Embedding Continuous Lifelong Verification in Service Life Cycles*

Domenico Bianculli
University of Lugano
Faculty of Informatics
Lugano, Switzerland
domenico.bianculli@lu.unisi.ch

Carlo Ghezzi
Politecnico di Milano
DEEP-SE group - DEI
Milano, Italy
carlo.ghezzi@polimi.it

Cesare Pautasso
University of Lugano
Faculty of Informatics
Lugano, Switzerland
cesare.pautasso@unisi.ch

Abstract

Service-oriented systems are an instantiation of open world software, which is characterized by high dynamism and decentralization. These properties strongly impact on how service-oriented systems are engineered, built, and operated, as well as verified. To address the challenges of applying verification to open service-oriented systems, in this position paper we propose to apply verification across the entire life cycle of a service and introduce a verification-oriented service life cycle.

1. Introduction

One of the defining properties of Software Service Engineering, as opposed to traditional Software Engineering, is the *open world* assumption [3]. The implications of this assumption affect all aspects of this emerging discipline, e.g., from how to design a service-oriented architecture to the notion of correctness and quality that can be applied to define and check the integrity and the validity of a system design. In this position paper we focus on the consequences of this assumption on the way verification is carried out in the service life cycle.

In particular, we argue for the need for embedding continuous lifelong verification across the entire service life cycle. Thus, it is no longer sufficient to apply verification during a particular phase or using a specific technique, such as proving properties based on a service contract at design time, or testing a service at deployment time to ensure and guarantee that it will continue to work as it has been specified or tested during the remainder of its lifetime. Since services live in an open world, where change is frequent, unexpected, and welcome [13], it becomes important to be able to assert properties that have a *lifelong* validity: both at design time, deployment time, and at run time. *Continuous* verification is about checking services as they are put into production, but also advocates that they are monitored (and checked) during their entire productive life.

This paper shows how different verification techniques [2] can be applied at different stages of the service life cycle, by proposing to enhance conventional life cycle models with a verification-oriented life cycle layer. This verification-oriented life cycle iteratively integrates and correlates different techniques, and makes possible to guarantee that the same properties verified based on services contract specifications, will still be checked once the service becomes an executable artifact and it is embedded into a larger service-oriented architecture. This way, continuous lifelong verification can provide a fundamental building block for delivering self-adaptive systems [8].

The rest of this paper is organized as follows. Section 2 briefly surveys the existing proposals for service life cycles and discusses to which extent they support verification, leading to Section 3, which motivates the need for continuous lifelong verification of service-based applications. Section 4 presents our verification-oriented model of service life cycle, which shows how to achieve continuous lifelong verification. Section 5 discusses related work and Section 6 concludes the paper with an outlook on possible future research directions.

2. Life cycle models

In this section we discuss how service verification is embedded in the software service life cycle models proposed by several leading SOA vendors and SOC researchers.

Some authors (e.g., [10]) suggest to comply with a traditional life cycle, following a sequence of phases such as *analysis*, *design*, *development*, *testing*, *deployment* and *administration*. This model has a dedicated phase for (functional) testing, which strictly follows the development phase and precedes deployment. Hints are also given about the

*This work has been partially supported by the EU-FP7-215483 project "S-Cube" and by the ERC grant 227977 "SMScom".

possibility to monitor the service usage during the administration phase, even though the topic is not further discussed.

In other cases, the life cycle model is adapted to the specific characteristics of service-centric software systems, as proposed by [7]. This is the case for life cycle models proposed by industrial vendors. IBM, for example, proposes a *model*, *assembly*, *deploy* and *manage* cycle, in which the last phase is also devoted to monitoring the performance of a service and detecting the failure of system components. Sun presents the SOA Repeatable Quality methodology, which includes the *conception*, *inception*, *elaboration*, *construction* and *transition* phases in an iterative way. However, verification activities are not explicitly mentioned in the documentation. Oracle/BEA proposes a life cycle model that clearly separates design-time activities (i.e., *identify business process*, *service modeling*, *build and compose*) from those carried out at run time (i.e., *publish and provision*, *integrate and deploy*, *secure and manage*, *evaluate*). In this model, verification activities belong to the *secure and manage* phase, and are mainly focused on SLA management, performance optimization and dealing with error events.

As for academic contributions, [14] illustrates a (Web) service life cycle model and a service-oriented design methodology. The life cycle starts with an initial *planning* phase, followed by a set of phases to be iteratively repeated: *analysis and design*, *construction and testing*, *provisioning*, *deployment*, *execution and monitoring*. Verification is performed before services are put into operation, by means of functional, performance, interface and assembly testing, and when services becomes operational, by means of QoS monitoring techniques. [11] proposes a stakeholder-driven life cycle, with much emphasis on the assignment of activities to stakeholders and on the interaction between and across them. The service provider is responsible for service functional testing at design time and service monitoring at run time. From the point of view of service consumers, verification activities include application testing at design time, in case the service consumer plays also the role of an application provider/service integrator, and — at run time — monitoring of the services that are consumed.

3. Motivation

As shown above, existing proposals of service life cycles advocate to perform verification either at a specific stage of the life cycle, e.g., at design time, before putting a service into operation, or at execution time, while the service is being provisioned. In some cases, verification is performed at both stages, but usually the properties verified at one stage are different from and not related to the ones verified at the other stage, e.g., by following the classical dichotomy between functional and non-functional properties.

Such narrow scope of verification does not entirely address the implications of the open world assumption on how verification activities are performed. Indeed, the key issue of continuous lifelong verification consists of spanning verification activities across the service life cycle, which can be iterated multiple times. We motivate the need for applying continuous verification during the entire life span of a service by means of the following three statements.

Design-time verification only gives limited guarantees. This kind of verification is carried out by assuming some properties about and using some models of the environment with which the service will interact. The environment is represented mostly by 3rd-party services and the distributed network infrastructure. As for the former, there is no guarantee that a service provider will eventually fulfill the obligations promised in a service agreement. For example, during a standard maintenance activity, a provider could inadvertently modify an existing service into an upgraded but incorrect and/or incompatible version, which could break the compatibility with service clients. Additionally, a malicious provider could modify the exported service, by offering a lower-quality service than the one promised through the agreement. Regarding the latter, the parameters estimations used to model the network infrastructure are often inaccurate, since they must be provided *a-priori* by domain experts and are related to quantities that may change over time. Thus a service that before deployment was proved to satisfy the requested quality of service requirements may turn out in practice to violate them, because of the mismatch between the abstract models that were used for verification before the deployment and the actual state of the environment at run time.

A service lives also between design and execution. A service life cycle contains other stages besides design and execution. Thus a coherent lifelong verification methodology should indicate the use of specific techniques at every stage of the life cycle. For example, when a service is about to be deployed, it should be "auditioned"' [5], i.e., it should be tested during the interaction with the actual services that will be provided by business partners, before exposing the service to public usage.

Execution-time verification can close the loop of iterative service life cycles. Some verification activities may operate on live data acquired by means of run-time monitoring, but it may not always be practical to perform them on-the-fly. Therefore such data can be used to fuel the next iterations of the service life cycle, by providing valuable feedback to the service architects, modelers, and developers. Live data can be exploited for a thorough audit, an accurate analysis and model calibration before deploying a new version of the service.

4. Verification-oriented life cycle

Since many of the life cycle models reviewed in Section 2 have a coarse-grained modularity in terms of activities [11], we ground our verification-oriented life cycle on a slightly modified version of the model described in [14]. This model has an intrinsic iterative structure of the life cycle, which is a prerequisite to achieve continuous lifelong verification. As we are going to show, it is thanks to the feedback provided by the verification activities that the loop in the iterative life cycle can be closed. With respect to the original formulation of the model, we shifted the *testing* and *monitoring* phases into the verification-oriented layer of the model.

The complete model is shown in Figure 1: non verification-related phases of the life cycle are depicted with a white-filled shape, while the corresponding verification activities are highlighted using a grey-filled shape.

The *analysis and design* phase identifies the requirements of the service-based application, builds the models of the business processes defining the applications and specifies the required services. We envision that at this stage, the requirements are captured and automatically formalized, such that their formal models will be made available across the entire lifespan of the service. This is a crucial step, since the formalized requirements represent the properties for which we want to assert their lifelong validity. A relevant contribution to this step is embodied by a modeling/specification language that should be able to capture all facets of the behavior and of the QoS related to services execution and interaction. The first step of continuous lifelong verification is to ascertain that the properties (both functional and non-functional) corresponding to requirements are met by abstract design models of the service, evaluated in a given context. To achieve this, we suggest the use of *static analysis* techniques such as model checking. The goal is to check if a formal description of a certain property holds in the service model by transforming it to a suitable representation that can be efficiently verified.

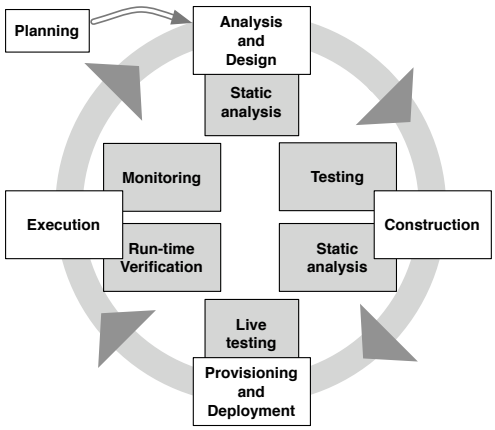

Figure 1. Verification-oriented life cycle

The *construction* phase is about developing the actual implementation of the service-based application. In this phase, we recommend to use *static analysis* techniques to verify the correctness of the application, as well as *testing* techniques that operate directly on the implementation. With respect to the previous phase, the main difference lies in the system under verification: in this phase, verification is performed against the concrete implementation of the service. Therefore, the verification techniques require more tuning; e.g., source-level analysis tools should be used. However, many of the verification artifacts used in the previous phase (e.g., temporal logic formulae or queueing network models) can still be used in this phase. To do so, the previously mentioned verification techniques can be used in combination: e.g., the counter-examples obtained from the execution of model checking can be used to derive dedicated test cases. Moreover, the properties captured during the analysis phase can be converted, using specific model-transformation techniques, into proper artifacts suitable for source-level verification and testing.

The *provisioning* phase is about making strategic decisions on service governance, certification, metering and billing, while the *deployment* phase is concerned about making the service publicly available. During these phases we claim that pre-deployment live testing, i.e., testing the interactions of a service with other actual services, can be a supporting tool for ensuring the validity of governance decisions, such as the choice of business partners to interact with, or the kind of service level agreement to be signed. Moreover, testing (both functional and non-functional) with the actual services may also reveal some problems that were not detected in the previous phase because of the abstract models that were used. Also in this phase, the original verification artifacts corresponding to requirements can be transformed into properties that can checked and/or measured during the live testing.

Finally, the *execution* phase is about the productive part of the cycle, when the service is kept in operation. In this phase, the main verification activities are monitoring and run-time verification. The former collects and analyzes data about the quality of the provisioned service and the 3rd-party services it interacts with, while the latter analyzes the execution trace to detect possible violations of the initial requirements. Indeed, the models corresponding to these requirements, are transformed, during the deployment, into data collectors (e.g., to monitor QoS properties such as response time and throughput) and failure detectors (e.g., to detect a violation of a functional or temporal assertion). One of the key aspects of continuous lifelong verification is that run-time verification activities may support model

calibration, by using the collected data to provide a better estimation of the parameters that define the external operating environment [9], and *post-mortem* analysis, which is usually performed off-line by means of a static analysis tool. The information collected by these verification activities is also fed into the next iteration of the cycle providing valuable input to the *analysis and design* phase.

5. Related work

Section 2 has already described the different service life cycle models and how they deal with the issue of verification. However, there have been some other proposals about the verification of service-based applications, which do not strictly fall under the umbrella of service life cycles. For example, in previous work [6] some of the authors addressed the issue of lifelong verification of service compositions and proposed a methodology to deal with this issue. However, this proposal considered a simplified model of a very generic service life cycle, which only distinguished between a design-time phase and a run-time phase. Agile methods like Test-driven development [4] (TDD) emphasize the role of continuous testing during the development process. In principle, this is similar to our approach towards continuous verification; however they focus only on the specific implementation/development phase of the life cycle. ASOP [15] is a proposal for an agile service-oriented (development) process; however, as far as verification is concerned, it only considers TDD-like verification techniques.

The methodology followed by this work to create a service life cycle tailored to specific viewpoints has been introduced by [13], where a change-oriented service life cycle has been proposed in the context of service evolution featuring support for: (re-)configuration, alignment, and control of services upon changes. A similar approach — even though not specific to service-oriented systems engineering — can be found in [12], where a waterfall software development life cycle has been extended to include security into every phase of the life cycle.

6. Conclusion and future work

Service-oriented systems live in an open world that requires new engineering methodologies to deal with the intrinsic dynamic and decentralized nature of these systems. This paper has focused on verification, by showing how existing service life cycle models are inadequate to support continuous lifelong verification of service-based applications — which we believe to be a key aspect of this kind of applications — and it has proposed a verification-oriented life cycle to achieve this goal.

Although in this paper we have grounded our verification-oriented life cycle on a specific life cycle model, in the future we will consider other existing life cycle models and try to overlay the proposed verification-oriented life cycle on them. In particular, we will investigate the challenges to adapt the verification-oriented life cycle layer to agile development processes, since our proposal augments an iterative model. In all cases, we will evaluate its adoption in the context of real service-based applications development, by measuring the impact on the project quality, duration and cost. Moreover, we will investigate the role and the use of models [1] in the context of the proposed life cycle, to define a model-driven methodology to achieve continuous lifelong verification.

References

[1] D. Ardagna, C. Ghezzi, and R. Mirandola. Rethinking the use of models in software architecture. In *Proc. of QoSA 2008*, volume 5281 of *LNCS*, pages 1–27. Springer, 2008.

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

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

[4] K. Beck. *Test Driven Development by Example*. Addison-Wesley Professional, November 2002.

[5] A. Bertolino, L. Frantzen, and A. Polini. Audition of web services for testing conformance to open specified protocols. In *Architecting Systems with Trustworthy Components*, volume 3938 of *LNCS*, pages 1–25. Springer, 2006.

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

[7] M. B. Blake. Decomposing composition: Service-oriented software engineers. *IEEE Software*, 24(6):68–67, June 2007.

[8] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 15(3):313–341, 2008.

[9] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburelli. Model evolution by run-time adaptation. In *Proc. of ICSE'09*. IEEE Computer Society, 2009. to appear.

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

[11] Q. Gu and P. Lago. A stakeholder-driven service life cycle model for SOA. In *Proc. of IW-SOSWE'07*, pages 1–7, 2007.

[12] A. M. Hoole, I. Simplot-Ryl, and I. Traore. Integrating contract-based security monitors in the software development life cycle. In *Proc. of FLACOS'08*, pages 25–30, 2008.

[13] M. P. Papazoglou. The challenges of service evolution. In *Proc. of CAiSE 2008*, volume 5074 of *LNCS*, pages 1–15. Springer, 2008.

[14] M. P. Papazoglou and W. V. D. Heuvel. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, year 2006.

[15] A. Qumer and B. Henderson-Seller. ASOP: An agile service-oriented process. In *New Trends in Software Methodologies, Tools and Techniques*, pages 83–92. IOS Press, 2007.