A Flow Analysis Approach for Service-Oriented Architectures

Bernhard Bauer, Melanie Langermeier, and Christian Saad

Software Methodologies for Distributed Systems, University of Augsburg, Augsburg, Germany
{bauer,langermeier,saad}@ds-lab.org

Abstract. The discipline of SOA (Service-oriented Architecture) provides concepts for designing the structural and behavioral aspects of application landscapes that rely on the interaction of self-contained services. To assess an architecture's quality and validate its conformance to behavioral requirements, those models must be subjected to sophisticated static analyses. We propose a comprehensive methodology that relies on data flow analysis for a context-sensitive evaluation of service-oriented system designs. The approach employs a model-based format for SOA artifacts which acts as a uniform basis for the specification and execution of various analyses. Using this methodology, we implement two analyses which reveal blocking calls and assess performance metrics. These applications are evaluated in the context of two case studies that have been developed in the SENSORIA and the ENVIROFI projects.

1 Introduction

The field of SOA is concerned with methods that enable the conceptual design of the relevant aspects of software ecosystems whose components interact in complex yet well-defined patterns to provide high-level services to consumers. This abstraction not only supports the task of documenting service landscapes in enterprises, the model-based formalization also facilitates automated code generation, following the principles of model-driven development (MDD). In this sense, SOA models represent integral artifacts of software development processes. Usually, the target system is first described at a higher level of abstraction with iterative refinements. To avoid the multiplication of errors in later stages, problems must be identified as early as possible. Analysis at the model level can also help in assessing the architecture's quality and in validating its conformance to functional and technical requirements before implementation begins.

Current approaches for the analysis of object-oriented models often focus on trivial metrics such as the number of classes, the number of methods per class or the number of sub/super classes [6]. Typically, the information aggregated by these methods only considers immediate neighbors [7]. For some use cases, canonical analysis techniques are not well-balanced with respect to their expressiveness and the resulting implementation effort: For context-sensitive measures and validation scenarios, methods such as the OCL are not sufficient while systems based on formal logic tend to introduce unnecessary complexity. The fact

that business models are often incomplete or inconsistent further hinders the application of strict formal systems.

To close this gap, we establish a unified analysis methodology which relies on the principle of information propagation to enable a context-sensitive evaluation of SOA models. The motivation for this approach can therefore be summed up as follows: We intend to provide developers with a generic framework for implementing analyses in the SOA domain that do not necessitate the usage of formal semantics but, nevertheless, cannot be expressed using traditional constraint languages such as OCL due to their context-sensitive nature. The technique is therefore intended as an extension, rather than a replacement, for existing methods such as formal verification [22].

For this purpose, we employ a uniform model-based format which acts as a foundation for the specification and execution of analyses to abstract from the diversity found in canonical modeling languages for SOA. Based on this representation, we employ the model-based data flow analysis as a declarative "programming language" for the implementation of various analyses that depend on the computation of a fixed point (for approximating a system's run-time behavior) and/or the modeling of complex information flows through the designed architecture. We subsequently demonstrate how this methodology can be applied to compute performance metrics and check for potential blocking calls in contract and interface-based architectures. The approach is evaluated in the context of two case studies, that have been published by the SENSORIA [17] and the ENVIROFI project [8] respectively. We base our work on previous research, in which we applied a similar strategy to the field of enterprise architecture management (EAM) [12]. More specifically, the contributions of this paper comprise the adaptation of the methodology to the SOA domain, its evaluation in the context of existing case studies and the implementation of relevant analysis scenarios. Furthermore, the application to SOA is intended to emphasize the viability of the proposed methodology across different fields of research.

In the following section, we present an overview over related work and establish the link to the SENSORIA project. In section 3, we detail the different aspects of the analysis methodology, namely the flow-based analysis of models and the generic representation of SOA data. The next two sections describe two different analysis scenarios and their application to the use cases. The first one checks for blocking calls (section 4) while the second one derives performance measures (section 5). We conclude with a discussion of our method (section 6).

2 Related Work

Although service-oriented architectures have attracted much attention, the concept has often only been applied in an ad-hoc fashion. The SENSORIA project [17] defines a comprehensive approach for the design, formal analysis and automated deployment as well as the re-engineering of service-oriented applications [22]. SOA models are analyzed using formal methods such as process calculi, temporal logic and stochastic logic. Analyses are both qualitative and quantitative, e.g. conformance with contracts, deadlock freedom of compositions or the

analysis of service properties like availability [22]. The formal foundation is tied to the UML4SOA [13] profile, a high-level domain specific language which incorporates behavioral aspects. This profile extends a previous version of the SoaML standard (to which the first author contributed) with concepts for modeling the behavior of services, service orchestrations, and service protocols. The current version of SoaML [3] (published in 2012) introduces two different architectural styles, based on interfaces and service-contracts respectively, and integrates UML sequence diagrams for the specification of communication protocols.

The Object Constraint Language (OCL) [2] is a widely used method for the specification of simple model analyses. For example, [6] provides a library for the extraction of metrics. OCL constraints enrich the abstract syntax of a modeling language, i.e. the meta model, with a definition of its static semantics. This means that analysis specifications are tightly integrated with the modeling ecosystem (i.e. canonical standards and tools) and therefore naturally support concepts such as generalization and instantiation. In contrast to the proposed flow-based method, OCL does however not support information propagation and fixed point convergence and is therefore restricted to basic validation scenarios.

In [12], we established a generic meta model (GMM) which encodes the structural composition of enterprise architectures in the form of a stereotyped graph. While we were able to redefine existing analyses using this unified representation, the lack of specific semantics has been identified as a challenge that must be addressed in future work.

3 Adaptive Analysis Methodology for SOA

This section describes an adaptive analysis methodology for the SOA domain which relies on static derivation of context-sensitive properties to validate an architecture's correctness and assess different types of quality attributes. The resulting approximation of the modeled system's runtime properties can provide valuable feedback, especially in early stages of the development process.

The design of an analysis methodology intended for use in the SOA domain poses different challenges: For one, architectures may be encoded in a variety of modeling languages, e.g. SoaML, UML4SOA or BPMN[1]. Furthermore, architectures may rely on different paradigms, such as contract- or interface-based styles. To avoid conceptual and technological gaps, the analysis technique should also be well-integrated with modeling standards such as the Meta Object Facility (MOF) and be capable of addressing a wide range of application scenarios.

We address these issues by combining a unified representation of SOA-specific (meta) model data with a framework for model-based flow analysis. Translation of the target SOA model into the unified SOA format can be achieved using canonical methods for model transformation such as Query/View/Transformation (QVT). Subsequently, the analyses can be executed. Figure 1 provides an overview of this process. This technique has already been successfully implemented for

[1] The Business Process Model and Notation (BPMN) can be used to specify behavioral aspects of services [16].

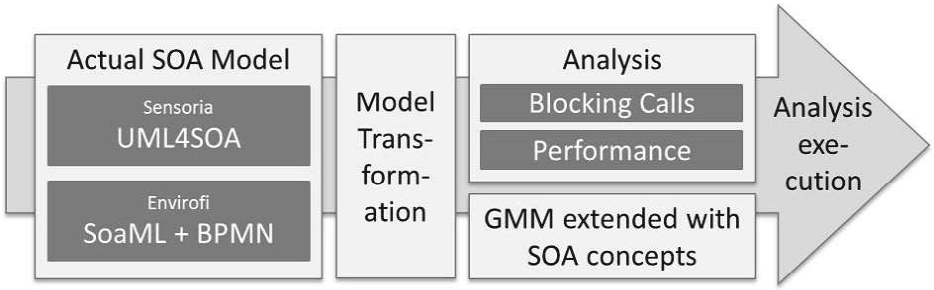

Fig. 1. Procedure of the data flow analysis approach for SOA

the EAM domain [12], a field which shares many of the described challenges, including the diversity of modeling standards and the fact that models may be incomplete or even inconsistent. To alleviate the problem of complex analysis specifications, we will subsequently extend the generic representation with domain-specific SOA concepts. Sections 4 and 5 exemplify the approach in the context of two case studies from the SENSORIA and the ENVIROFI project and two analysis scenarios, detection of blocking calls and computation of performance metrics.

3.1 Flow-Based Model Analysis

The technique of data flow analysis (DFA) is commonly employed in the area of compiler construction to analyze and optimize the control flow of programs by examining how information that is computed locally at the nodes (basic blocks) of a program control flow graph is disseminated. A canonical examples consists in the *reaching definitions* analysis, in which variable definitions generated inside basic blocks are propagated through the graph to determine the availability of variable assignments at subsequent instructions. By applying fixed point evaluation semantics, it is possible to compute with cyclic equation systems that result from the presence of loops in the control flow. Analyses are usually specified in a way that ensures that the result represents a conservative approximation of the program's run-time behavior.

The analysis specifications presented in the subsequent sections rely on the approach detailed in [15] which transfers the notion of data flow analysis to the modeling domain. Inspired by the related technique of attribute grammars [5], it supports the declaration of *data flow attributes* which can be assigned to classes in a target meta model. In some respects, this process can therefore be compared to the Object Constraint Language which is often used to formalize the static semantics of modeling languages by assigning constraints to meta model classes. However, each data flow attribute is connected to two *data flow equations*, which compute the attribute's initial value and its fixed point iteration result(s) respectively. Furthermore, to compute the result for a specific attribute, its data flow equation may access values of neighboring attributes, thereby inducing an information flow between the model's elements.

To execute the analysis, the *data flow solver* is supplied with the meta model, the data flow specification and the target model. In a first step, the attributes

are instantiated for model elements of the respective types and initialized with their start value. Afterwards, the iteration values are computed by executing the associated equations. The solver monitors the propagation of data flow information between attribute instances and, if necessary, initiates the recomputation of unstable instances, until a fixed point has been reached.

This approach has multiple advantages: The method is fully integrated with the modeling domain, thus avoiding potential semantic gaps between different technological spaces. Furthermore, the declarative nature of the data flow specifications allows for an intuitive definition of analyses which rely on the (transitive) propagation of information along model paths. When computing the result for a concrete element in the model, it is therefore possible to take into account its overall context, that results from (transitive) connections to other elements. Finally, the fixed point semantics enable a conservative approximation of the run-time behavior of the modeled system.

3.2 Generic Meta Model

In the field of SOA, many competing standards and practices exist. It is therefore essential to provide a unified basis for analysis specifications to avoid the constant adaptation of existing analyses. Instead, the interpretation of language artifacts will be encoded in transformations for different source languages such as SoaML.

In previous work [12] we established a generic meta model (GMM) for enterprise architecture analysis. This format constitutes a high-level view on model data by abstracting from characteristics which may vary between different standards. In essence, it conforms to a stereotyped graph which incorporates model-oriented extensions such as properties and generalization relationships. It is important to note that a GMM instance represents both meta and model data. Case studies carried out in the field of enterprise architecture analysis have shown that this approach supports a wide variety of different modeling paradigms although the lack of domain-specific semantics tends to complicate analysis specification.

To provide better support for SOA-specific features, we extended the original GMM with concepts found in canonical ontologies, modeling languages and reference models from the SOA domain [10]. In their work, Kreger and Estefan examine different standards and conclude that the specifications agree on a set of core concepts. Based on this study, we established an abstract SOA model which incorporates features from The Open Group's SOA ontology [19] and Reference Architecture [20] as well as the OASIS Reference Model [14]. Figure 2 shows the resulting meta model with the essential classes and relationships.

The identified core concepts have been woven into the GMM as shown in figure 3. This representation can therefore be understood as a domain-specific language tailored to the specification of structural analyses in the SOA domain. While the right hand side encodes model data in the form of stereotyped *Nodes*, *Edges* and *Properties*, the left hand side represents the meta structure of the respective SOA language. Each specific concept inherits from the generic *MetaModelNode* while each relationship type is represented as a sub class of *MetaModelEdge*. Since they

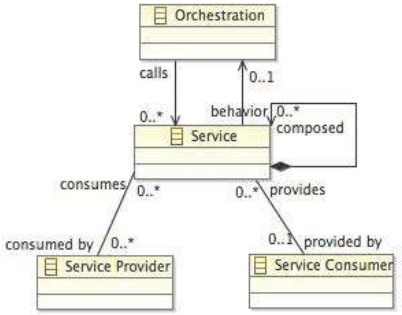

Fig. 2. Meta model capturing the core structure of service-oriented architectures

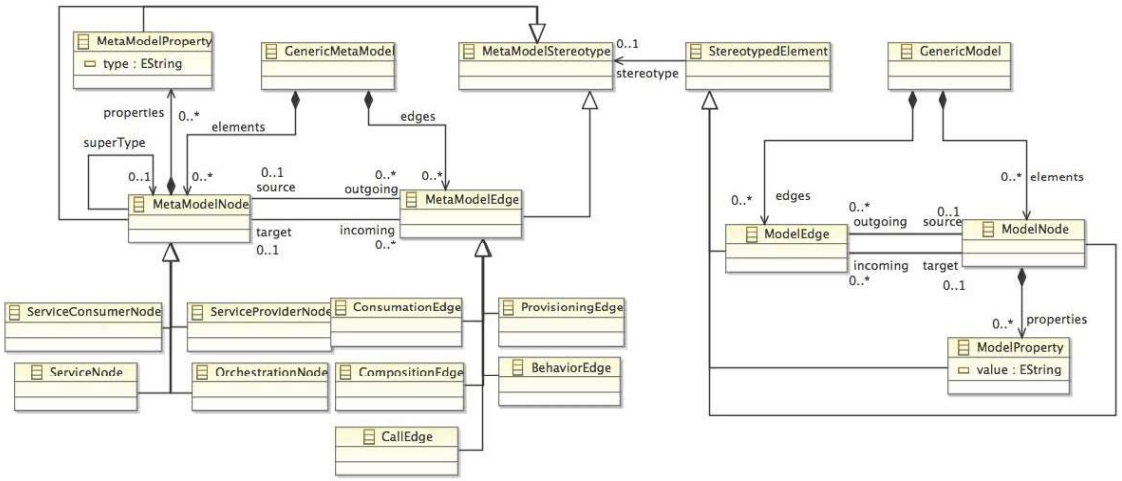

Fig. 3. Generic Meta Model [12] adapted for the SOA domain

are also specializations of the abstract class *StereotypedElement*, these concepts act as "data types" for model data.

The chosen layout allows for a certain degree of freedom when importing SOA models, as language-specific characteristics can be represented without modifications to the GMM, eliminating the need to adapt existing analyses. This generic approach is viable, since data flow analyses rely on information propagation and can therefore cope with extensive changes in the underlying language' structure.

The potential downside of this approach consists of increased complexity in the transformation logic. The benefits of the generic representation, namely the robustness of the analyses themselves, must therefore be weighed against the effort required for the translation of SOA models. If it can be expected that the underlying structure remains constant, it can therefore be beneficial to tie the data flow specifications directly to the target language's meta model.

3.3 Case Studies

The case studies, which form the basis for the evaluation of the proposed methodology, consist of two models that rely on different architectural styles. The first

is an extended version of the automotive case study *On Road Assistance* from the SENSORIA [17,9] project. The second describes a *Personal Environmental Information System* (PEIS) and was developed in the ENVIROFI [8,11] project.

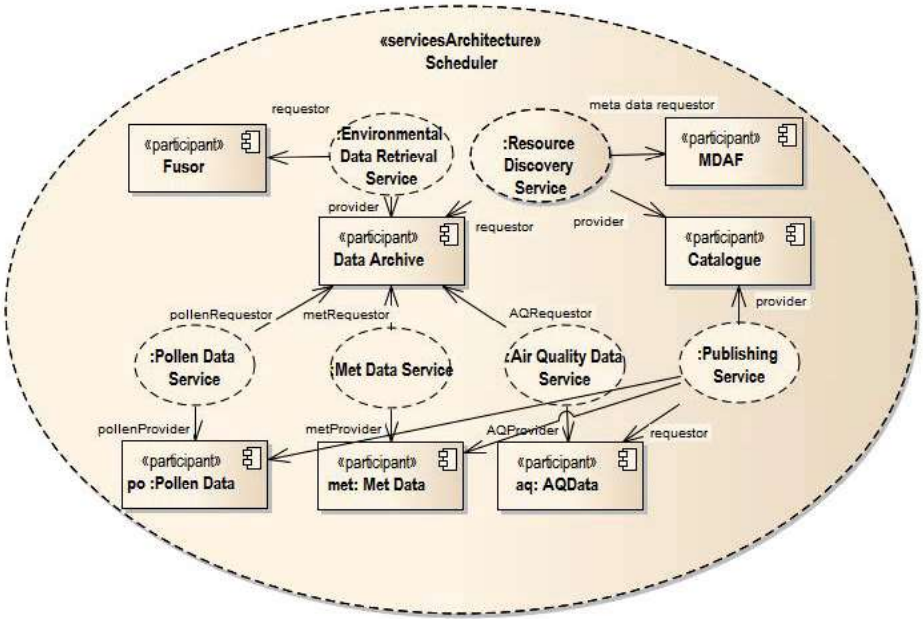

Fig. 4. Service Architecture for the participant *Scheduler* in the *PEIS* use case [11]

The *On Road Assistance* scenario supports the driver of a car if an engine failure makes it impossible to reach the planned destination. For this purpose, the SOA participant *OnRoadAssistant* invokes multiple services to find the "best" repair shops (garages) and rental car stations nearby, once the driver has made a security payment. The architecture uses the interface-based style of the SoaML specification [3] while behavioral aspects regarding service composition are modeled using the UML4SOA profile [13] developed by SENSORIA. We extended the definitions from [9] with a second participant *AssistanceStore* (excerpts can be seen in figure 5). The interactions between both participants form the basis for the analysis of blocking calls in section 4. Mappings between the UML4SOA/Activity Diagram and the extended GMM are shown in table 1.

The second scenario represents a *Personal Environmental Information System*, which generates personalized reports of pollen, air quality and meteoro-

Table 1. Mapping of UML4SOA concepts to the extended GMM

Ext. GMM concept	*UML4SOA/Activity Diagram concept*
ServiceNode	ServiceInterface
OrchestrationNode	ActivityDiagram
AggregationEdge	ServiceInterface ▷ Port
BehaviorEdge	Port ▷ LinkPin ▷ ActivityDiagramElement ▷ ActivityDiagram
UseEdge	ActivityDiagram ▷ ServiceSendAction ▷ LinkPin ▷ Port ▷ ServiceInterface

logical data depending on the current location of the user (cf. figure 4). The architecture relies on the contract-based SoaML approach [3]. Internal participant behavior is modeled using BPMN [1]. As proposed in [11] and [16], the BPMN diagrams have been enriched with mappings that connect service actions to their corresponding service interfaces. In this case, a different set of mapping rules has to be applied to correctly represent the contract based architectural style as depicted in table 2.

Table 2. Mapping of (contract-based) SoaML/BPMN concepts to the extended GMM

Ext. GMM concept	*SoaML/BPMN concept*
ServiceNode	ServiceContract
OrchestrationNode	BPMNDiagram
AggregationEdge	ServiceContract ▷ ServiceContract
BehaviorEdge	ServiceContract ▷ Participant ▷ BPMNDiagram, ServiceContract ▷ Participant ▷ ServiceArchitecture
UseEdge	BPMNDiagram ▷ SendAction ▷ Service

4 Analysis of Blocking Calls

Dependencies relating to the orchestration of services are an important aspect of service-oriented architectures. In larger systems, the call hierarchies (directly and indirectly invoked services) are often not obvious which may lead to blocking calls/deadlocks. In [4], Acciai et al. propose a type system to ensure deadlock freedom of the subsequent conversations after service invocation in well-typed CaSPiS processes. We implement a light-weight alternative to this approach based on a static approximation of the system's runtime behavior with the goal of detecting *potential* cyclic invocations.

In the general case, detecting blocking calls in an orchestrated architecture requires the identification of all the direct and indirect service calls that are required for the execution of a service. If this set contains the original service, the architectural design *may* result in a deadlocked system. Since orchestrations are typically described via process diagrams (e.g. UML Activity Diagrams, BPMN or BPEL models), the analysis has to focus both on the internal composition of a process and the (transitive) interactions between different orchestrations. By computing a fixed point of required service calls for each service, it is possible to either guarantee that the architecture will not result in blocking calls or to indicate potentially problematic situations to the user. In the following we will describe how this analysis can be specified and applied to both case studies.

4.1 Analysis Specification

To assess implicit service calls, we assign a data flow attribute *requiredServices* to the *ServiceNode* class in the architecture's GMM representation whose instances

compute the sets of invoked services for each node. A second data flow attribute *serviceCalls* computes the set of called services for *OrchestrationNodes*. The respective data flow equation rules are shown in algorithms 1 and 2.

Algorithm 1. Data flow equation for the attribute *requiredServices*

```
1: DFA-EQUATION ServiceNode::requiredServices returns Set<ModelNode>
2:   Set<ModelNode> services = new Set<ModelNode>();
3:     // acquire values from composed services
4:     foreach (ModelEdge outgoingAggregation in self.outgoing)
5:       if (outgoingAggregation.target.stereotype is 'ServiceNode')
6:         services.addAll(outgoingAggregation.target.requiredServices());
7:     // acquire values from called services in the orchestration
8:     foreach (ModelEdge outgoingBehavior in self.outgoing)
9:       if (outgoingBehavior.target.stereotype is 'Orchestration')
10:        services.addAll(outgoingBehavior.target.serviceCalls());
11:  return services;
```

Algorithm 2. Data flow equation for the attribute *serviceCalls*

```
1: DFA-EQUATION OrchestrationNode::serviceCalls returns Set<ModelNode>
2:   Set<ModelNode> services = new Set<ModelNode>();
3:   // acquire values of called services from the orchestration
4:   foreach (ModelEdge outgoingCall in self.outgoing)
5:     if (outgoingCall.target.stereotype is 'ServiceNode')
6:       services.addAll(outgoingCall.target.requiredServices());
7:   return services;
```

The result set for the attribute *requiredServices* is first initialized with an empty call set (line 2). For aggregated services, the required calls of the substructures must be added as well. This is accomplished by requesting the value of *requiredServices* at those elements (which will implicitly trigger the data flow solver to recursively evaluate these dependencies) and adding them to the result (lines 4 - 6). To include behavior-related invocations, lines 8 - 10 process service calls of orchestrations (computed by *serviceCalls* for *OrchestrationNodes*).

The data flow solver will automatically terminate once a fixed point has been reached, i.e. no more elements are added to any result set. By examining the values computed by *requiredServices* for *ServiceNodes*, it is possible to detect whether a service may trigger its own invocation. If this is not the case for any service, the system will never execute a blocking call. Otherwise, a deadlock might exist, although not every execution necessarily results in that situation.

4.2 Case Study

The described analysis has been carried out for both use cases. For this purpose, the meta and model data has been transformed into the extended GMM format using the mappings from tables 1 and 2. For reasons of clarity, we will depict the results in the original representation rather than using the GMM concepts.

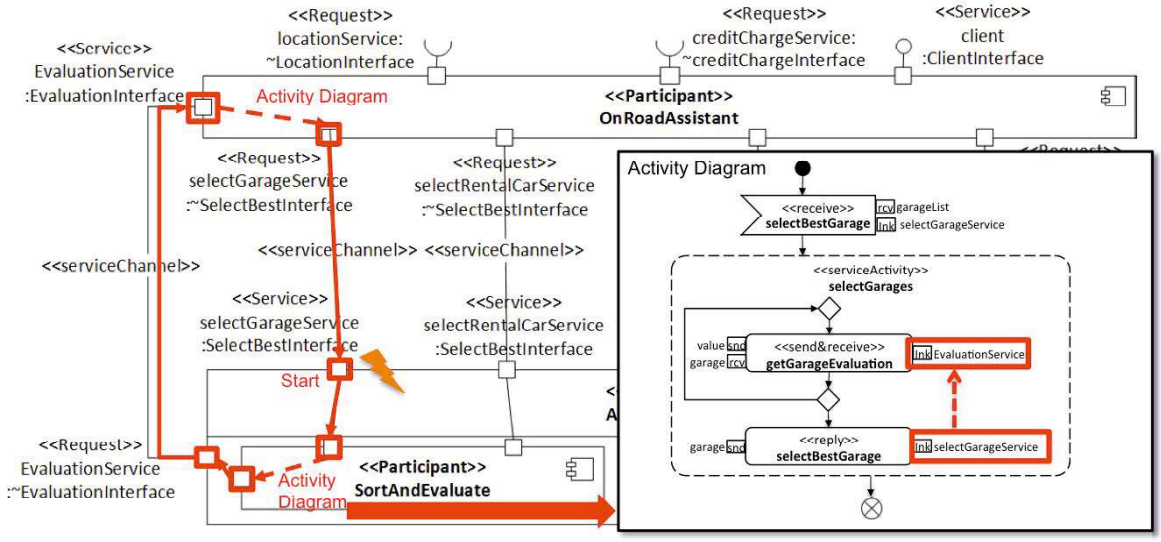

Fig. 5. Illustration of the blocking call in the use case *On Road Assistance*[9]

Figure 5 shows the identified potential deadlock for the *OnRoadAssistance* model. In this case, we assume that data flow analysis started with the *ServiceInterface SelectGarageService*, which relies on the providing participant *AssistanceStore* (bottom). Its *ServicePoint* delegates to the internal component *SortAndEvaluate*, which has an associated Activity Diagram describing its internal behavior (right corner of figure 5). In this diagram, the *selectGarageService* is connected to the *ReplyAction selectBestGarage* via a *LinkPin*. Here, the set of preceding *SendActions* contains only the action *getGarageEvaluation*, which is linked to the *RequestPort EvaluationService*. This service, in turn, is connected to a *ServicePort* of the *OnRoadAssistance* participant.

Analysis of the component's behavior indicates that the *EvaluationService* may trigger the execution of the *selectGarageService* and the *selectRentalCarService*. As both implement *SelectBestInterface*, this element will be added to the result set. Subsequently, the algorithm determines the service calls of *selectRentalCarService*, which is provided by the *SortAndEvaluate* participant and uses the *EvaluationService*. Afterwards, the execution of the data flow rules terminates, as the analysis has converged in a fixed point. Because the call hierarchy of *SelectBestInterface* is cyclic, the absence of blocking behavior cannot be guaranteed for this architecture. The concrete result sets for all services are as follows[2]:

ClientInterface: { LocationService, creditChargeService, EvaluationService, SelectBestInterface, FindInterface }
EvaluationService: { SelectBestInterface, EvaluationService }
SelectBestInterface: { EvaluationService, SelectBestInterface }

For the PEIS example, the connection of BPMN actions to the used services is modeled via links as proposed by [16] and [11]. The call hierarchies for *Resource*

[2] Since the *LocationInterface*, the *CreditChargeInterface* and the *FindInterface* have no required service calls, they have been omitted from this representation.

Discovery Service and *Publishing Service* are empty since they have no required service calls. The result set for *Pollen Data Service*, *Meterological Data Service* and *Air Quality Data Service* consists only of the *Publishing Service*. Finally, the *Environmental Data Retrieval Service* may request the invocation of *Resource Discovery Service*, *Pollen Data Service*, *Meteorological Data Service*, *Air Quality Data Service* and *Publishing Service*. Since no result for the attribute *requiredServices* contains the associated service itself, we can conclude that the architecture of the *Scheduler* participant in the PEIS has no blocking calls. This is also evident from the structural overview shown in figure 4: Even without detailed knowledge about the inner structure of the participants, there exists no sequence of service calls that could potentially result in a cyclic invocation.

5 Performance Analysis

Performance aspects can be of vital importance in an SOA environment, especially if the architecture relies on a complex orchestration of many services. In this case, a static assessment of the run-time properties can provide valuable early feedback to the developer which can help in improving the system design to shorten response times for critical components. Examples for this kind of analysis include a method based on layered queuing networks for UML4SOA models that have been extended with MARTE (a UML profile for real-time systems) [21] or the use of *queuing network modeling* [18]. The latter proposal applies the analysis to a *distributed message passing* architecture with asynchronous message streams (messages are being queued at the components) where the response time of a system is defined as the *Population/Arrival Rate*. The latter denotes the number of incoming requests per time unit while the former describes the sum of currently processed and all waiting requests [18].

Transferring this method to the SOA domain necessitates some adaptations since the system's functionality is not only expressed by participants' internal components but also by service orchestrations. Furthermore, instead of having a single arrival rate for the whole system, each service may possess its own arrival rate which has to be based on the rates of all its requestors and an internal rate. The population of a service then depends on the internal population plus the population of all requested services during the provisioning.

It should be noted, that - in contrast to the blocking calls analysis - the evaluation of the performance metrics does not induce a fixed point computation. Instead, the declarative nature of data flow analysis specifications in combination with the information propagation principle is used to realize succinct and intuitive implementations of the recursive formulae.

5.1 Analysis Specification

Response time for (composite) services is computed by three data flow attributes:

- *arrivalRate* (r_s)*:* the arrival rate for each service,
- *population* (p_s)*:* the population for each service, and
- *responseTime* (t_s)*:* the response time for each service

Furthermore, the SOA model must be enriched with additional data: *Services* require a property *service time* (s_s) which denotes the execution time for internal actions (excluding external service requests). A second property *local arrival rate* (r_s^l) specifies the number of local consumer requests not triggered by other services. Based on this information, the attribute *arrivalRate* can be computed using the data flow equation shown in algorithm 3.

Algorithm 3. Data flow equation for the attribute *arrivalRate*

```
1: DFA-EQUATION ServiceNode::arrivalRate returns Integer
2:   Integer rate = self.localArrivalRate;
3:   // for all calling services
4:   foreach (ModelEdge incomingCall in self.incoming)
5:     if (incomingCall.source.stereotype is 'Orchestration')
6:       rate += incomingCall.source.arrivalRate();
7:   return rate;
```

The arrival rate of a *Service* equals to the sum of the local arrival rate (line 2) and the arrival rates of all requesting services (lines 4 - 6). For this purpose, the latter part requests the *arrivalRate* result for the orchestration(s) from which the service is invoked, thereby computing the arrival rate from its service (and, if necessary, the compound services) recursively.

After the arrival rate has been determined, it is possible to calculate the internal and the composed population. The internal population is defined as $p_s^i = \frac{u_s}{1-u_s}$, with $u_s = r_s \cdot s_s$ representing the utilization of service s. The population of a composition is given as $p_s = p_s^i + \sum p_i$, where p_i are the populations of the requested services. The data flow equation is listed in algorithm 4.

Algorithm 4. Data flow equation for the attribute *population*

```
1: DFA-EQUATION ServiceNode::population returns Integer
2:   Integer utilization = self.arrivalRate() * self.serviceTime;
3:   Integer population = utilization / (1 - utilization); // internal population
4:   // add populations of all directly requested services
5:   foreach (ModelEdge outgoingEdge in self.outgoing)
6:     if (outgoingEdge.target.stereotype is 'Orchestration')
7:       foreach (ModelEdge outgoingCall in outgoingEdge.target.outgoing)
8:         if (outgoingCall.target.stereotype is 'ServiceNode')
9:           population += outgoingEdge.target.population();
10:  return population;
```

The internal population of a *Service* is computed using the model property *service time* and the data flow attribute *arrivalRate* (lines 2 - 3). For the overall population, this result is added to the populations of all *Services*, that are called in the respective *Orchestration* (lines 5 - 9). This process involves a recursive access to the *population* attribute (line 9) for which the data flow solver ensures that the indirectly required services are considered as well.

5.2 Case Study

We will now illustrate the performance analysis for the service *FindInterface* in the *OnRoadAssistance* model. The mappings of the meta model concepts are equal to those in the blocking calls scenario. In this case, we assume deadlock freedom and therefore will not consider the *EvaluationInterface* and its connections. The internal service times are given as $s_{FindInterface} = 0,03s$ and $s_{ClientInterface} = 0,05s$. The local arrival times are $r^l_{ClientInterface} = 10$ and $r^l_{findGaragesServices} = r^l_{findRentalCarStationsService} = 5$ requests/s. Analysis execution starts by evaluating the *FindInterface's population* which automatically triggers the computation of its *arrivalRate*. In addition to the arrival rates of the implementing service ports $r_{findGaragesService}$ and $r_{findRentalCarStationsService}$, the corresponding request ports are determined based on the behavior diagram of the *OnRoadAssistant* participant. The ports are used to provide the *client* service port, i.e. the set of *usingServicePorts* for both request ports, which contains only one element, the *client*. The arrival rate of the *client* service port is equal to its local arrival rate, because this port has no connections to other request ports. The arrival rate for the *FindInterface* is thus computed by:

$$R_{FindInterface} = r_{findGaragesService} + r_{client} + r_{findRentalCarStationsService} + r_{client} = 5 + 10 + 5 + 10 = 30$$

The utilization is then calculated as $u_{FindInterface} = r_{FindInterface} \cdot s_{FindInterface} = 30 \cdot 0{,}03 = 0{,}9$ and, consequently, the internal population yields:

$$p_{FindInterface} = \frac{u_{FindInterface}}{1 - u_{FindInterface}} = 9$$

Since no service calls are required for the provisioning of the *FindInterface*, its overall population is equal to the internal population and the analysis of this interface is finished. The results show that the *FindInterface* will have a utilization of 90%, with a population of 9. The application of the performance analysis for the *PEIS* use case can be carried out in an identical fashion.

6 Discussion and Conclusion

In this paper we presented a comprehensive analysis methodology for SOA models, which combines a generic representation of (meta) model data with an analysis technique that relies on information propagation and fixed point computation to enable a context-sensitive evaluation of model elements and the approximation of run-time behavior. We have demonstrated the viability of this methodology in the context of two use cases and two analysis scenarios, which implement a deadlock analysis and an evaluation of performance metrics respectively.

As shown in previous work, the unified representation has the benefit of providing a simple, yet concise foundation for the specification of flow sensitive analyses. To further improve the accessibility of this approach, we extended the generic structure of the GMM with SOA specific characteristics. It is thereby possible to base analyses on common SOA semantics while preserving the support for a wide range of different modeling languages and styles. By propagating locally computed information through the model, the data flow technique supports

the implementation of context-sensitive analyses. As information is propagated across multiple (transitive) relationships, the specifications are invariant against a variety of structural changes in the underlying modeling language.

It should be noted that complications may arise if the concepts of the respective SOA language cannot be directly mapped to the analysis specification types. While it would generally be possible to implement the necessary adaptations in the transformation logic, a more generic solution for this problem would be advantageous. This could, for example, be realized through a sophisticated mapping technology which supports the translation of complex SOA concepts to suitable structures in the GMM representation. Possible implementations of this are for example graph transformations or semantic web technologies.

Dedication. We would like to thank the organizers for their hard work in coordinating this festschrift in honor of Martin Wirsing's emeritation and for the opportunity to contribute to it.

Personal Dedication of Bernhard Bauer: At the University of Passau, Martin teached me in Software Engineering and logic based specification methods. No other person influenced my research interests as profoundly as Martin did. I am deeply grateful to him for giving me the opportunity to conduct my first research steps at his chair in Passau and for always being open to my ideas. Apart from being a great researcher, Martin has also impressed me as a person. His helpful and friendly guidance as well as his professional expertise formed the motivation that made Software Engineering my passion and primary research interest.

Thank you very much for the many interesting and enjoyable years!

References

1. Business Process Modeling Notation (BPMN) 2.0 Specification (2011)
2. Object Constraint Language (OCL) 2.3.1 Specification (January 2012)
3. Service oriented architecture modeling language 1.0 specification. Tech. rep. (2012)
4. Acciai, L., Bodei, C., Boreale, M., Bruni, R., Vieira, H.T.: Static Analysis Techniques for Session-Oriented Calculi. In: Wirsing, M., Hölzl, M. (eds.) SENSORIA. LNCS, vol. 6582, pp. 214–231. Springer, Heidelberg (2011)
5. Babich, W.A., Jazayeri, M.: The Method of Attributes for Data flow Analysis. Acta Inf 10, 245–264 (1978)
6. El-Wakil, M., El-Bastawisi, A., Boshra, M., Fahmy, A.: Object-Oriented Design Quality Models - A Survey and Comparison. In: 2nd International Conference on Informatics and Systems (INFOS 2004) (2004)
7. Engelhardt, M., Hein, C., Ritter, T., Wagner, M.: Generation of Formal Model Metrics for MOF based Domain Specific Languages. ECEASST 24 (2009)
8. ENVIROFI: Environmental Observation Web and its Service Applications within the Future Internet - Environmental Usage Area, `http://www.envirofi.eu`
9. Koch, N., Heckel, R., Gönczy, L.: UML for Service-Oriented Systems (second version). Sensoria Deliverable D1.4b, `http://pst.ifi.lmu.de/projekte/Sensoria/del_54/D1.4.b.pdf`
10. Kreger, H., Estefan, J.: Navigating the SOA Open Standards Landscape Around Architecture. Whitepaper W096, The Open Group (2009)

11. Langermeier, M.: A model-driven approach for open distributed systems. Technical Report 2013-03, University of Augsburg (2013)
12. Langermeier, M., Saad, C., Bauer, B.: A unified Framework for Enterprise Architecture Analysis. In: Proceedings of the Enterprise Model Analysis Workshop in the Context of the 18th Enterprise Computing Conference, EDOC 2014 (2014)
13. Mayer, P., Koch, N., Schroeder, A., Knapp, A.: The UML4SOA Profile. Technical Report, LMU Muenchen Version 3.0 (2010)
14. Metz, R., McCabe, F., Laskey, K., MacKenzie, C.M., Brown, P.F.: Reference Model for Service Oriented Architecture 1.0. Official OASIS Standard (2006), http://docs.oasis-open.org/soa-rm/v1.0/
15. Saad, C., Bauer, B.: Data-Flow Based Model Analysis and Its Applications. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) MODELS 2013. LNCS, vol. 8107, pp. 707–723. Springer, Heidelberg (2013)
16. Sadovykh, A., Desfray, P., Elvesæter, B., Berre, A.-J., Landre, E.: Enterprise architecture modeling with SoaML using BMM and BPMN - MDA approach in practice. In: 6th Central and Eastern European Software Engineering Conference, pp. 79–85. IEEE (2010)
17. SENSORIA: Software Engineering for Service-Oriented Overlay Computers (2010), http://www.sensoria-ist.eu
18. Spitznagel, B., Garlan, D.: Architecture-based performance analysis (1998)
19. The Open Group: Service-Oriented Architecture Ontology. Standard (2011)
20. The Open Group: SOA Reference Architecture. Standard (2011)
21. Tribastone, M., Mayer, P., Wirsing, M.: Performance prediction of service-oriented systems with layered queueing networks. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 51–65. Springer, Heidelberg (2010)
22. Wirsing, M., Hölzl, M., Koch, N., Mayer, P.: SENSORIA – software engineering for service-oriented overlay computers. In: Wirsing, M., Hölzl, M. (eds.) SENSORIA. LNCS, vol. 6582, pp. 1–14. Springer, Heidelberg (2011)