International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 07, July-2019

A Precise Method for Modelling Service Systems using UML

Tran Khanh Dung
National University of Civil Engineering
55 Giai Phong, Hai Ba Trung, Hanoi, Vietnam
dungtk@nuce.edu.vn

Doan Nhat Quang
University of Science and Technology of Hanoi, Vietnam Academy of Science and Technology, 18 Hoang Quoc Viet, Cau Giay, Hanoi, Vietnam
doan-nhat.quang@usth.edu.vn

Dang Thanh Trung
Hanoi National University of Education, 136 Xuan Thuy, Cau Giay, Hanoi, Vietnam
trungdt@hnue.edu.vn

***Abstract*—The architecture of service systems enables business functionalities to be invoked over a remote network. This kind of systems is more and more developed nowadays. The modelling method for developing service-oriented solutions thus is required not only specifically but also precisely. Our work focuses on the method for modelling service systems having service-oriented architecture. Our approach is based on SoaML, a standard widespread notation introduced by OMG as a standard UML profile. In this paper, we propose a precise method for modelling the services and the service systems, named PreciseSoa, and illustrate the method using a case study.**

Keywords:- Modelling methods; service-oriented systems; service systems; SoaML; UML

I. INTRODUCTION

Service Oriented Architecture (SOA) [1] is a way to build business applications as a set of participants that are considered as service providers and consumers. Up to now, SOA has been associated with a variety of approaches and technologies and became a solution for building a system for change. SOA allows application functionalities to be provided and consumed as sets of services, and enables a community, an organization or a system of systems to work together more cohesively using services without getting overly coupled.

The traditional development methods, related techniques, and notations have been found inadequate to support the development of service-oriented systems, so this has motivated work on development methods for SOA based systems.

In this paper we introduce a "precise" (exactly or sharply defined or stated, Merrian-Webster Dictionary) modelling method for the design of service oriented systems, based on SoaML (Service oriented architecture Modelling Language) [2], a UML standard profile.

SoaML is an OMG (http://www.omg.org/) standard notation for service modeling adopted in December 2009 [2]. UML is the core modelling standard notation of OMG, and they built SoaML as a UML profile for modelling services and service oriented architecture. SoaML is relevant since it has been developed by putting together various different proposals expressed by the main companies and research centers interested in SOA.

Let us recall that service is defined as the delivery of value to another party, enabled by one or more capabilities [2]. A service is provided by a participant acting as the provider and used by a participant acting as the consumer. Those participants provide and consume several services to fulfill a purpose. The specification of a service includes: the roles each participant plays in the service such as provider or consumer; the interfaces provided and used by each participant; the messages exchanged between the participants while enacting the service; the choreography of the interactions between the participants.

A service oriented system consists of a set of participants providing and consuming services. A participant may provide or consume several services, whereas a service establishes a connection exactly between two participants: the provider and the consumer. A participant may be also structured in terms of services to be able to be fulfill the contracts corresponding to the provided and consumed services.

In this paper, we consider not only the modelling of the services but also other typical aspects of a SOA-based system such as service architecture, participants, and configurations. The work in modelling service isolation in [3] is thus repeated again in this paper for presenting full content of the method.

We will illustrate our method for using SoaML with a running example, a sample service system, precisely the "Dealer Network" taken in [2], informally described as follows.

The Dealer Network is a business community including three primary parties: the dealers, the manufacturers and the shippers. They are independent parties but they want to work together. There are three services to be built: the Place order service, the Ship status and the Shipping request. The dealers use Place order service to make an order for the amount of product they want to buy from the manufacturers. The service supplies the dealers with the quotes, and sends back an information on the status of the order. This information includes whether the order is cancelled or confirmed, and the way bill number to track the shipment information.

We introduce in Sections II our method for developing precise service system models using SoaML and illustrate them on the Dealer Network case study, in which we illustrate the service model using the Place order service. Finally, we present some the related works in Section III, and draw some conclusions in Section IV.

II. PRECISE SERVICE SYSTEM MODEL WITH SOAML

We propose here an approach for service system modelling using the SoaML notation in a precise way. We provide a metamodel defining the structure of the service system model, equipped with well-formedness constraints so as to guide the use of the notation. This leads us to select a subset of SoaML constructs, which relieves the modeller from the work of choosing which one to use. We present the structure of the precise Service System Model by means of a metamodel shown in Fig. 1.

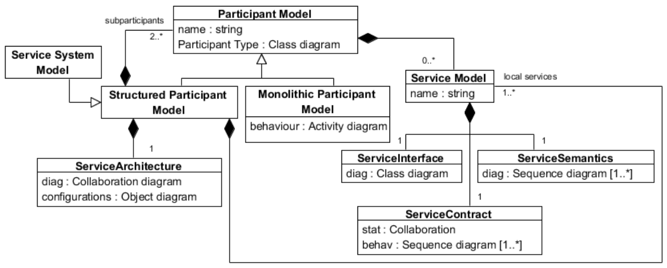

subparticipants
2..*
Participant Model
name : string
Participant Type : Class diagram
0..*
Service System Model
Structured Participant Model
Monolithic Participant Model
behaviour : Activity diagram
Service Model
name : string
local services
1..*
1
ServiceArchitecture
diag : Collaboration diagram
configurations : Object diagram
1
1
ServiceInterface
diag : Class diagram
ServiceSemantics
diag : Sequence diagram [1..*]
1
ServiceContract
stat : Collaboration
behav : Sequence diagram [1..*]

Fig. 1: PreciseSoa Service System Model: metamodel

PreciseSoa provides two different kind of models for structured and monolithic participants; both offer the possibility to model the services required and provided by a participant, whereas for the structured participants it is possible to define the subparticipants and how they are organized into a service architecture, and for the monolithic participants instead it is possible to define their behaviour.

Recall that a service system is a structured participant neither offering neither consuming services (and thus without ports). Thus, we consider that a service system model is a special case of a participant model. A participant model includes a number of service models, one for each of the services that the participant consumes and provides. A structured participant model includes also some participant models (one for each kind of its subparticpants) and a service architecture. The behaviour of a monolithic participant may be depicted by an UML activity diagram.

The service architecture presents how the subparticipants of a particpant P interact among them using the services they provide and use, in the meantime they allow P to provide its services and obviously they use the services P uses.

A service model describe a service; the service interface define the messages together with the associate message data needed to use the service, distinguished in in and out, the service contract illustrates the allowed sequences of messages that can be received sent by the service itself, and the semantics the effect of the in messages on the service realm and how the realm itself influence the out messages.

In the case of a structured participant the models of its subparticipants are also part of its model.

For space reasons we give here some examples of well-formedness constraints on the elements of the model, such as: *A service interface uses and realizes two interfaces (one is the type of the role of service consumer and another is the type of the role of service provider); The interfaces realized and used by a participant must be attached at its ports.; The bindings between parts and collaboration uses must be labelled by the nouns of the roles.*

A. Service Model

A precise service model based on SoaML consists of a *name*, a *Service Contract*, a *Service Interface*, and a *Choreography*. A Service Contract specifies an agreement between the involved participants on how the service is provided and consumed. A Service Interface defines the roles of the participants through the interfaces and an associated choreography.

We present the structure of the precise Service Model by means of a metamodel shown in Fig. 2.

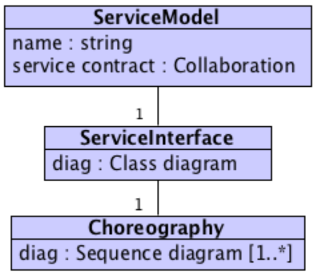


Fig. 2: Precise service model with SoaML

1) Service Contract: A Service Contract is a UML collaboration stereotyped by <<Service Contract>> and named as the service. This collaboration has exactly two parts corresponding to the roles of the provider and the consumer.

Fig. 3 shows a generic service contract. The dashed oval is the icon of the collaboration, whereas the inside boxes represent the collaboration parts and are used to model the roles of who provides and of who consumes the service (the stereotypes <<use>> and <<provide>> allow to distinguish the two roles). The parts are typed by interfaces. Thus, in Fig. 3, Prov1 is the role for provider and the Provider Interface1 is the interface that it implements to play that role, whereas Cons1 is the role for consumer and Consumer Interface1 is the interface that it implements to play that role in Serv1. The two parts are connected by a UML connector, to emphasise that they will communicate.

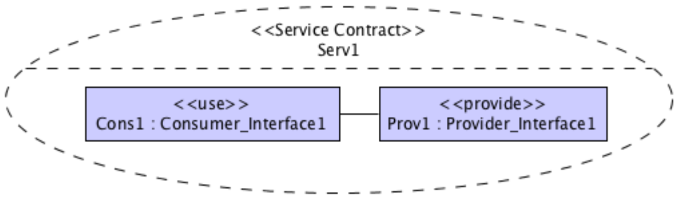


Fig. 3: A generic Service Contract

Fig. 5a shows the Place Order service contract. This diagram states that the role for the participant using the service is named buyer and is typed by the interface OrderPlacer, whereas the role for the participant offering the service is named seller and is typed by the interface OrderTaker. Those parts are bound to fulfill the Place Order service by the service contract.

2) Service Interface: A Service Interface is defined by a class stereotyped <<Service Interface>> and named as the service itself. It should realize and use respectively the interfaces of the correspondent provider and consumer. Fig. 4 shows a generic service interface.

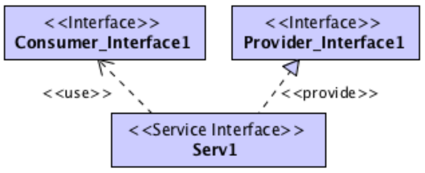


Fig. 4: A generic Service Interface

The class stereotyped by <<Service Interface>> should realize the provider interface (represented by the UML realiziation symbol: the dashed arrow with closed head) and use the consumer interface (represented by the UML dependency: the dashed arrow with open head)

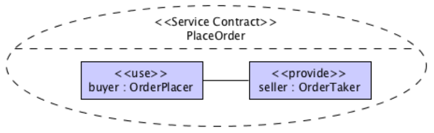
The operations of the interfaces correspond to the messages exchanged between the service and the participants using and providing it.

We use <<MessageType>> (a stereotype of datatype) to stereotype the type used by the interface operations. Message types contain attributes that have primitive type, datatype, or other message types. The definition of the message types should be given together with the interface. Our method suggests all the information exchanged between parts must be contained in the parameters of the operations of the interfaces, and that such parameters will be typed by message types; moreover the operations of the interfaces cannot have a return type.

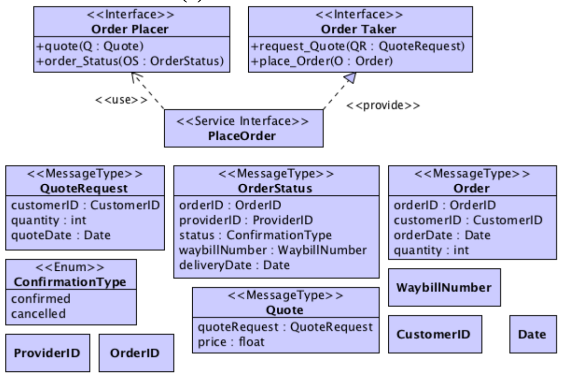
Although UML offers the possibility to hide the details of classes and datatypes, in our method we suggest to show the details of the various <<MessageType>> datatypes of the service.

In Fig. 5b we define two possible cases for an order: confirmed or cancelled that may be the values of attribute status typed in enumerated type ConfirmationType. Moreover, if the order is accepted, the buyer will receive further information about the order by other attributes of datatype OrderStatus, they are the providerID of the order, the delivery date of the shipment, and the waybill number of the shipment. The identification of the buyer is defined by the attribute customer ID in the definition of datatypes QuoteRequest and Order.

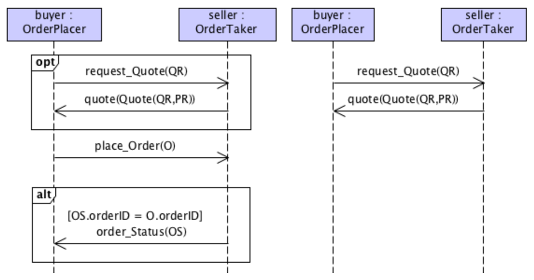
Fig. 5b shows the interface of service Place Order. It realizes and uses respectively the interfaces OrderPlacer and OrderTaker, which define the operations that they implement to play their own roles. The type of the provider role OrderTaker is the interface that the provider seller will implement on a port to provide this service. The type of the consumer role OrderPlacer is the interface that buyer will implement.



(a) Place Order Contract



(b) Place Order Interface



(c) Place Order Choreography

Fig. 5: Precise SoaML model of the Place Order Service

3) Choreography: A Service Interface is associated with one choreography that is defined by a set of UML sequence diagrams having exactly two lifelines (one for the service provider roles and one for the service user role).

The service interface introduces the operations that the participants implement to play the provider and consumer roles, however, it does not represent how the participants work together. So a choreography is used to show how the participants exchange the messages to enact the service. The sequence diagrams, included in the choreography, show examples of use of the service but are not a complete exhaustive specification.

SoaML allows to use also the activity diagrams to represent the choreography, but in our opinion, they are not as much as expressive and precise as the sequence diagrams.

Fig. 5c shows the choreography of service Place Order, consisting in two sequence diagrams. Sending a quote request is an option for the buyer, therefore we built two sequence diagrams, the sequence on the right illustrates the case when the buyer just requests the quote, and after does not place any order. The sequence on the left shows the cases where either requesting before a quote or not, the buyer place an order. **alt** is the UML combinator corresponding to the guarded alternative: the enclosed messages will be part of the traces determined by the sequence diagram only if the guard is true. **opt** is the optional combinator, therefore its enclosed messages may or may not be in the traces of the sequence diagram. The service provider and consumer must abide to that choreography when offering and using that service.

Our precise approach also includes a set of well-formedness constraints on the service model to guide the development of good quality models and avoid frequent mistakes. For space reasons we just give an example of such a constraint on choreography: *All messages in the choreography sequence diagrams should be built by operations of the used and provided interfaces, and all these interface operations should appear at least in one of the choreography sequence diagrams.*

B. Participant Model

In PreciseSoa the participants of a service system may of different kinds and each kind is described by a specific participant model.

A participant model introduce a class stereotyped Participant, that will be used to type the specific participants (instances) of that kind, and that we will name participant class.

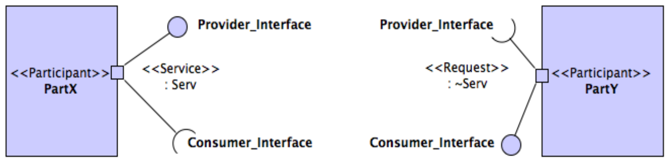


Fig. 6: A generic participant view

A participant is a service provider if it offers a service and is a service consumer if it uses a service. A participant may provide and consume any number of services. It means that the same participant may be "provider" of some services and a "consumer" of other. The UML mechanism of the ports is used to indicate the points of interaction through which participants interactwith each others to enact services, and the needed ports are added to the participant class. There are two kinds of port that a participant class may have, one is stereotyped by Service known as a service port where a service is provided by participant of this type, one is stereotyped by Request where a participant makes a request for service from other participants. A port is then typed by a service interface. A service port has the type of the relevant service interface and the request port has the type of the conjugate service interface.

All the models of the subparticipants of a structured participant are collect in a participant view.

Fig. 6 presents a generic participant view including two participant classes. The instances of the participant class PartX are the provider of Serv and thus the class has a Service port, typed by the conjugate service interface of Serv, denoted by Serv. Participant class PartY types the consumers of this service and has a Request port. The port of PartX provides the Provider Interface interface and requires the Consumer Interface interface of Serv, and vice versa for PartY. These ports are the points for engaging two participants PartX and PartY to enact Serv.

All the kinds of participants that provide and consume services in the DealerNetworkingSystem are presented by participant view show in Fig. 7 by just giving their participant classes (here we do not further model these participants, since we not even define if they are structured or monolithic). There are three kinds of participant: Dealer, Manufacturer, and Shipper.

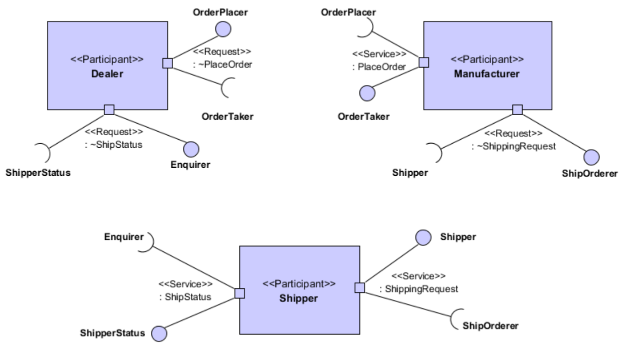


Fig. 7: DealerNetworkingSystem participant view

C. *Service Architecture*

A *service architecture* is defined by a UML collaboration with stereotype ServiceArchitecture, as shown in Fig. 8, and by a set of *architecure configurations*. A service architecture consists of a set of services and a set of (roles for) participants that work together by providing and consuming services for some business goals.

A service is provided and consumed by two participants, and we use the wording "the service is used" or "service usage" to indicate it, this associated relationship is represented by a collaboration use of the collaboration part of the service contract (such as S1:Serv). There may be several usages of the same or of different services in a service architecture, and each of them involves a possible different set of roles (and of the related connectors). The name of a service in a collaboration use is structured of S: Serv, where S is optional (as in our example in Fig. 9) and can be a short name for service Serv, or be the same as Serv (Serv: Serv for example) as in SoaML examples (c.f, [2]), and Serv is the name of the service, suggesting its purpose. This structure abides the name of a UML collaboration element.

A participant role in the service architecture is displayed as a UML part (a solid rectangle) that contains the role name and the participant class typing the role, i.e., X : PartX, where X is the role name and PartX is the type of this role. The two participants, who play the providing and consuming roles in a service, will be defined in the contract of this service. The roles that the two participants play in a service usage, i.e., who is provider and who is the consumer, are represented by the labels on the dashed line connecting the parts and the collaboration use.

Fig. 8 shows a generic service architecture for a service system, i.e., a structured participant without any port, and thus unable to interact with the outside by means of service calls.

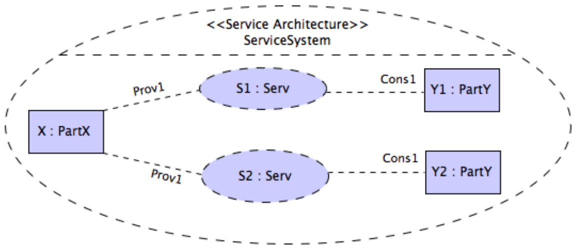


Fig. 8: A generic service architecture for a service system

The service architecture of the DealerNetworkingSystem is shown in Fig. 9 and in 10 (where an architecture configuration is shown). The DealerNetworkingSystem architecture depicts a community of participants providing and consuming services for realizing the aims of the DealerNetwork.

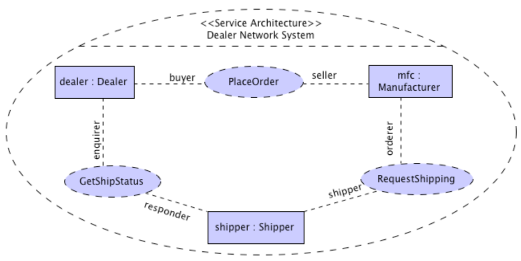


Fig. 9: DealerNetworkingSystem Service Architecture

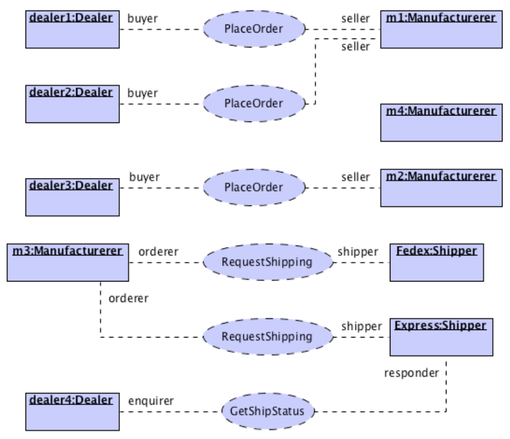


Fig. 10: A DealerNetworkingSystem Architecture Configuration

Fig. 9 illustrates the possible roles for participants in the high level view of how they work together in the DealerNetworkingSystem. The three services and the participants appearing in this diagram will be described by service and participant models in the following sections. Fig. 10 shows instead a possible configuration of this architecture, where several participants of several types play various roles using and providing services; notice how at the same time several participants may use some service, offered by the same or by different other participants.

In case of a structured participant P offering and using services in the service architectures there will be also special parts, denoted by dashed boxes, representing the roles of the those using or providing services to P.

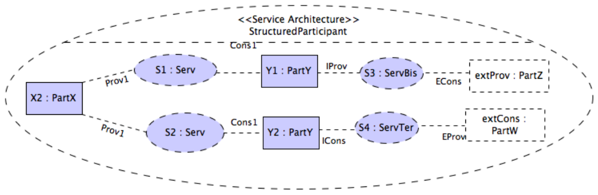


Fig. 11: A generic service architecture for a structured participant

The service architecture of a generic structured participant is illustrated in Fig. 11. In this service architecture, the roles of other participants are demonstrated by the roles with dashed outlines (i.e., X : Part X), whereas the roles played by subparticipants within structured participant are normal roles.

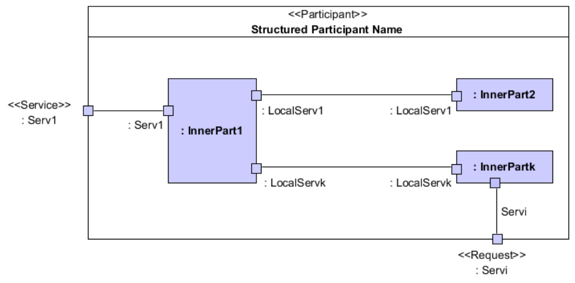


Fig. 12: A generic structured participant

An architecture configuration shows a snapshot of the architecture at a specific point in time. A configuration is presented by a UML object diagram. It includes a set of participant instances and the services that they offer and use at that particular time. Each service is is represented by a use of the collaboration part of the definition of its contract. The links among them are bindings of the collaboration.

A structured participant class, say PC, should be a UML structured class having as subclasses the participant classes typing its subparticipants; all these classes will have their respective ports. There will connectors between the ports of the subparticipant classes and between the ports of the subparticipant classes and the ports of PC. The class in Fig. 12 is a generic structured participant class.

III. RELATED WORKS

In terms of modelling services for the design of SOA systems, several proposals using UML have been developed, e.g, UML4SOA developed in Sensoria project [4]–[6]. The idea for introducing a semantic view of a service, not present in other approaches, was prompted by an interesting report of the Sensoria project [7].

SRML a formal semi-visual notation proposed in the Sensoria project [8], [9]. SRML [10] is based on a different view about SOA requiring e.g., to distinguish the interactions among modules (components) into business protocols, layer protocols and interactions protocols, whereas in SoaML there is a unique type (the contract between who offer a service a who uses it).

In [11], the definition of SOA is extended and a lightweight formal framework for capturing SOA main components is proposed, based on a critical assessment of existing design formalization techniques in object and component-oriented programming domains.

A recent formal approach given in [12], [13] uses priced-timed automata and address the formal verification of functional, timing and resourcewise correctness.

At last, the protocol of the services that is specified at precise level of our models shall be effective in refactoring services to maintain service systems for future change. This makes agility, one of the most important capabilities of SOA-based system, to be taken in to account.

IV. CONCLUSION

We have described our precise method for modelling service systems with SoaML and illustrated it on a case study. The method applies to a widespread notation, involves a precise definition of the resulting models by a metamodel and its associated constraints that guarantee a good level of quality by construction.

In this paper, we have considered the modelling of the services and other aspects of a SOA-based system such as service architecture, participants, and configurations. With model-driven approach, we wish to combine our method with formal specification methods for extending the work towards automatic transformations into application fragments.

REFERENCES

[1] T. Erl, *SOA Principles of Service Design*. The Prentice Hall Service-Oriented Computing Series from Thomas Erl, 2007.

[2] OMG, Service oriented architecture Modeling Language (SoaML) Specification for the UML Profile and Metamodel for Services (UPMS) V. 1.0. December 2009.

[3] C. Choppy, G. Reggio, and K.-D. Tran, “Formal or not, but precise modelling of services with casl4soa and soaml,” in *4th International Conference on Knowledge and Systems Engineering (KSE12)*, Aug. 2012, pp. 187–194.

[4] P. Mayer, N. Koch, and A. Schroeder, “The UML4SOA Profile,” Ludwig-Maximilians-Universitaet Muenchen, Tech. Rep., July 2009.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 07, July-2019

[Online]. Available: http://www.uml4soa.eu/wp-content/uploads/UML4SOA.pdf

[5] N. Koch, P. Mayer, A. Shroeder, and A. Knapp, “The UML4SOA Profile,” Ludwig-Maximilians-Universita ̈t Mu ̈nchen, Tech. Rep., 2010.

[6] H. Foster, L. Go ̈nczy, N. Koch, P. Mayer, C. Montangero, and D. Varro ́, “UML extensions for service-oriented systems,” in *Results of the SEN- SORIA Project*, ser. LNCS 6582. Springer, 2011, pp. 35–60.

[7] L. Bocchi, A. Fantechi, L. Gonczy, and Nora, “Sensoria Ontology,” Sensoria, Munich, Germany, Tech. Rep. D1.1a, 2006, available at www.sensoria-ist.eu.

[8] Software Engineering for Service-Oriented Overlay Computers, “IST project founded by EEC,” 2009, www.sensoria-ist.eu.

[9] M. Wirsing and M. M. Ho ̈lzl, Eds., Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing, ser. LNCS 6582. Springer, 2011.

[10] J. L. Fiadeiro, A. Lopes, L. Bocchi, and J. Abreu, “The Sensoria Reference Modelling Language,” in *Results of the SENSORIA Project*, ser. LNCS 6582. Springer, 2011, pp. 61–114.

[11] A. Yanchuk, A. Ivanyukovich, and M. Marchese, “A lightweight formal framework for service-oriented applications design,” in *ICSOC*, 2005, pp. 545–551.

[12] A. Causevic, C. Seceleanu, and P. Pettersson, “Modeling and reasoning about service behaviors and their compositions,” in *Proc. of 4th Int. Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2010), LNCS 6416*. Springer, 2010, pp. 82–96.

[13] A. Causevic, “Formal approaches to service-oriented design: From behavioral modeling to service analysis,” Licentiate thesis, June 2011.