# Lifelong Verification of Dynamic Service Compositions*

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

## ABSTRACT

Service-oriented computing is an emerging paradigm that supports the development of distributed applications, which are realized by composing third-party services to provide new added-value services. These applications live in an open world, where new services can appear and disappear, and compositions may change dynamically. In this scenario, the traditional boundary between design time and run time is blurring. Verification, which traditionally pertains to design time, must now extend to run time. The goal of this thesis is thus to provide a holistic approach for lifelong verification of dynamic service compositions.

## Categories and Subject Descriptors

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

## General Terms

Reliability, Languages

## Keywords

Service composition, Specification language, Verification, Model Checking, Monitoring

## 1. INTRODUCTION

Service-oriented architectures (SOAs) have emerged as a promising solution to the problem of developing decentralized, distributed, and evolvable applications that live in an open world [5]. This architectural paradigm has been adopted in new and innovative computing domains, like ambient intelligence, context-aware applications and pervasive computing. Many current technologies, such as Web services, Jini and OSGi, have been associated with SOAs. In these architectures, *services* represent software components that provide specific functionality, exposed for possible use by many clients, who can dynamically discover and access them through network infrastructures.

*This work has been partially supported by the IST EU project "PLASTIC" - contract number 026955.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*FSE-16 Doctoral Symp.,* November 10, Atlanta, Georgia, USA
Copyright 2008 ACM 978-1-60558-378-5 ...$5.00.

This emerging scenario is highly dynamic, open, and decentralized; it is an instantiation of the *open-world software* scenario described in [5]. Services may evolve dynamically and autonomously. New services may be developed and published in registries, and then discovered dynamically by possible clients. Previously available services may disappear or become unavailable.

Thus, applications built as a service composition are made out of components that are and remain out of the control of the application itself. Moreover, service compositions make use of dynamic binding techniques to support continuous evolution and contextual adaptation. This implies that at design time the external services orchestrated in a service composition should only be known through their abstract interface, which provides a contractual specification, while their concrete identity may become known only later at run time, when bindings are resolved.

These characteristics have a severe impact on the notion of correctness and quality of the application and on the kind of guarantees that can be offered on the functional and non-functional properties of the resulting system.

To address this problem, in my thesis I will propose a continuous, lifelong verification approach to ensure that the required properties of a composite service are satisfied at design time but their validity is also maintained by the operational system at run time.

The key issue of continuous lifelong verification consists of complementing the design-time verification phase, which is usually carried on within the conventional development process and environment, with run-time verification techniques.

Design-time verification, indeed, does not prevent errors from occurring at run time. In fact, there is no guarantee that a service implementation eventually fulfills the contract promised through its interface. The service provider may dynamically modify the exported service in a malicious way, for example offering a lower-quality service than the one promised through the interface specification. Likewise, a service provider may inadvertently modify an existing service into an incorrect version during a maintenance activity. Thus a service that worked properly might be upgraded to a new version that violates its assumed interface, even though it may have passed successfully the testing phase at the provider's site. Furthermore, during design-time verification, it is difficult to model and simulate the behavior of the underlying distributed infrastructure, which plays an important role in the provision of networked services. Thus a service composition that at design time was proved to satisfy the requested quality of service (QoS) requirements may turn out in practice to violate them.

These examples motivate the need for extending the scope of verification to also cover run time. Once it is embedded into the service composition run-time environment, lifelong verification can provide the basis for advanced autonomic self-regulation capabilities to be added to SOAs. If a deviation is caught, suitable compensation policies and recovery actions should be activated. Moreover, data acquired during run-time verification can then be used for other purposes, e.g., to provide input to a reputation-based service ranking algorithm. Also, this information can be collected so that it provides precious feedback to fuel the next iterations of the service development cycle.

## 2. BACKGROUND AND RELATED WORK

My research area is software engineering for service-oriented systems. In particular, my focus is on techniques for specifying and analyzing dynamic service compositions. In this section, I will briefly survey existing work in this area.

Service compositions, besides being implemented by using traditional programming languages, are usually described by means of specific languages, which have been developed mainly for the Web services domain. These languages can be either orchestration, workflow-based, as BPEL [16], or choreography-based like WS-CDL [22]. A technology-neutral language is JOpera [17], which is a visual language for composing, in a model-driven, process-like style, heterogeneous types of services.

Model checking has been proposed by several researchers for verifying safety and liveness properties of Web service compositions, in conjunction with different formalisms (e.g., Büchi automata, process algebras, Petri nets). For example, in [13], the interaction between BPEL processes is modeled as a conversation and then properties expressed in LTL (Linear Temporal Logic) are verified by using the SPIN model checker. In [12], 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. [21] uses Petri Nets to define the semantics of BPEL; validation is then performed by using the LoLA model checking tool.

Several works define specification languages for functional and non-functional properties (often expressed in the form of an SLA, Service Level Agreement) and propose an associated monitoring architecture. There are approaches that focus only on monitoring the behavior of a service composition, which is described with assertion languages like RTML [2] and WSCoL [6]. Other approaches consider non-functional aspects, either described in terms of basic QoS attributes (e.g., response time or throughput) as in [15], or by means of a formal language as WSLA [20], SLAng [18] and WS-Policy [11].

All these approaches focus on analysis and verification techniques for a specific life cycle phase (i.e., either design time or run time). Furthermore, they take into account service compositions implemented with a specific technology (i.e., Web services). Moreover, many of the approaches abstract from data analysis, i.e., they do not consider the content of the messages exchanged by a composite service, even if, in principle, the behavior of a service could be affected by these data (e.g., in context-aware applications).

An initial attempt to integrate design-time and run-time verification for Web service compositions has been presented in [19]. The approach is based on the Event Calculus, which is used to model and reason about the set of events generated by the execution of a business process. At design time the control flow of a process is checked for livelocks and deadlocks, while at run time it is analyzed to check if the sequence of generated events matches a certain desired behavior. The main weaknesses are the lack of support for data-aware properties and the limited expressiveness of the supported temporal properties.

## 3. PROPOSED APPROACH

As outlined in the introduction, the goal of this thesis is to provide a methodology and the accompanying tools to engineer dependable service compositions, by means of a continuous, lifelong verification approach. The underlying hypothesis is that in *open-world software*, like SOAs, the dynamic characteristics of the components (e.g., services) and of the environment may invalidate the verification results obtained at design time, making thus necessary to perform additional verification activities at run time.

The main requirements of the proposed approach are to be model-driven and technology-neutral. The former allows for defining service compositions and their specifications in term of abstract models that can be subsequently transformed both into formal descriptions suitable for the verification and into representations close to the implementation. The latter requirement ensures that the verification techniques that will be developed in this thesis remain applicable to future developments of the service-oriented computing paradigm, which is a rapidly changing area.

The approach assumes that the external services which a composite service interacts with are known through their specifications, which may include both functional and non-functional properties. Moreover, it leverages existing work on techniques and languages for designing service compositions, by assuming that composite services are described by means of an appropriate modeling language.

The contribution of this thesis will be threefold:

- A formal language for specifying functional and non-functional properties of a service composition and of its component services.

- A design-time verification technique for checking the conformance of a composite service with respect to its specifications, assuming that the external services used by the service composition behave according to their specifications.

- A run-time verification technique for monitoring the execution of a composite service and of the services it interacts with, to check for possible violations of service specifications.

In the rest of this section, I will describe the research challenges related to the key contributions of my thesis.

### *Specification Language for Service Compositions*

Service specifications play a very important role in the realm of SOAs. In fact, services are developed by independent parties and are exposed as black boxes that can only be invoked by clients. Their specification is the only information about them available to clients, since their implementation is normally not accessible. A well-designed language is thus required to capture and constrain the properties that a service composition and its component services should guarantee.

The expressiveness of the language is an important quality attribute since specifications may describe different aspects of a service (e.g., syntactic interface, structure, behavior, quality of service). Thus the language will support the specification of functional and non-functional properties, such as assertions on the service state and data values, properties on historical traces of events (triggered by internal state changes but also external interactions) and QoS attributes like response time, throughput, and reliability, possibly expressed in terms of probabilistic models. The language will also support the specification of both stateless and stateful interactions. In addition, the language will be aware of the constructs defined in the language adopted for designing abstract models of service compositions, to enable associating specifications with specific scopes and execution points of a composite service. Furthermore, when designing the language, it will be very important to find the right balance between expressiveness and usability.

### *Design-time Verification*

The approach will promote the annotation of abstract service composition models with specifications expressed in the aforementioned language. These specifications may represent both expectations of the quality of service and functionality the external services will provide, as well as properties that the composite service should guarantee to its clients.

The annotated model will then be transformed into several formal representations, corresponding to different models of computation, (e.g., different kinds of automata) suitable for static verification. The analysis will check 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 specifications. In particular, the specifications of external services will be transformed into abstractions of the corresponding services, which will be used to refine the model of the composite service under verification.

The analysis tool (typically, a model checker) will be realized by incorporating existing verification techniques and their related tool implementations. This will require to address challenges such as: modeling the exchanged data and non-functional properties of both stateless and stateful service interactions, finding the proper abstraction level to describe soundly the external environment a composite service interacts with, and dealing with quantitative aspects such as timeliness constraints.

### *Run-time Verification*

After successful completion of design-time verification, the abstract model of a service composition can be transformed into an executable version, which may also be bound to concrete service implementations. This model transformation step will also include the automatic synthesis, from service specifications, of platform-specific helper modules for run-time verification, such as software monitors, message interceptors, and failure detectors. Such transformations will involve the development of automatic code generation techniques that address the constraints of heterogeneous execution platforms.

Once in operation, the run-time verification modules will check both the service composition and its component services, for possible deviations from the correct behavior and expected QoS as they were verified at design time.

The verification techniques that will be implemented in the run-time modules will be designed by taking into consideration how to transparently embed the modules in the execution environment with minimal overhead. In this direction, essential design decisions will concern how to collect and analyze large amounts of data, the degree of invasiveness of the instrumentation, and the timeliness in discovering undesirable situations.

## 4. PROGRESS TO DATE

Preliminary investigation has been conducted by focusing on the verification of Web service-based service compositions described in BPEL.

Static analysis of BPEL processes by means of model checking has been proposed in [9], which presents a translator from BPEL to the input language of the Bogor model checker. This tool supports the verification of data-aware properties written in WSCoL, and temporal properties expressed in LTL.

Our subsequent research steps focused on specification languages for formally describing functional and —partially— also non-functional properties of service compositions. We first explored in [7] the use of algebraic specifications for describing the behavior of stateful conversational services and monitoring their execution. Afterwards, we enriched WSCoL with time-related operators, leading to the definition of Timed WSCoL [3]. The WSCoL family of languages has then been the basis for the definition of a new language, ALBERT (Assertion Language for BPEL Process Interactions) [4]. ALBERT aims at being an expressive specification language for defining both functional and non-functional properties of a BPEL process. It is also supported by several prototype tools used for design-time verification (by means of model checking) and run-time verification (by means of dynamic monitoring).

ALBERT and its design- and run-time verification tools are the main constituents of a a methodology, called SAVVY-WS [10], which supports lifelong verification of Web service-based service compositions. An outlook on the adoption of this methodology in the context of generic service compositions has been proposed in [8].

Current work addresses the following issues. First of all, to meet the requirements of designing a model-driven and technology-neutral approach, I seek a language for defining service compositions that fulfill these prerequisites, and on which the whole approach could be based. JOpera seems to be a promising candidate, as it is also supported by a comprehensive modeling environment and an execution engine, on the top of which the proposed verification techniques could be implemented.

As for the specification language, my research follows two directions, whose starting point is represented by ALBERT. One direction is about revising and enhancing the language, to improve its expressiveness. I will consider the introduction of new language constructs, such as operators to describe probabilistic properties, and to support variable-binding mechanisms like the freeze quantifier [1]. I will also evaluate how the new language constructs affect the techniques adopted for design-time and run-time verification. The other direction is about modifying the specification language to make the whole approach technology-neutral. In this sense, I plan to replace some BPEL-specific constructs of ALBERT with the ones required by the aforementioned modeling language.

Concerning design-time verification, my main focus is on two issues related to model checking, i.e., the support for enhanced handling of time-related and probabilistic properties, and for compositional verification.

With respect to run-time verification techniques, current work concentrates on the efficient monitoring of temporal properties, by exploiting the correlation between temporal logic and alternating automata.

## 5. EVALUATION

The assessment of the outcome of this thesis will be conducted by applying the proposed approach to several scenarios, that are illustrative of the various type of service compositions and of their expected functional and non-functional properties. These scenarios will be derived both from literature and from collaborations with other research institutes, both academic and industrial, within the context of several research projects currently carried on by our research group. To assess that the approach is technology-neutral, I will experiment with different SOAs implementations, such as the Web services stack and OSGi.

The evaluation of the three main contributions of the thesis will be performed as follows. The requirements description and the specifications of the scenarios will be used to check if the specification language is expressive enough to formalize the properties that characterize real-world, service-based applications. These properties will also be the basis for defining a collection of patterns of service interaction specifications, as already done for other domains like embedded systems [14]. As for the design-time verification technique, I will check if the chosen abstraction methods make the verification scalable and are suitable for an incremental approach. Finally, the main aspect of the run-time verification technique that will be evaluated is the overhead (in terms of space and time) it adds to the normal execution profile of a composite service.

## Acknowledgments

The author wishes to thank Carlo Ghezzi and Cesare Pautasso for their comments on earlier versions of the paper.

## 6. REFERENCES

[1] R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–203, 1994.

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

[3] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. A timed extension of WSCoL. In *ICWS 2007 Proceedings*, pages 663–670. IEEE, 2007.

[4] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET Softw.*, 1(6):219–232, 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] D. Bianculli and C. Ghezzi. Monitoring conversational web services. In *IW-SOSWE'07 Proceedings*, pages 15–21. ACM, 2007.

[8] D. Bianculli and C. Ghezzi. Towards a methodology for lifelong validation of service compositions. In *SDSOA 2008 Proceedings*, pages 7–12. ACM, 2008.

[9] D. Bianculli, C. Ghezzi, and P. Spoletini. A model checking approach to verify BPEL4WS workflows. In *SOCA 2007 Proceedings*, pages 13–20. IEEE, 2007.

[10] D. Bianculli, C. Ghezzi, P. Spoletini, L. Baresi, and S. Guinea. A guided tour through SAVVY-WS: a methodology for specifying and validating web service compositions. In E. Börger and A. Cisternino, editors, *Software Engineering*, volume 5316 of *LNCS*, pages 131–160. Springer, 2008.

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

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

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

[14] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *ICSE '05 Proceedings*, pages 372–381. ACM, 2005.

[15] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In *WWW'08 Proceedings*, pages 815–824. ACM, 2008.

[16] OASIS. Web Service Business Process Execution Language Version 2.0 Specification, 2007.

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

[18] F. Raimondi, J. Skene, , and W. Emmerich. Efficient online monitoring of web-service SLAs. In *SIGSOFT 2008 - FSE 16 Proceedings*. ACM, 2008. to appear.

[19] 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.

[20] 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.

[21] B.-H. Schlingloff, A. Martens, and K. Schmidt. Modeling and Model Checking Web Services. *ENTCS*, 126:3–26, 2005.

[22] W3C. Web services choreography description language. `http://www.w3.org/TR/ws-cdl-10/`, 2005.