Towards a Methodology for Lifelong Validation of Service Compositions

Domenico Bianculli
Faculty of Informatics
University of Lugano
via G. Buffi 13 - CH-6900, Lugano, Switzerland
domenico.bianculli@lu.unisi.ch

Carlo Ghezzi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
p.za Leonardo da Vinci 23, I-20133, Milano, Italy
carlo.ghezzi@polimi.it

ABSTRACT

We argue that the dynamic nature of compositions in service-oriented architectures requires re-thinking the whole lifetime of an application, from requirements, to design and implementation, to deployment, to operation. In particular, the traditional boundary between design time and run time is blurring. Validation, which traditionally pertains to design time, must now extend to run time. This position paper digs into the requirements and outlines a research agenda, based on the experience gained in previous research projects, to engineer dependable service compositions.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Validation*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Specification techniques*

General Terms

Languages, Design

Keywords

Service composition, Specification language, Validation, Contract

1. INTRODUCTION

Service-oriented architectures (SOAs) are emerging as a promising solution to the problem of developing decentralized, distributed, and evolvable applications that live in an open world [5]. In these architectures, *services* represent software components that provide specific functionality, exposed for possible use by many clients, who can then dynamically discover the services and access them through network infrastructures [20].

SDSOA'08, May 11, 2008, Leipzig, Germany.

This emerging scenario is highly dynamic, open, and decentralized. The global system is not under control and coordination of a single authority. In principle, and according to an extreme viewpoint, multiple autonomous stakeholders contribute to the wealth of available resources. Furthermore, new added-value services may be provided by composing existing services and, recursively, service compositions. To take advantage of the availability of new resources and new services, compositions should also be highly dynamic.

Understanding service composition is a key to build quality service-oriented applications. In this paper we outline a research road map that focuses on dependable service compositions, i.e., compositions that can be validated in a rigorous manner to assess that the provided services satisfy their goals, expressed in terms of some predefined functional and non-functional properties. This research road map is based on the experience gained in the context of several research projects, most notably the EU IST SecSE [36] and PLASTIC [31] projects.

The rest of this paper is structured as follows. Section 2 describes the main issues in developing service compositions, while Sect. 3 illustrates a research road map to overcome those challenges. This road map is at the basis of a validation methodology that is proposed in Sect. 4. Section 5 discusses the related work. Section 6 concludes the paper, by summarizing our research vision.

2. PROBLEM AREAS

In this section we describe the main problem areas that should be considered for defining a research road map, whose ultimate goal is to develop a comprehensive methodology for engineering dependable service compositions.

Service Composition Facets

SOA-based applications are built by integrating distributed components that export one or more functionalities as services. There are two models of service composition [11]:

- *Process-oriented* composition is achieved by means of a workflow language that orchestrates the execution of external services to define a new service component.
- *Structural* composition identifies the participating components of a composite service and how these components are connected.

Developing service compositions thus requires to deal with both aspects of the composition, that is both the structural view, which focuses on the interaction of a (composite) service with external services, and the internal definition, in

terms of business logic, of the composite service. The composition might be described at different levels of abstraction and also by separating different concerns. For example, one might define a high-level workflow and separately define policies that detail how certain abstract services can dynamically be accomplished by some lower-level concrete services. A key point of service composition is that the ultimate behavior and the quality of service (QoS) provided by a composite service is the result of the aggregation of the behavior and the QoS of individual, external services, composed according to the internal workflow definition. How to produce such a composite behavioral and QoS description out of the behavior and the QoS of individual, external services is a key issue.

Services in an open world

The dynamic world of SOAs is an instantiation of the *open-world software* described in [5]. In such a world, services may evolve dynamically and autonomously, and even unexpectedly, even after the system is deployed: new services may be developed and published in registries, and then discovered by possible clients; previously available services may disappear or become unavailable. As opposed to the conventional components in a component-based system, services are developed, published and run by independent parties. A service composition can thus be made out of services on which the provider has no jurisdiction: this leads to situations in which a provider might never truly possess complete knowledge of the system it is providing, even if it is ultimately responsible for the overall functionality and quality of service of the composite system. This high degree of dynamism has a severe impact on —amongst other aspects— the system's correctness and on the way validation of service compositions can be performed.

Service Contracts

Contracts play a very important role in the realm of SOAs. Since services are developed by independent parties, it is very unlikely that a service requester will get access to the internal description of a service. The only available information for interacting with a service is its contract. Contracts thus allow for interoperability, enabling the interaction of services developed and operated in different organizational domains. A service contract may describe different aspects of a service, among which we list:

- *Interface*: it consists of the operations provided by the service, where each operation is described in terms of its input and output parameters.
- *Behavior*: it describes functional properties of the service like pre- and post-conditions and invariants on a service operation, or more generic assertions on the historical trace of the interaction with the service.
- *Quality of Service*: it describes the non-functional properties of the interaction with a service, including attributes like response time, throughput, availability, and policies concerning security and reliability of the communication.

We believe that all these different aspects of service contracts should be captured by the expressiveness of a specification language for service contracts.

Technologies for implementing SOAs

Although some authors [15] connect the definition of an SOA to its implementation by means of web services technologies, we want to point out the web services are only one *possible* —even though the most used— way to realize SOAs, since there are other technologies to accomplish that, like OSGi, Jini, message-oriented middleware etc. Therefore, we believe that research efforts should adopt a technology-independent approach when proposing solutions supporting the engineering phase of SOAs.

Service analysis and validation

As will be detailed in Sect. 5, there are many existing research efforts that focus on different aspects for analysing and validating service compositions, thus resulting in several proposals dealing with either specification languages, or design- and run-time validation techniques. However, there is no approach —with the few exceptions of [33] and [3], which focus on web service orchestrations— that integrates uniformly a specification language with techniques for design- and run-time analysis and validation of service compositions. Therefore, we believe that a holistic approach is necessary to validate modern dynamic applications based on service compositions, in a technology-independent manner.

3. RESEARCH DIRECTIONS

The research problems discussed above bring forth several research directions, which will be presented in the rest of this section.

The need for continuous, lifelong validation of service-based applications

Coping with services living in an open-world requires lifelong validation activities: besides performing design-time validation, it is also necessary to perform continuous run-time validation to ensure that the required properties are maintained by the operational system.

Definition of a high-level, abstract language for service compositions

Service compositions are usually described using orchestration, workflow-based languages like WS-BPEL [29] or manually implemented using common programming languages like Java. The heterogeneity of existing languages and standards, and the willingness to adopt technology-independent approaches, advocate to propose a high-level language for defining abstract models of service compositions, easily mappable both into concrete implementations and into formal models for guaranteeing rigorous and sound analysis of the models themselves. Furthermore, the language should factor in constructs that allow a service aggregator to use well-known service compositions patterns.

Definition of a specification language for service contracts

The key-role played by contracts in SOA-based applications, requires a well-designed language for stating *service contracts*, i.e., the properties that services part of a composition should guarantee and that are to be verified both at design time and at run time. The expressiveness of the language is an outstanding quality attribute since, as discussed before, a contract may describe different aspects of a service. In

particular, the language should allow for the specification of non-functional and functional properties, such as assertions on the service's state and data values, and properties on historical traces of events; moreover, it should support the specification of both stateless and stateful interactions. Special care should be adopted during the design phase of the language to find the right balance between the expressiveness and the usability of language. Moreover, the language should be aware of the constructs defined in the *service composition language* mentioned above, to enable associating contracts with specific execution points of the composite service. Furthermore, the language should support an assume-guarantee approach [13], to promote modularity and reuse of specifications and allow for a partial specification of the evolvable dynamic environment where services are executed.

Design-time validation

By using the two languages described above, an engineer should be able to design a service composition and annotate it with service contracts. The whole model (composite service plus contracts) should then be analyzed with a static verification tool. To achieve this, we expect to tune existing tools such as model checkers and theorem provers, for handling service compositions and their specifications (contracts). Crucial challenges of this tuning are how to model service interactions, finding the proper abstraction level to soundly model the external environment a composite service interacts with, and how to deal with quantitative aspects such as timeliness constraints.

Automatic code generation at deployment-time

Starting from a model described in the high-level language for describing service compositions, it should be possible to automatically generate an executable version of the composite service (e.g., a Java or WS-BPEL implementation), supporting multiple bindings (e.g., web services, JMS, EJB), and multiple deployment/hosting environments (e.g., J2EE container, OSGi). Moreover, an all-important feature is to support the automatic synthesis, from contracts, of helper modules (e.g., software monitors) for run-time validation. These transformations demand for developing automatic code generation techniques that support the different facets and constraints of multiple and heterogeneous target languages, bindings and deployment environments.

Run-time validation

Once a composite service is put into operation, run-time validation is the last required step to achieve the goal of continuous validation. Run-time validation techniques should take into consideration both how to embed run-time validation modules in the executing environment and how to perform validation, i.e., by deciding, for example, how to collect and analyze data, the degree of invasiveness and the timeliness in discovering undesirable situations.

4. SAVVY: A METHODOLOGY FOR LIFELONG VALIDATION OF SERVICE COMPOSITIONS

The research directions illustrated in the previous section lead us to propose SAVVY (Service Analysis, Verification and Validation methodologY), a methodology for lifelong validation of service compositions. SAVVY is still in a very preliminary phase of its development; therefore, here we describe our high-level, long-term vision of how SAVVY could fit within a complete development process. As detailed in Fig. 1, we associate a set of SAVVY-related tasks and artifacts to each step of the development process.

The first step of this SAVVY-aware development process is designing the composite service using a high-level abstract language for service compositions.

At design-time, it is assumed that external services are only known through their contracts, expressed in a suitable language that allows for describing interfaces, and functional and non-functional properties. These contracts are called *assumed contracts*, since they represent expectations of the quality of service and functionality the external services will provide. Based on the assumed contracts of all external services the composite service interacts with, the abstract model of the composition can be also annotated with *guaranteed contracts*, i.e., properties that the composite service should guarantee to its clients (step 2 in the figure).

The model of the composite service, including both assumed and guaranteed contracts, is then analyzed with a formal verification tool (step 3 in the figure), which checks that the composite service delivers its expected functionality and meets the required quality of service, under the assumption that the external services used in the composition fulfill their contracts. This design-time validation phase is continuously repeated, after adjusting the model, until the tool proves that the guaranteed contracts are satisfied.

Design-time verification does not prevent errors from occurring at run time. In fact, there is no guarantee that a service implementation eventually fulfills its contract. The service provider may either be malicious, by offering a service with an inferior experienced quality of service and/or a wrong functionality to increase its revenue on the service provision, or it might change the service implementation as part of its standard maintenance process: in this case, a service that worked properly might be changed in a new version that violates its previous contracts.

To solve this problem, SAVVY supports continuous validation by automatically generating (step 4 in the figure), at deployment time, software monitors that check for possible deviations from the correct behavior and QoS verified at design time. These monitors are automatically generated from the contract specifications, and they can be run on the environment/platform where the composite service is deployed. Also at deployment time, the abstract model of the composite service, used in the previous design and analysis phases, is transformed into a concrete implementation, runnable on the execution environment.

Finally, the composite service is put into operation (step 5 in the figure). In parallel to the running service, a run-time monitoring architecture checks the behavior and the QoS of the composite service, and it reports violations of service contracts to the service provider.

Although in the above description we referred to common steps of a traditional software development life cycle, we contend that SAVVY can be used within more specific, SOA-based life cycles like the *model-assemble-deploy-manage* one proposed in [21] by IBM. For example, steps 2 and 3 of our SAVVY-aware development process could be carried on during the *assemble* phase of IBM's SOA life cycle, while steps 4 and 5 could be part of, respectively, the *deploy* and

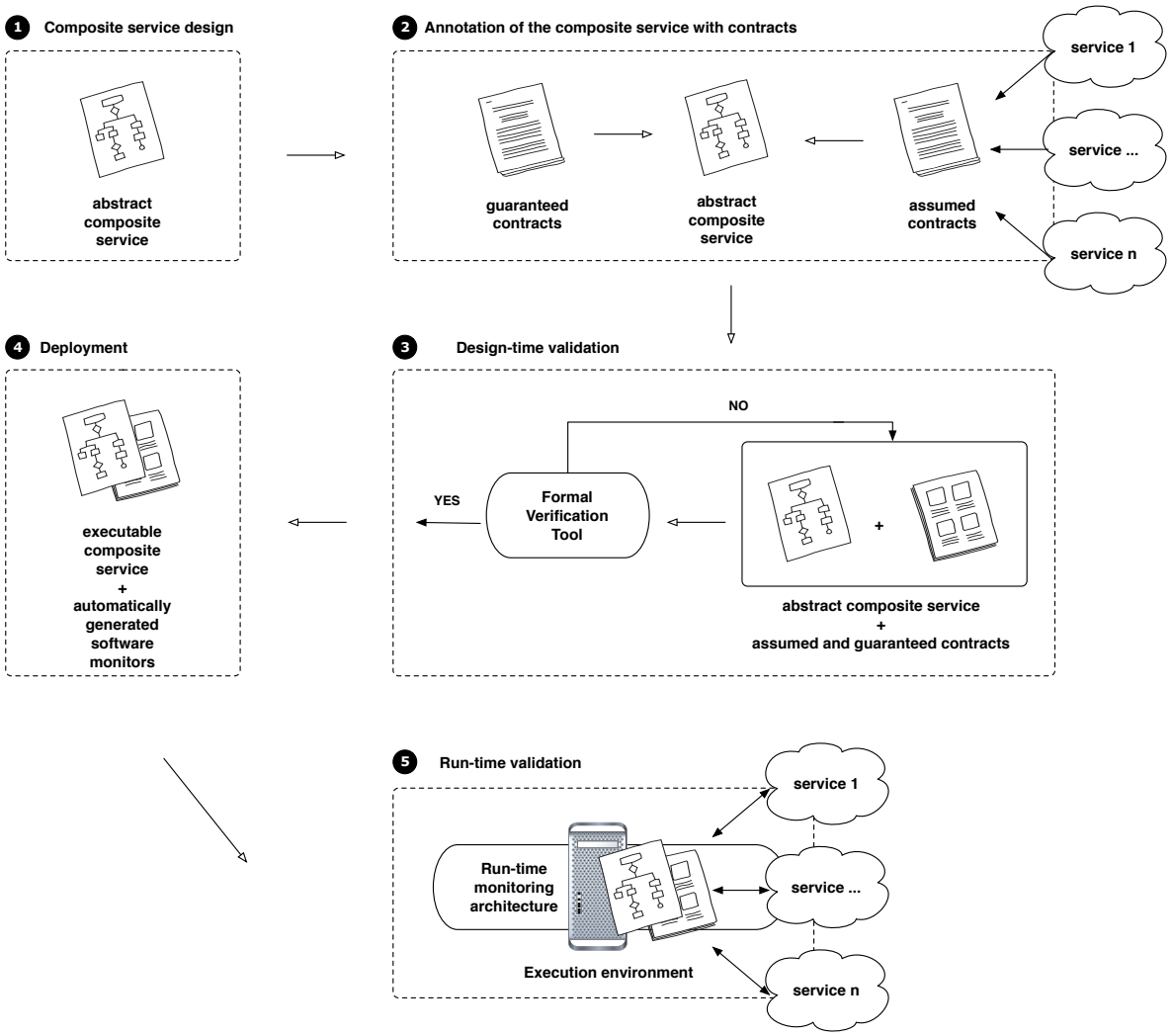

Figure 1: SAVVY-aware development process.

manage phases.

5. RELATED WORK

The research agenda discussed in this paper encompasses a broad range of research topics.

We believe that one of the most relevant contributions in the field of service composition languages is JOpera [30]. It is a visual language for composing, in a process-like style, heterogeneous types of services (such as web services, Java code snippets, shell scripts etc). JOpera is supported by a visual environment for describing control- and data-flow among services, and by a run-time architecture that optimizes the execution of the process. [38] evaluates the use of FuseJ —an architectural description language that tries to unify cross-cutting concerns and components— as a language for the composition of web services. However, being an architectural description language, FuseJ is only able to describe the structural facet of a service composition. [14] describes a BPEL-style solution for composing OSGi services on a semantically-enriched OSGi platform. In [39], semantic web service compositions are specified using a UML profile that allows for the automatic construction of OWL-S specifications from UML diagrams.

There are few approaches, like [33] and [3], which propose a lifelong validation framework for service compositions. Both consider only web service compositions described in BPEL; however they use different languages and techniques. In [33], they use the Event Calculus of Kowalski and Sergot [25] to model and reason about the set of events generated by the execution of a business process. At design-time they check the control flow of a process for livelocks and deadlocks, while at run-time they check if the sequence of generated events matches a desired behavior. [3] proposes a rich temporal logic language named ALBERT, which allows for specifying functional and non-functional properties both of the services a BPEL process interacts with and of the process itself, using an assume-guarantee approach. At design-time, model checking is used to prove that certain global properties of the workflow —expressed in ALBERT— are satisfied, assuming that the external services orchestrated by the workflow conform to their specifications, also expressed in ALBERT. At run-time, these properties are then veri-

fied using a monitoring architecture embedded in a BPEL engine. Actually, the work presented in [3], even though restricted to the realm of web services, represents a very preliminary step, at the incubator phase, of the methodology presented in this paper.

Many other approaches, on the other hand, tackle isolated aspects related to the main issue of engineering dependable service compositions. Design-time validation is addressed, for example, in [18], where the interaction between BPEL processes is modeled as a conversation and then verified using the SPIN model checker; in [17], design specifications (in the form of Message Sequence Charts) and implementations (in the form of BPEL processes) are translated into the Finite State Process notation and checked with the Labelled Transition System Analyzer. Other approaches focus on run-time validation of service compositions, considering either the behavior, as in [2, 27], or the non-functional aspects [28, 32, 35] (including industrial approaches such as Cremona [26], Colombo [12] and IBM Tivoli Composite Application Manager for SOAs [22]), or both [16]. Once again, all these approaches consider only web service compositions; we refer the reader to [4] for a comprehensive monograph on web services design- and run-time validation.

Specific techniques for performing run-time validation (e.g., how to specify and analyze finite traces of events or how to embed monitors in a run-time architecture) of generic software systems —therefore not explicitly related to the domain of service compositions— are the main topics of the Run-time Verification Workshops series [34]. Moreover, many techniques for run-time validation (e.g., MOP [10]) suggest the use of Aspect-Oriented Programming [24] as a mean to implement run-time monitoring architectures. Extensions to support AOP in service composition languages like BPEL has been proposed in [9] and in [7, 8].

Design- and run-time validation activities are related to the language that is used to specify the properties that are to be validated. Besides more traditional approaches based on assertion languages like WSCoL [6], or languages for defining service-level agreements (SLAs), such as WSLA [23], WS-Agreement [1] and SLAng [37], a third trend is based on languages for defining policies, such as ' WS-Policy [40]. It supports the definition of a set of assertions that describe how a web service should work, where an assertion is "an individual preference, requirement, capability or other property". An extension of WS-Policy, called WS-Policy4MASC, is defined in [16] to support monitor and adaptation of composite web services. Even though it is not specifically bound to a validation framework, the StPowla approach [19] aims at supporting policy-driven business modeling for SOAs. StPowla is a workflow-based approach that attaches to each task of a workflow modeling a business process, a policy that expresses functional and non-functional requirements and business constraints on the execution of the task.

6. SUMMARY

This position paper has analyzed some problem areas related to service compositions, like:

- Service Composition Facets
- Services in an open world
- Service Contracts
- Technologies for implementing SOAs
- Service Analysis and Validation

and it has presented a research road map to engineer dependable service compositions. This road map encompasses aspects like:

- The need for continuous, lifelong validation of service-based applications
- High-level languages for defining service compositions
- Specification languages for service contracts
- Design-time validation
- Automatic code generation at deployment time
- Run-time validation

One of the long-term goal of our research is to complete the development of SAVVY, a methodology for lifelong validation of dynamically evolvable software service compositions, which we have quickly sketched in the paper. SAVVY aims at integrating specification, analysis, verification and validation techniques, in a technology-independent manner, supported by a rich set of tools.

7. ACKNOWLEDGMENTS

The authors wish to thank Mehdi Jazayeri and Cesare Pautasso for their helpful comments on earlier versions of this paper.
This work has been partially supported by the IST EU project "PLASTIC" contract number 026955, and the Italian FIRB project "ART DECO".

8. REFERENCES

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). `http://www.ogf.org/documents/GFD.107.pdf`.

[2] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *ICWS '06 Proceedings*, pages 63–71, Washington, DC, USA, 2006. IEEE Computer Society.

[3] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET Software*, 1(6):219–232, 2007.

[4] L. Baresi and E. Di Nitto, editors. *Test and Analysis of Web Services*. Springer, 2007.

[5] L. Baresi, E. Di Nitto, and C. Ghezzi. Toward open-world software: Issue and challenges. *IEEE Computer*, 39(10):36–43, 2006.

[6] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *ICSOC 2005 Proceedings*, volume 3826 of *LNCS*, pages 269–282. Springer, 2005.

[7] M. Braem and N. Joncheere. Requirements for applying aspect-oriented techniques in web service composition languages. In *SC 2007 Proceedings*, volume 4829 of *LNCS*, pages 152–159. Springer, 2007.

[8] M. Braem, K. Verlaenen, N. Joncheere, W. Vanderperren, R. Van Der Straeten, E. Truyen, W. Joosen, and V. Jonckers. Isolating Process-Level Concerns Using Padus. In *BPM 2006 Proceedings*, volume 4102 of *LNCS*, pages 113–128. Springer, 2006.

[9] A. Charfi and M. Mezini. Aspect-oriented workflow languages. In *CoopIS 2006 Proceedings*, 2006.

[10] F. Chen and G. Roşu. MOP: an efficient and generic runtime verification framework. In *OOPSLA '07 Proceedings*, pages 569–588, New York, NY, USA, 2007. ACM Press.

[11] F. Curbera. Component contracts in service-oriented architectures. *IEEE Computer*, 40(11):74–80, 2007.

[12] F. Curbera, M. J. Duftler, R. Khalaf, W. A. Nagy, N. Mukhi, and S. Weerawarana. Colombo: lightweight middleware for service-oriented computing. *IBM Syst. J.*, 44(4):799–820, 2005.

[13] W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, 2001.

[14] R. P. Díaz Redondo, A. Fernández Vilas, M. Ramos Cabrer, J. J. Pazos Arias, J. García Duque, and A. Gil Solla. Enhancing residential gateways: a semantic OSGi platform. *IEEE Intelligent Systems*, 23(1):32–40, Jan/Feb 2008.

[15] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*, chapter 3. Prentice Hall, 2005.

[16] A. Erradi, P. Maheshwari, and V. Tosic. WS-Policy based monitoring of composite web services. In *ECOWS '07 Proceedings*, pages 99–108, Washington, DC, USA, 2007. IEEE Computer Society.

[17] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based Verification of Web Service Compositions. In *ASE 2003 Proceedings*, pages 152–163. IEEE Computer Society, 2003.

[18] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *WWW '04 Proceedings*, pages 621–630, New York, NY, USA, 2004. ACM Press.

[19] S. Gorton, C. Montangero, S. Reiff-Marganiec, and L. Semini. StPowla: SOA, Policies and Workflows. In *WESOA 2007*, 2007.

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

[21] R. High, S. Kinder, and S. Graham. IBM's SOA Foundation: An Architectural Introduction and Overview. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-whitepaper.pdf`, November 2005.

[22] IBM. Tivoli Composite Application Manager for SOA. `http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-soa/`.

[23] A. Keller and H. Ludwig. The WSLA framework: specifying and monitoring service level agreement for web services. *Journal of Network and System Management*, 11(1), 2003.

[24] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP'97 Proceedings*, volume 1241 of *LNCS*, pages 220–242. Springer, 1997.

[25] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.

[26] H. Ludwig, A. Dan, and R. Kearney. Cremona: an architecture and library for creation and monitoring of ws-agreents. In *ICSOC '04 Proceedings*, pages 65–74, New York, NY, USA, 2004. ACM Press.

[27] K. Mahbub and G. Spanoudakis. A framework for requirements monitoring of service based systems. In *ICSOC '04 Proceedings*, pages 84–93, New York, NY, USA, 2004. ACM Press.

[28] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In *WWW'08 Proceedings*, 2008. to appear.

[29] OASIS. Web Service Business Process Execution Language Version 2.0 Specification, april 2007. OASIS standard.

[30] C. Pautasso. *A Flexible system for visual service composition*. PhD thesis, ETH Zürich, 2004.

[31] PLASTIC Project. Description of Work. `http://www.ist-plastic.org`, 2005.

[32] F. Raimondi, J. Skene, L. Chen, and W. Emmerich. Efficient monitoring of web service SLAs. Technical Report RN/07/01, UCL, Dept. of Computer Science, Jan. 2007.

[33] M. Rouached, O. Perrin, and C. Godart. Towards formal verification of web service composition. In *BPM 2006 Proceedings*, volume 4102 of *LNCS*, pages 257–273. Springer, 2006.

[34] Run-time verification workshop series. `http://wwww.runtime-verification.org`.

[35] A. Sahai, V. Machiraju, M. Sayal, L. J. Jin, and F. Casati. Automated SLA monitoring for web services. In *DSOM 2002 Proceedings*, volume 2506 of *LNCS*, pages 28–41. Springer, 2002.

[36] SeCSE Project. Description of Work. `http://secse.eng.it/`, 2004.

[37] J. Skene, D. D. Lamanna, and W. Emmerich. Precise service level agreements. In *ICSE '04 Proceedings*, pages 179–188, Washington, DC, USA, 2004. IEEE Computer Society.

[38] D. Suvee, B. De Fraine, M. A. Cibran, B. Verheecke, N. Joncheere, and W. Vanderperren. Evaluating FuseJ as a web service composition language. In *ECOWS '05 Proceedings*, pages 25–37, Washington, DC, USA, 2005. IEEE Computer Society.

[39] J. T. E. Timm and G. C. Gannod. Specifying semantic web service compositions using UML and OCL. In *ICWS 2007 Proceedings*, pages 521–528. IEEE Computer Society, 2007.

[40] W3C Web Services Policy Working Group. WS-Policy 1.5. `http://www.w3.org/2002/ws/policy/`.