SOA-CoM: Building a Correct by Design Service Oriented Architectural Style
Supporting Structural and Non-functional Properties

Imen Graja, Imen Loulou and Ahmed Hadj Kacem
ReDCAD Laboratory, Department of Computer Science, University of Sfax, B.P. 1088, 3018 Sfax, Tunisia

Keywords: Correct by Design, Service-oriented Architectural Style, Communication Schemas, Formal Specification.

Abstract: As a piece of software continues to evolve, it inevitably becomes more complicated and harder to understand, maintain, reuse, evolve and improve. Software architecture has emerged as a solution to these issues particularly for complex systems. Having a correct software architecture is critical to the success of the design and the development of a system. In order to design a correct software architecture the concept of architectural styles is used. In this paper, we propose SOA-CoM, a formal approach for the correct modeling of service oriented architectural styles. We specify a set of communication schemas that define SOA structural and interaction properties. These schemas are modeled as UML graphs. In order to reuse them and to build the style, we define composition rules that can be applied to them. A software architect can then extend the designed style with non-functional properties (NFP) using extension rules. To ensure design correctness, we specify these communication schemas using the formal language ASL (ArchWare Style Language). All specifications are implemented and checked using the ASL Toolkit.

1 INTRODUCTION

The ever-changing nature of business requires that software systems continue to constantly evolve. As a software system changes, its design can be analyzed from an architectural style perspective to serve as a basis for static system analysis. An architectural style is an approach that simplifies software design and reuse by capturing and exploiting system design knowledge (Monroe et al., 1997). Traditionally, the primary focus of reuse research has been on the reuse of software entities, such as objects or components. In most cases, the reuse was at the code-level, even though these are not the only entities that can be successfully reused. Furthermore, there have been significant improvements in reuse technology and methods at more abstract levels. At this level of abstraction, key issues include the definition of a methodology to design architectural styles that support design fragments for reuse-driven implementation.

Defining a new style, however, is not an easy task. In particular, software architects may not be skilled enough to build correct styles. So the question that arises is: How can we ensure that the designed style is correct? To answer this question we require new foundations that enable architects to reason about and plan based on reuse-driven and correct by design development at an abstract level.

We are interested in a service-oriented architecture that becomes increasingly popular for complex and dynamic distributed web applications. The building of a software solution based on SOA typically requires combining existing services using techniques like choreography and orchestration.

This paper aims to define a formal approach that supports the correct modeling of SOA styles. This approach includes all relevant mechanisms of service publication, discovery, connectivity and composition and provides a modeling solution based on a set of communication schemas that we propose. These schemas integrate structural and interaction properties of SOA. By defining these schemas our approach not only provides the opportunity for reuse at an abstract level but also provides a formal guarantee of correctness of representing service-based styles. We also elaborate composition rules that have to be applied to these schemas to build the desired architectural style. The communication schemas and the designed style are specified formally in ASL (ArchWare Architecture Style Language) (Leymonerie, 2004). ASL is a formal language that allows style formalization, defines families of architectural elements and expresses

Graja I., Loulou I. and Hadj Kacem A..
SOA-CoM: Building a Correct by Design Service Oriented Architectural Style - Supporting Structural and Non-functional Properties.
DOI: 10.5220/0004889001720179
In *Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2014), pages 172-179
ISBN: 978-989-758-030-7

related constraints related to their structure and behavior.

The rest of this paper is organized as follows. We survey related work in section 2. Section 3 provides a general overview on the SO architectural styles. Sections 4, 5 and 6 present our SOA-CoM approach. They describe our methodology to design correct SOA styles and extend them with non-functional properties. In section 7, we present a travel reservation system case study to illustrate our methodology. Then in section 8 we prove the consistency of these specifications. Finally, section 9 concludes our work and discusses future works.

2 RELATED WORK

The work presented in this paper focuses on the general research area of architectural styles and the modeling of SOA styles in particular.

Shaw and Garlan in (Shaw and Garlan, 1996) define software architectural styles as a vocabulary of the type of components and connectors, and a set of constraints on how they can be combined. Further, they codify structural, behavioral, interaction, and composition guidelines that are likely to result in software systems with desired properties. We base our work on these same concepts and aim to define an architectural style modeling approach that allows software architects to work at a much more abstract level.

There are several architectural styles; the pipe and filter, client-server, peer-to-peer, publish/subscribe and service-oriented architecture.

One of the benefits of SOA is that it improves reusability in a distributed environment with independent services. Thus, the main components of SOAs are services that provide operations (software functionalities) to their consumers. In order to publish services and to allow these consumers to access to them, a service registry must be used.

A lot of works have been published in the area of the modeling and validation of SOA applications (IBM, 2005) and, SOA Meta-Model based on the UML profile and Pattern-Oriented SOA model (Baresi et al., 2006; Wada et al., 2011; Amir and Zeid, 2004; Sanz and Marcos, 2012). However, all these models are not abstracted as an architectural style. Although, Tang et al (Tang et al., 2010) formally define Enterprise SOA style model which is a special type of SOA for enterprise.

The proposal of Baresi in (Baresi et al., 2006) defines the SOA style as graph transformation systems from the abstract business level style to the SOA specific-style. Nevertheless, these approaches are especially interested in service discovery mechanism and they do not focus on service composition (choreography and orchestration).

In the same context, Wada et al (Wada et al., 2011) describe a solution based on UML profile which considers only non-functional aspects while Amir and Zeid in (Amir and Zeid, 2004) define a SOA modeling approach that considers functional and non-functional aspects. Moreover, Lopez-Sanz and Marcos in (Sanz and Marcos, 2012) define a framework called ArchiMeDes that follows the MDA approach for the specification of software architectures using the concepts behind SOA. These works center their attention in the definition of the architectural model and they do not support service discovery mechanism. In (Zhao et al., 2010) Zhao et al define SOA architectural styles based on graph grammars but, this approach considers only the concept of service composition.

Other works that focus on correct modeling by design of architectural styles like P/S-CoM (Loulou et al., 2010). This work is interested in formal modeling of event-based style using the communication schemas coded in Z notation. Once the style is obtained the behavioral properties can be added like presented in (Krichen et al., 2012). This work proposes P/S-CoM+ approach as an extension of the P/S-CoM with behavioral properties. We are inspired by these approaches to define a correct modeling of SOA styles.

To conclude our literature review, we can say that there is a need to complement these contributions by taking into account the service compositional aspect, dynamic service discovery and correct modeling by design. The approaches mentioned above are interesting and helpful in defining and achieving our goal. We develop a visual and formal approach which aims to help an architect design correct SOA architectural styles based on a set of reusable communication schemas and composition rules. The designed style can later be extended using extension rules with non-functional properties.

In general we are not in an ad-hoc solution that represents an arbitrary solution where the architect will be free in its design. However, our methodology guides the architect to have as a result an SOA style. Such approach cannot be considered as an arbitrary solution. In particular our solution ensures the correctness and the reliability of the designed style using the composition rules and the proved communication schemas that integrate the SOA properties. This makes our work considered as correct-by design approach.

3 SERVICE-ORIENTED ARCHITECTURAL STYLE

Service-oriented architectures are an important kind of dynamic architectures (Baresi et al., 2006). The service is their central concept. It supports different roles depending on the functionality given by the system. Moreover, depending on the atomicity of the service, it can be classified as a simple service (atomic service) or a composite one.

There are two types of service composition approaches. One approach is the choreography and the second one is the orchestration that defines one service which assumes the coordination role named "Orchestrator".

Additionally, services need user invocation to interact with each other or to modify their behavior (Sanz et al., 2008). To that end, the business process must be initiated by the user which can be an application front-end, a web portal or another service. For this reason, our approach contains a component representing the user of the application. This component can have the role of a consumer or consumer and provider in the same time.

Since service consumers can invoke services directly or through an intermediary. We define a broker that can be used as an intermediary between consumers and their services. This component performs tasks needed to convert the data format, route messages and save data as a buffer (Endrei et al., 2004). This broker (resp. network of brokers) can build distributed applications that require asynchronous, multi-cast interaction like publish-subscribe models (Cugola and Nitto, 2008) in SOA.

Services communicate and interact with each other through an asymmetric connection based on request/reply message exchanges. So, we choose to represent this connector by a straight line (using UML connector) used for call-return style. However, we give for the composition link a dependency notation between required and provided interface that was represented by a lollipop notation (UML1).

To make services accessible to users; they must be published, then searched in registries. Components and service discovery can interact in either a pull or push mode. In (Zisman et al., 2008), the author supports a proactive runtime service discovery using pull and push modes. On the one hand, the registry can be passive and allow only service publication thus requiring the consumer to support pull runtime service discovery. On the other hand, the registry can be active so the client can be notified when a required service is published (Cugola and Nitto, 2008). In both cases, the register can allow both active and passive discovery (pull and push). As we have already mentioned, interactions between components and service discovery are specified for architectures that can use a centralized or a distributed registry. In this context, there are various approaches that have been defined concerning service publishing and discovery model. In (Verma et al., 2005; Yulin et al., 2010) the authors describe a peer-to-peer network of UDDI registries like MWSDI that was built by the Meteor-S project. However, other approaches presented in (Cugola and Nitto, 2008; Baresi and Miraz, 2006) support a distributed service registries using a publish/subscribe network. Hence, we determine that the distributed service discovery can follow either a peer-to-peer topology or a publish/subscribe one.

4 A METHODOLOGY TO DESIGN A CORRECT SOA STYLE

In this section, we define a set of reusable and simple communication schemas that integrate structural and interaction properties of SOA. They can be composed to have as a result a correct service-oriented architecture style. Next, this style can be extended by non-functional properties using extension rules. These steps are represented in fig.1. In order to ensure the correctness of the designed style, some properties have to be defined.

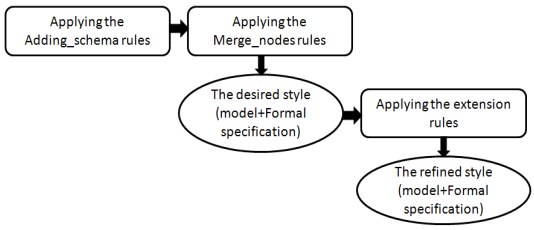

Figure 1: The steps for designing a SOA style.

4.1 Structural and Interaction Properties

The communication schemas represent the basic blocks that can be composed to build correct SOA styles. They integrate structural aspects and the properties of interaction between the components. These properties are defined as follows: 1)The service must be published in a registry service, after being created. 2)The client discovers the desired service through a registry service. 3)The client receives and saves the description of the relevant service. 4)The communication with the service discovery is established

through the push and/or the pull mode. 5)The distributed service discovery can follow either a peer-to-peer or publish/subscribe topology. 6)Services can be composed using orchestration or choreography. 7)With the central composition, all the communications between services must be through an orchestrator. 8)The distributed composition is based on P2P style that can be classified into three sub-styles: a static, mobile or hybrid (Dillon et al., 2007) 9)The communication between services must be triggered by the consumer. 10)The consumer communicates directly or through a broker with services. 11)The broker utilizes a callback or publish/subscribe interaction mechanisms. 12)The broker-based communication can be through a single broker or through a distributed number of brokers. 13)The interconnection topology of the distributed brokers can be peer-to-peer or hierarchical. 14)In hierarchical structure, the communication between brokers can be facilitated by their parents broker where sub-brokers are controlled by the parent broker. 15)SOA supports bidirectional communication between services. 16)The consumer can interact with more than one component.

4.2 Non-functional Properties

Most of the existing works deal with the modeling of functional aspects and a few of them consider non-functional requirements in their proposals. In order to induce certain desired properties that can be functional or non-functional (Sterritt and Cahill, 2008), architectural styles place certain constraints. However, each instance of a software family will have similar constraints but different non-functional requirements. Therefore after designing a style, it can be refined simply by adding non-functional properties (NFPs).

The semantic of messages transmission and message processing can be specified for each connector (Wada et al., 2006) through three properties. The first property is the synchrony semantic of message transmissions between the source and its destination that can be: synchronous or asynchronous. The second property is the assurance level of message delivery where three different semantics are defined: at most once, at least once or exactly once. Finally, to specify how to order messages, we extend the style with the following policies: FIFO, LIFO, highest priority first or earliest deadline first.

These properties- synchrony, the assurance level of message delivery and how to order messages- are deductible from the experimentations and from our research in the related works. They can be integrated in the designed style using extension rules that were defined later.

4.3 Generic Communication Schemas

We elaborate a reusable set of proved communication schemas which are used to integrate structural and interaction properties of SOA. They facilitate the task of the architect because they can be used for reuse and they ensure the compliance with these properties.

The communication schemas are characterized according to different phases: the publication phase, the discovery phase, the invocation phase and the composition phase.

To define our proposal, a service is described with a node that contains a stereotype ≪Service≫ and three labels: Role, Type and Var (Variable: service instances). First, the label Role can be either ”Prov” (Provider) that denotes a set of operation being executed to achieve the system functionality, or ”Orchest” (Orchestrator) which represents the process responsible for service composition.

In addition, a service discovery, a composed service and a broker are represented with nodes that contain respectively the stereotype ≪ServiceDiscovery≫, ≪ServiceComp≫ and ≪Broker≫ and two labels: Type and Var. A service discovery can also be found in a distributed environment but in this case, it is described with a stereotype ≪DistServiceDiscovery≫ and with tree labels: Type, Var and Topology that can be either P2P or P/S.

Further, a component that represents the application user can be specified with a stereotype ≪Component≫ and three labels: Role, Type and Var. In this case, two possible roles were identified: Consumer (Cons) and Provider/Consumer (PrCons).

Each component contains two types of ports: the find and publish port that are defined to the centralized or a distributed service discovery and, the provided and required port that are described for other components.

In order to characterize communication schemas, we model interactions between components according to each phase in the next section.

4.3.1 The Publication Phase

In the publication phase, we define interactions between services and a service discovery through the push mode. In this case, the service is the initiator of the communication. It can have the provider role, the orchestrator role or it can be a composed service. In addition, these services can publish their information to a centralized service discovery or a distributed one through the push mode from services to the service discovery. We define six communication

schemas, three of them with centralized service discovery named Re_pub_S and the others with the distributed one named ReD_pub_D. For example, fig.2 represents the communication schema between a service that has the role Prov and a single service discovery.

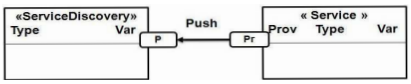

Figure 2: The communication schema Re_pub_S.

4.3.2 The Discovery Phase

In the discovery part, we define two communication links between components: pull and push. First, we can establish the push mode between the service discovery and components (like the service that has the role Prov or Orchest, the composed service, and component that has the role Cons or PrCons). In this case, the set of communication schemas is named Re_push_C. Likewise, the same mode of communication is used between these previous components and the distributed service discovery. Here, the set of communication schemas is named ReD_push_C. In fact, with the push mode, the initiator of the communication is the centralized service discovery or the distributed one. In addition, we can attribute the pull mode between the service discovery and the above components that play the role of the initiator of the communication. Consequently, we have defined two sets of communication schemas. The first set is named Re_pull_C with a centralized service discovery and the second one is named ReD_pull_C with a distributed service discovery. Finally, between the above elements we can consider these two links "pull and push" involving one communication schema. To illustrate, a communication schema from the set Re_pull_push_C with a composed service and with a service discovery using the pull and push mode is depicted in fig.3. Moreover, we develop a set of schemas with a distributed register named ReD_pull_push_C. Thus, we declare fifteen communication schemas with centralized service discovery and fifteen others with distributed one.

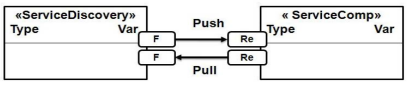

Figure 3: The communication schema Re_pull_push_C.

4.3.3 The Invocation Phase

In this part, we take into consideration the fact that the communication between users and services can be done directly or through a broker. Services (that have the role Prov, Orchest or that can be a composite service) communicate with their users (Components that can have the role Cons or PrCons) through a connection based on bidirectional message exchanges. In this context, we define nine communication schemas with direct invocation and ten others with brokered one. Thus we characterize, in the fig.4, an example of a communication schema of direct invocation with a consumer and with a service orchestrator named Li_Dir_CS. In this set, the requestor is the consumer (resp. PrCons). But when this latter is the provider of information then we elaborate another set of communication schemas named Li_Dir_SC.

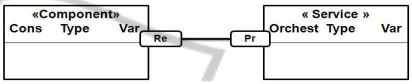

Figure 4: The communication schema Li_Dir_CS.

Fig.5 illustrates an example of communication schema Li_Ind_CB using a Broker and a consumer. The difference between the set Li_Ind_CB and the set Li_Ind_BC is that the broker in the first example represents the requester of information. However, in the second one, it represents the provider of information.

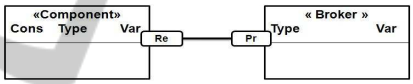

Figure 5: The communication schema Li_Ind_CB.

4.3.4 The Composition Phase

As we have previously stated that there are two different types of composition. For each type, we define their corresponding communication schemas. In the context of an orchestration, we define five communication schemas which represent the interactions between a service orchestrator and other services (service that has the role Prov, another orchestrator or a composed service). When the requester is an orchestrator, then the set of communication schemas is named Orchest_SOC. However, when the orchestrator represents the provider of information, then the set of schemas is named Orchest_CSO. Fig6 illustrates an example of communication schema from the set Orchest_SOC that shows the interaction between an orchestrator and a service which has the role "Prov".

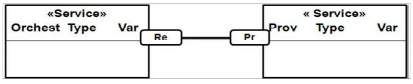

Figure 6: The communication schema Orchest_CSO.

In the context of a choreography, we presented only two communication schemas named Chor_SCS

that contain a composition link between a composed service and either a service having the role Prov or another composed service. Fig.7 represents an example of a Chor_SCS with a service provider.

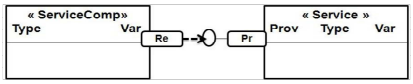

Figure 7: The communication schema Chor_SCS.

Each of these communication schemas is formally specified using ASL (ArchWare Architecture Style Language) which allows the definition of architectural styles. As illustration, we represent the Re_pull_C specification.

```
Re_pull_C is style where {
constructors{
Re_pull_C is constructor( consumer: Alias,
 discovery : Alias,c : set[ Component ],
 d: Component, pullrc: Connector);{
   d is discovery;i is consumer;
   iterate c by i : consumer do {
     new pllrc;
        attach i~RE1~input to
        pullrc#last~comp_conn_port~output;
        ........}}}
constraints {
only_Cons_SP_SO_SC is constraint {..},
only_pull_connector is constraint {..}}}}}
```

The ASL language has the ability to specify structural and behavioral constraints. So all the properties are defined in the ”constraint” part in the ASL code. Finally, to design a correct SOA style and to have a formal specification of the designed style, we must define some composition rules.

5 COMPOSITION RULES

Based on communication schemas, we construct the global designed style by applying some composition rules. These rules ensure the correctness of the defined properties and help the architect to instantiate and to compose the selected schemas to have as a result the designed style. This style meets the requirements of the user and respects the correctness of SOA properties.

- **Adding_schema:** in this step, the architect chooses the adequate communication schemas that meet their needs. For each node, the architect can modify the instantiable labels like Type, Var (variable) and the Topology that can be either P2P or P/S with a distributed service discovery. In addition, he must rename the pull and push mode used with keeping the prefix Pull or Push depending on the mode. For example, the link ”Pull” can have the name ”Pull_CD”.
- **Merge_ nodes:** In order to be merged, two nodes must have the same type and role. They must also communicate in the same manner with other components. In particular, the nodes that contain the stereotype ≪DistServiceDiscovery≫ must have the same topology. In other words, giving two communication schemas, we can merge their consumers (resp. PrCons) and their services which have the role Prov (resp. Orchest) only if they have the same role and type. But, if they are Borkers or ServicesDiscovery they must have only the same type to be merged. In the same way, we can apply this rule on distributed service discovery if they have the same type and the same topology. In all these cases, it was necessary to make sure that nodes which are connected to the same type of component will be merged only if they interact with the same types of links. To illustrate, in fig.8 we apply the rule Merge_nodes on the service discovery nodes from communication schemas Re_pull_C and Re_pub_S.

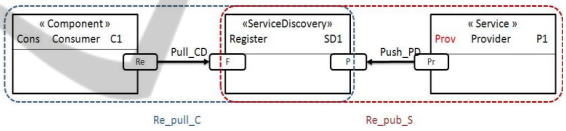

Figure 8: Graph obtained by composition of two communication schemas.

6 EXTENSION RULES

Once the style is designed progressively, the architect can extend it by adding non-functional properties. As a result, a new style can be formed by adding these properties to an existing style. This section makes our methodology to design a correct style more extensible and facilitates reuse. Then, a style can be described relative to its parent by applying one or more extension rules.

- **Synchrony_Ext:** this rule extends one connector in the style with the synchrony property.
- **DeliveryAssurance_Ext:** Each connector also can be extended with delivery assurance semantic by applying this rule.
- **OrderingPolicy_Ext:** During a communication between two components, a queue is used for storing messages in storage. So by applying this rule, we specify the connector’s ordering technique.

7 CASE STUDY

To illustrate our work, we use a simple travel reservation system as represented in fig.9. This system involves four different participants: client wishes to use the travel agency service that in turn, contacts hotels, to perform its work. In fact, the hotel service is responsible for reserving the available rooms. Finally, the last participant is the register that contains all service descriptions. In this example, the travel agency is a composed service and we choose to define a register as a centralized service discovery.

Figure 9: The travel reservation system.

According to our methodology, in order to design a correct SOA style, we start by applying the first composition rule named Adding_schema. This rule allows the architect to choose and to instantiate all necessary schemas according to each phase.

- Publication Phase: In this phase, two communication schemas are chosen from Re_pub_S. The first one is represented with a composed service and the second one is picked with a service that has the role Prov. These schemas integrate the first and the fourth property.
- Discovery Phase: from Re_pull_C, two schemas are selected. One of them is represented with a consumer and the other one is picked with a composed service. In this phase, we promote the second and the third property.
- Invocation Phase: the architect chooses only the direct communication between the user (client) and the travel agency service. So, he instantiates the communication schema, from the set Li_Dir_CS, with a consumer and a composed service. This schema integrates the ninth and the sixteenth property.
- Composition Phase: the eighth and the fifteenth property was depicted in the following added schemas. The service choreography is represented by the schema Chor_SCS using the hotel as an atomic service and the travel agency as a composite service.

Next, we apply the Merge_nodes rule as explained previously. In the ASL code we were just calling each communication schema constructor used in the previous step. Then, we define, in the constructor parameters, the necessary values. Finally, we obtain our global style represented in fig.10 and its associated code ASL.

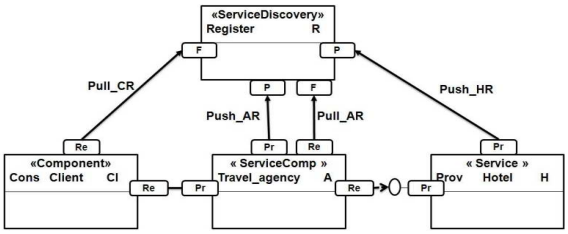

Figure 10: Graph obtained by composition of communication schemas.

To summarize, we provide to the architect the possibility to build a correct by design SOA styles that were generated by applying our methodology based on both semantic and visual representation. This style integrates structural and interaction properties. Next, to extend it with non-functional properties, we apply the extension rules. For example, the architect can specify the synchrony semantic of message transmissions between the client and the service "Travel agency" that can be in an asynchronous way. And it can specify the at least once as delivery assurance semantic with applying the Synch_Ext and DeliveryAssurance_Ext rules.

8 ANALYZING ARCHITECTURAL STYLE

The supported analysis includes checking the consistency (Mateescu and Oquendo, 2006). In Archware, The analysis covers the description and verification of properties using the Archware Architecture Analysis Language (AAL). It enables automated verification by model checking or theorem proving.

8.1 Checking the Consistency

A schema specification is consistent if at least one model exists. This instance must be conform to the schema that means it must satisfy its constraints. So this consistency can be checked simply using the property verification tool which is the Analyzer. To do so, it must ensure that all properties are verified. Otherwise, an error message is sent to the designer. So, if an instance of the schema is found, we say that the schema is consistent. But if no instance can be found the schema cannot be satisfied by any other model. According to this analysis, all schema specifications are consistent. While consistency checking formally establishes the existence of initial model for each schema specification, in most cases this is not as strong a check as we would like. We also need to show that all the composition rule specifications

preserve consistency. In order to accomplish that, we need to check certain properties of a model after the composition of schema instances.

9 CONCLUSION

In this paper we presented a formal approach for correct modeling of SOA styles named SOA-CoM. Our aim was to define a formal methodology based on a reusable set of communication schemas and composition rules that can be applied to generate the global architectural style. These schemas include all structural and interaction properties of an SOA. Then, we formally designed the NFPs of the SOA which can be integrated into the architectural style based on the extension rules. To ensure the style correctness, we specified the semantics of these schemas using the formal language ASL. As the style was designed based on the applied composition rules, the specification of this style will be built progressively in ASL. As an extension to our work, we can add other NFPs according to the user need in order to complement our approach. Then, our ongoing work will extend our SOA-CoM approach by reasoning about foundations regarding architectural evolution.

REFERENCES

Amir, R. and Zeid, A. (2004). A uml profile for service oriented architectures. In *OOPSLA Companion*, pages 192–193. ACM.

Baresi, L., Heckel, R., Th"one, S., and Varr'o, D. (2006). Style-based modeling and refinement of service-oriented architectures. *Software and System Modeling*, 5(2):187–207.

Baresi, L. and Miraz, M. (2006). A distributed approach for the federation of heterogeneous registries. In *ICSOC*, pages 240–251.

Cugola, G. and Nitto, E. D. (2008). On adopting content-based routing in service-oriented architectures. *Information & Software Technology*, 50:22–35.

Dillon, T. S., Wu, C., and Chang, E. (2007). Reference architectural styles for service-oriented computing. In *Proceedings of the 2007 IFIP international conference on Network and parallel computing*, NPC'07, pages 543–555, Berlin, Heidelberg. Springer-Verlag.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., and Newling, T. (2004). *Patterns: service-oriented architecture and web services*. IBM Corp. Riverton, NJ, USA.

IBM (2005). Ibm's soa foundation an architectural introduction and overview. Version 1.0.

Krichen, I., Loulou, I., Dhouib, H., and Kacem, A. H. (2012). P/s-com+: A formal approach to design correct publish/subscribe architectural styles. In Perseil, I., Breitman, K., and Pouzet, M., editors, *ICECCS*, pages 179–188. IEEE Computer Society.

Leymonerie, F. (December 2004). *ASL language and tools for architectural styles*. PhD thesis, University of Savoie.

Loulou, I., Jmail, M., Drira, K., and Hadj Kacem, A. (March 2010). P/s-com : Building correct by design publish/subscribe architectural styles with safe reconfiguration. *Journal of Systems and Software, 83(3):412-428*.

Mateescu, R. and Oquendo, F. (2006). Pi-aal: an architecture analysis language for formally specifying and verifying structural and behavioural properties of software architectures. *SIGSOFT Softw. Eng. Notes*, 31(2):1–19.

Monroe, R. T., Kompanek, A., Melton, R., and Garlan, D. (1997). Architectural styles, design patterns, and objects. *Software, IEEE*, 14(1):43–52.

Sanz, M. L., Acuña, C. J., Cuesta, C. E., and Marcos, E. (2008). Defining service-oriented software architecture models for a mda-based development process at the pim level. In *WICSA*, pages 309–312.

Sanz, M. L. and Marcos, E. (2012). Archimedes: A model-driven framework for the specification of service-oriented architectures. *Inf. Syst.*, 37(3):257–268.

Shaw, M. and Garlan, D. (1996). *Software architecture. Perspectives on an emerging discipline.* Prentice Hall Publishing.

Tang, L., Dong, J., Peng, T., and Tsai, W.-T. (2010). Modeling enterprise service-oriented architectural styles. *Service Oriented Computing and Applications*, 4(2):81–107.

Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., and Miller, J. (2005). Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. *Information Technology and Management*, 6(1):17–39.

Wada, H., Suzuki, J., and Oba, K. (2006). Modeling non-functional aspects in service oriented architecture. In *Proceedings of the 2006 IEEE International Conference on Service Computing*, pages 222–229, Chicago, IL.

Wada, H., Suzuki, J., and Oba, K. (2011). Leveraging early aspects in end-to-end model driven development for non-functional properties in service oriented architecture. *J. Database Manag.*, 22(2):93–123.

Yulin, N., Huayou, S., Weiping, L., and Zhong, C. (2010). Pdus: P2p-based distributed uddi service discovery approach. *Service Sciences, International Conference on*, 0:3–8.

Zhao, Y., Zhao, B., Liu, M., Hu, C., and Ma, D. (2010). Towards a graph grammar based verification approach for runtime constrained evolution of service-oriented architectures. In *SOSE*, pages 159–164. IEEE.

Zisman, A., Dooley, J., and Spanoudakis, G. (2008). Proactive runtime service discovery. *IEEE International Conference on Services Computing (July 2008)*, pages 237–245.