Science Foundation (SINERGIA grant nr. CRSI22_127386) in the context of the SOSOA project.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web services: Concepts, Architectures and Applications*. Springer, November 2003.

[2] O. Babaoglu, M. Jelasity, and A. Montresor. Grassroots approach to self-management in large-scale distributed systems. In *Proceedings of the EU-NSF Strategic Research Workshop on Unconventional Programming Paradigms*, Mont Saint-Michel, France, September 2004.

[3] O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, and M. van Steen. *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations*, volume 30 of *LNCS*. Springer, 2005.

[4] L. Baresi and L. Pasquale. Live goals for adaptive service compositions. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, pages 114–123, 2010.

[5] W. Binder, I. Constantinescu, and B. Faltings. A flexible directory query language for the efficient processing of service composition queries. *International Journal of Web Services Research*, 4(1), 2007.

[6] W. Binder, A. Mosincat, S. Spycher, I. Constantinescu, and B. Faltings. Multiversion concurrency control for the generalized search tree. *Concurrency and Computation: Practice and Experience*, 21:1547–1571, 2009. http://dx.doi.org/10.1002/cpe.1387.

[7] K. Birman. *Reliable Distributed Systems: Technologies, Web Services and Applications*. Springer, 2005.

[8] G. Brettlecker and H. Schuldt. Reliable Distributed Data Stream Management in Mobile Environments. *Information Systems Journal*, Nov. 2010.

[9] C. Bussler. *B2B Integration. Concepts and Architecture*. Springer, 2002.

[10] I. Constantinescu, W. Binder, and B. Faltings. An extensible directory enabling efficient semantic web service integration. In *3rd International Semantic Web Conference (ISWC 2004)*, pages 605–619, Hiroshima, Japan, Nov. 2004.

[11] I. Constantinescu, B. Faltings, and W. Binder. Large scale, type-compatible service composition. In *IEEE International Conference on Web Services (ICWS-2004)*, pages 506–513, San Diego, CA, USA, July 2004.

[12] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15:313–341, 2008. 10.1007/s10515-008-0032-x.

[13] S. Dustdar, C. Dorn, F. Li, L. Baresi, G. Cabri, C. Pautasso, and F. Zambonelli. A roadmap towards sustainable self-aware service systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, pages 10–19, 2010.

[14] S. Dustdar and W. Schreiner. A survey on web services composition. *International Journal of Web and Grid Serivces (IJWGS)*, 1(1):1–30, 2005.

[15] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[16] D. Florescu, A. Grünhagen, and D. Kossmann. Xl: an xml programming language for web service specification and composition. *Computer Networks*, 42(5):641–660, 2003.

[17] C. Ghezzi. The challenges of open-world software. In *Proceedings of the 6th International Workshop on Software and Performance (WOSP 2007)*, page 90, Buenos Aires, Argentina, February 2007.

[2] http://sosoa.inf.usi.ch/Instruments

[18] R. A. Golding and T. M. Wong. Walking toward moving goalposts: agile management for evolving systems. In *HotAC I, the First International Workshop on Hot Topics in Autonomic Computing*, May 2006.

[19] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to Web services architecture. *IBM Systems Journal*, 41(2):170–177, 2002.

[20] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, August 2004.

[21] M. C. Huebscher and J. A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys*, 40(3):1–28, 2008.

[22] IBM. Autonomic Computing: Special Issue. *IBM Systems Journal*, 42(1), 2003.

[23] N. M. Josuttis. *SOA In Practice*. O'Reilly, August 2007.

[24] J. O. Kephart. Research challenges of autonomic computing. In *Proc. 27th International Conference on Software Engineering (ICSE2005)*, pages 15–22, May 2005.

[25] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[26] R. Khalaf, A. Keller, and F. Leymann. Business processes for web services: Principles and applications. *IBM Systems Journal*, 45(2):425–446, 2006.

[27] M. Klusch, B. Fries, and K. Sycara. Owls-mx: A hybrid semantic web service matchmaker for owl-s services. *Web Semant.*, 7:121–133, April 2009.

[28] C. Langguth and H. Schuldt. Optimizing Resource Allocation for Scientific Workflows Using Advance Reservations. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM 2010)*, volume 6187 of *Lecture Notes in Computer Science*, pages 434–451. Springer, June-July 2010.

[29] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 23–31, Washington, DC, USA, 2009. IEEE Computer Society.

[30] F. Leymann, D. Roller, and M.-T. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.

[31] L. Liu, S. Thanheiser, and H. Schmeck. A reference architecture for self-organizing service-oriented computing. In U. Brinkschulte, T. Ungerer, C. Hochberger, and R. Spallek, editors, *Architecture of Computing Systems ARCS 2008*, volume 4934 of *LNCS*, pages 205–219. Springer Berlin / Heidelberg, 2008.

[32] D. Milano and N. Stojnić. Shepherd: node monitors for fault-tolerant distributed process execution in osiris. In *Proceedings of the 5th Workshop on Emerging Web Services Technology (WEWST 2010)*, pages 26–35, Ayia Napa, Cyprus, Dec. 2010.

[33] T. Möller and H. Schuldt. OSIRIS Next: Flexible Semantic Failure Handling for Composite Web Service Execution. In *Proceedings of the 4th IEEE International Conference on Semantic Computing (ICSC 2010)*, Pittsburgh, PA, USA, Sept. 2010.

[34] E. Newcomer and G. Lomow. *Understanding SOA with Web Services*. Addison Wesley, 2005.

[35] H. R. M. Nezhad, B. Benatallah, F. Casati, and F. Toumani. Web services interoperability specifications. *Computer*, 39(5):24–32, May 2006.

[36] M. P. Papazoglou and W.-J. Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[37] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[38] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly, May 2007.

[39] C. Schuler, H. Schuldt, C. Türker, R. Weber, and H.-J. Schek. Peer-to-peer execution of (transactional) processes. *International Journal of Cooperative Information Systems*, 14(4):377–406, 2005.

[40] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[41] C. Szyperski. Component technology - what, where, and how? In *ICSE '03: Proc. of the 25th International Conference on Software Engineering*, pages 684–693, Portland, Oregon, 2003.

[42] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture*. Prentice Hall, 2005.

[43] O. Zimmerman, M. Tomlinson, and S. Peuser. *Perspectives on Web Services: Applying SOAP, WSDL, UDDI to Real-World Projects*. Springer, 2003.

[44] O. Zimmermann, M. Milinski, M. Craes, and F. Oellermann. Second generation web services-oriented architecture in production in the finance industry. In *OOPSLA Conference Companion*, 2004.