SeSAME: Modeling and Analyzing High-Quality Service Compositions

Svetlana Arifulina, Sven Walther  
Department of Computer Science  
University of Paderborn  
[s.arifulina | sven.walther]@upb.de

Matthias Becker, Marie C. Platenius  
Heinz Nixdorf Institute  
University of Paderborn  
[matthias.becker | m.platenius]@upb.de

ABSTRACT

Today, software components are traded on markets in form of services. These services can be service compositions consisting of several services. If a software architect wants to provide such a service composition in the market for trade, she needs to perform several tasks: she needs to model the composition, to discover existing services to be part of that composition, and to analyze the composition’s functional correctness as well as its quality, e.g., performance. Up to now, the architect needed to find and use different tools for these tasks. Typically, these tools are not interoperable with each other. We provide the tool SeSAME that supports a software architect in all of these tasks. SeSAME is an integrated Eclipse-based tool-suite providing a comprehensive service specification language to model service compositions and existing services. Furthermore, it includes modules for service matching, functional analysis, and non-functional analysis. SeSAME is the first tool that integrates all these tasks into one tool-suite and, thereby, provides holistic support for trading software services. Thus, it contributes to the acceptance and success of a service market.

Categories and Subject Descriptors
D.2.11 [Software Engineering]: Software Architectures - Languages; I.6.4 [Simulation and Modeling]: Model Validation and Analysis

Keywords
Service Oriented Computing, Service Specification, Service Matching, Verification, Non-functional Analysis

1. INTRODUCTION

Today, software components are traded on markets in form of services. These services can be service compositions consisting of several services. If a software architect wants to provide such a service composition in the market for trade, she needs to perform several tasks: she needs to model the composition, to discover existing services to be part of that composition, and to analyze the composition’s functional correctness as well as its quality, e.g., performance. Up to now, the architect needed to find and use different tools for these tasks. Typically, these tools are not interoperable with each other. We provide the tool SeSAME that supports a software architect in all of these tasks. SeSAME is an integrated Eclipse-based tool-suite providing a comprehensive service specification language to model service compositions and existing services. Furthermore, it includes modules for service matching, functional analysis, and non-functional analysis. SeSAME is the first tool that integrates all these tasks into one tool-suite and, thereby, provides holistic support for trading software services. Thus, it contributes to the acceptance and success of a service market.

Today, software components are traded on markets in form of services. These services can be service compositions consisting of several services. If a software architect wants to provide such a service composition in the market for trade, she needs to perform several tasks: she needs to model the composition, to discover existing services to be part of that composition, and to analyze the composition’s functional correctness as well as its quality, e.g., performance. Up to now, the architect needed to find and use different tools for these tasks. Typically, these tools are not interoperable with each other. We provide the tool SeSAME that supports a software architect in all of these tasks. SeSAME is an integrated Eclipse-based tool-suite providing a comprehensive service specification language to model service compositions and existing services. Furthermore, it includes modules for service matching, functional analysis, and non-functional analysis. SeSAME is the first tool that integrates all these tasks into one tool-suite and, thereby, provides holistic support for trading software services. Thus, it contributes to the acceptance and success of a service market.

1. INTRODUCTION

Technical advances like cloud computing and world-wide computer networks enable the cost-efficient development and adaptation of flexibly composable software components. These components are traded in terms of services on markets like Google APIs Explorer [5]. Software providers can request a service to use it as part of a service composition and, again, provide the service composition on a market for other users. For example, consider a software architect who wants to sell a trip planning service. This service should be able to create a round trip in a given city using the local traffic system. For this purpose, she models a composition of a map service and a traffic system’s service. However, in order to be able to provide a well-sold, high-quality service composition to the market, she needs to (1) find existing services that satisfy her requirements and can, thereby, be reused in her composition. Furthermore, she needs to (2) find out whether the final composition is good wrt. (2a) functional correctness and (2b) quality, e.g., performance. SeSAME (Service Specification, Analysis, and Matching Environment) is a tool-suite that supports the software architect in describing her composition by providing a comprehensive service specification language. Further integrated modules provide support for different modeling tasks:

- a modeling environment for compositions allows the architect to specify the composition in terms of high-level control and data flow,
- a modeling environment for existing services allows to specify functional and non-functional properties of existing services offered on a market to be discovered,
- a configurable matching process determines whether services are suitable as parts of the composition,
- a verifier analyzes the functional correctness of the composition’s protocols and pre- and postconditions,
- a quality analysis analyzes the performance and scalability of the composition using prediction techniques.

Up to now, existing tools cover only parts of this functionality. For example, there are tools providing matching approaches, supporting composition design, or conducting analysis. However, using them together is problematic because they often lack interoperability, e.g., they require models having different formats and expressiveness. The contribution of SeSAME is the integration of multiple kinds of collaborating specification and analysis components. This integration significantly reduces the architect’s effort in trading high-quality service compositions in a service market.

In the next section, we provide an overview of the functionality realized in SeSAME. In Section 3, we present the most important technical information about the realization of SeSAME. Section 4 briefly summarizes the most related existing work. Finally, we draw conclusions in Section 5.
2. OVERVIEW OF THE FUNCTIONALITY

In this section, we give an overview of the functionality provided by SeSAME including service specification, design of service compositions, service matching, and analysis.

2.1 Service Specification

SeSAME provides the possibility to comprehensively specify software services. Service specifications created with SeSAME serve as a basis for the tasks of matching and analysis. Therefore, comprehensive specifications are needed for these tasks to produce high-quality results.

SeSAME comprehensive specifications can be black-box or white-box. Black-box specifications are usually created by service providers, who specify the properties of their services without exposing their implementation details. In Fig. 1, the provided interfaces of the services Map Service and Traffic System Service have such black-box specifications attached.

White-box specifications are usually created by software architects. They design a service composition and, thus, specify its internal structure including the corresponding control and data flow. In Fig. 1, the specification of the Orchestrator in the service TripPlanner with two required interfaces is a part of such a white-box specification of a composition. For both specifications, SeSAME supports the specification of functional and non-functional properties specified in Sections 2.1.1 and 2.1.2. Additionally, in the white-box specification of services, their control flow is modeled by a so-called service effect specification (SEFF) [3]. SEFFs have a notation similar to activity diagrams, and are exploited to analyze the functional correctness of compositions.

2.1.1 Functional Properties of Services

Using SeSAME, a service provider can specify the functionality of her services using interfaces and operation signatures. For an example, refer to the signature of the operation getBusStation in Fig. 1. Operation signatures use ontological data types from a domain ontology for their parameters. A domain ontology represents the domain knowledge in terms of domain concepts and their relations. For example, the domain ontology represents the domain knowledge in terms of data types from a domain ontology for their parameters. A domain ontology serves as a basis for the tasks of matching and analysis. Therefore, comprehensive specifications are needed for these tasks to produce high-quality results.

SeSAME provides the means to specify performance, privacy, reputation, and price for services. These specifications allow (1) to check whether the service level agreements of two interfaces comply and (2) to analyze whether a service holds the specified quality properties.

SeSAME models services in a way that performance for a service operation or a service interface can be predicted. For performance predictions, firstly, internal resource demands within an operation are specified. For example, the Map Service requires 30 HDD units to execute the operation getBusStation. Secondly, the resource environment where the service will be executed and a usage profile are specified (not shown in our example). Results of the performance prediction are modeled as a mean response time in the service specification. For example, the provided interface of the Orchestrator claims its response time to be less than 5 sec.

SeSAME privacy allows service providers to specify how requesters’ data must be handled by provided services. In the privacy specification, a privacy policy including restrictions like delegation depth and retention period are modeled for each parameter of the operation signature. Delegation depth determines to which services the data can be forwarded. Retention period defines how long the data can be stored by the service provider. In addition, SeSAME allows to specify reputation of services in terms of their ratings given by previous users, as well as simple price constraints.

2.2 Composition Design

In order to build a desired service composition, the software architect needs to model it with SeSAME. The architect starts with a black-box specification, which specifies the exposed properties of the Orchestrator from Fig. 1. Firstly, she models the provided interface of the Orchestrator, to which the provided interface of the service composition is
delegated. The specification of these interfaces including the operation planTrip is presented in Fig. 1.

Secondly, she creates a white-box specification of the Orchestrator by modeling interfaces of services, which have to be a part of the composition (two required interfaces of the Orchestrator). The required interfaces state the requirements on searched services modeled as black box specifications, in which functional and non-functional properties of searched services are specified as explained in Section 2.1. These specifications will be used for matching during the discovery of provided services specified by service providers.

As a part of the white-box specification, the control flow between the required service interfaces has to be specified. For example, start with finding the closest bus station by calling getBusStation and continue with calculating a bus route from this bus station by calling calcBusRoute. Due to space restrictions, we omit the specification of the required interfaces in our example in Fig. 1. These specifications look slightly different from the shown specifications of provided interfaces, for which we check the compliance via matching.

2.3 Service Discovery and Matching

In order to discover services to be part of the composition, specifications of required and provided interfaces need to be matched to each other. Matching is most accurate if different properties of a service are specified and taken into account. Thus, we provide a matching process consisting of matching steps for functional service properties (keywords, signatures, pre-/postconditions, and protocols) and non-functional properties (privacy, reputation, and price).

An extension adding more matching steps, e.g., performance matching, is work-in-progress.

For an example matching, consider the signatures for the provided MapService depicted in Fig. 1. A requested signature like getStation(Location b, City c): Station s matches to the depicted signature as the matcher follows the rules of covariance and contravariance based on ontological relations: (a) the inputs at the provider side are a subset of the inputs at the requester side and their types match as the provider’s inputs are equivalent or more general than the requester’s inputs and (b) the requester’s outputs are a subset of the provider’s outputs and their types match as the requester’s output is more general than the provider’s output (Station is a superclass of BusStation). Similarly, the other parts of the specification (e.g., pre-/postconditions) are matched with respect to the question whether the provided service satisfies the requester’s requirements. For more details, please refer to our earlier work, e.g., [9, 10].

The matching process is configurable in two ways: On the one hand, some of the matching steps can be configured. For example, for the signature matcher, we can decide whether parameter names and/or operation names should be taken into account or not. On the other hand, the process itself can be configured with respect to the selection and order of matching steps. For example, it is practical to perform fast matching steps before slower ones such that mismatching services can be excluded early. Furthermore, some of our matching steps support fuzzy matching [9].

The matching process is not to be confused with the subsequent analysis as the analysis takes into account the whole composition to be created, while matching takes into account single, already existing services. As matching has to deal with large markets, it has to be faster than the analysis, while the analysis is designed to be much more accurate.

2.4 Analysis

Analysis of a composition is based on the specification of services in the composition, and their interaction, as defined in the control and data flow of the composition.

2.4.1 Functional Analysis

In SeSAME, functional analysis is done as formal verification. SeSAME supports two types of verification: (a) The composition’s (required) postcondition indeed follows from its specification; and (b) the required call order (protocol) can indeed be served by the composition, and does not violate the protocols of the single services of the composition. As a result, the requirements are either proven to hold, or a counterexample is provided.

For verification based on pre- and postconditions, SeSAME derives a logical representation $\varphi$ of the composition, based on its control and data flow and actual services [12]. The composition is correct, if the required postcondition can always be entailed from $\varphi$, assuming the composition gets correct input (guaranteed precondition holds) and utilizing the available domain knowledge. SeSAME encodes this question as satisfiability problem and uses standard solvers to prove or disprove this functional requirement.

In Fig. 1, the architect requires a maximum trip duration (predicate durationLessThan, operation planTrip); at the same time, no such guarantee is given by the traffic system service. As a result, functional analysis will produce a counterexample including a trip with a duration which violates the architect’s requirements.

As the satisfiability problem for first-order logic is generally undecidable, successful verification is not guaranteed. However, domain knowledge encoding often restricts itself to a decidable subset of first-order logic. Additionally, depending on the complexity of pre- and postconditions (e.g., usage of quantifiers), underlying encoding of control and data flow (e.g., encoding of set data types), and theory support of actual solvers (e.g., quantifiers), verification of concrete examples is often decidable in practice.

Verifying protocols includes two checks: (1) All orders as defined by the required protocol must be executable by the composition (composition’s orders result from its control flow and protocols of its services). (2) Calling an operation must not lead to a deadlock. SeSAME transforms the composition’s control flow and the service’s protocols to Promela process specifications, which serve as input to the SPIN model checker [6].

2.4.2 Non-Functional Analysis

Classic performance metrics like response time and utilization can be predicted for usage scenarios modeled in SeSAME, if services are annotated with internal resource demands, see Fig. 1, and resource environments are annotated with their capacity. Furthermore, SeSAME supports predictions for elasticity, and efficiency metrics for software services running in cloud computing environments [1].

For classical performance analyses SeSAME provides three analysis techniques: (a) analytical solving, (b) simulation, and (c) performance prototypes. The listed analysis techniques differ in their run-time and accuracy of their predictions. Analytical solving is fast but restricted to mean response times. Simulation is slower than analytical solving.
but enables various statistical aggregations for performance metrics like response time. Performance prototypes are generated services that can be deployed on target run-time environments, like web servers, and consume the resources in the run-time environment as specified. Thus, performance prototypes deliver realistic performance measurements, but also require manual effort. Prediction of elasticity, and efficiency metrics is currently limited to simulation in SeSAME.

3. REALIZATION
SeSAME is realized as Eclipse plugins using model-driven software development technologies like Eclipse Modeling Framework (EMF) [11]. The service specification language in SeSAME is an extension of Palladio Component Model [3] defined as metamodels with the corresponding generated editors. Ontologies used in SeSAME are modeled in the OWL2 format [8]. Our tool relies on different external APIs (e.g., Apache Jena for ontology handling), and tools (e.g., solvers like Z3 [4], model checkers like SPIN [6]). SeSAME also supports transformations from existing languages into the SeSAME specification language (e.g., SAWSDL). SeSAME is easily extensible, e.g., with the help of the extension point mechanism of Eclipse. New matchers, solvers, or transformations can easily be integrated into SeSAME.

4. RELATED TOOLS
SeSAME is the first tool-suite to provide an integrated support for all mentioned tasks that have to be performed for designing a service composition. We present the closest related tools, covering only parts of SeSAME functionality.

Palladio [3] is a model-driven software performance engineering approach. Palladio’s key feature is its integrated support for design and performance analysis of component-based software. For this purpose, Palladio introduces its own component model, the Palladio Component Model (PCM), which allows to annotate performance-relevant information. SeSAME builds on Palladio and extends the PCM. For example, the non-functional analysis in SeSAME is based on Palladio, however, the contribution of SeSAME are further modules (listed above) and their integration.

The Fujaba Real-time Tool Suite is a tool-suite for modeling and formally verifying the software of mechatronic systems using MechatronicUML [2]. Analogous to SeSAME, it provides tool support for a broad range of specifications and verifications for software components. However, it focuses on real-time systems instead of services and covers neither matching nor quality analysis.

Furthermore, there are a lot of different matchers available. One up-to-date matcher, for example, is iSem, which achieved the best results in the Semantic Service Selection contest in 2012 w.r.t. the trade-off between matching accuracy and response time [7]. iSem is comparable to our signature matcher and also covers pre-/postcondition matching. However, as other matchers, iSem only focuses on matching and is not able to do further analyses.

5. CONCLUSIONS
This paper presents SeSAME, a tool that allows to model services and service compositions, to discover services, and to analyze a resulting composition. It provides a comprehensive service specification language, which allows to specify functional and non-functional properties of services and service compositions as black-box and white-box specifications. The service matching supported in SeSAME is based on these comprehensive specifications and produces highly precise matching results. Analysis of service compositions performed in SeSAME ensures their functional correctness w.r.t. their specifications. Furthermore, the analysis predicts the non-functional properties of the compositions. All in all, SeSAME is an integrated tool-suite that facilitates the creation of high-quality service compositions.

As future work, we plan to extend our tool by integrating techniques to cope with uncertainty induced by incomplete specifications or fuzzy requirements. Furthermore, we want to add more sophisticated methods for model transformations and to investigate to which extent SeSAME functionality can be applied to service specifications from existing repositories after being transformed into SeSAME.

6. ACKNOWLEDGMENTS
This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center “On-The-Fly Computing” (CRC’901). We also would like to thank our students for their contribution to SeSAME.

7. REFERENCES