Towards an Approach for Modeling and Formalizing SOA Design Patterns with Event-B

Imen Tounsi, Mohamed Hadj Kacem, Ahmed Hadj Kacem, Khalil Drira

Towards an Approach for Modeling and Formalizing SOA Design Patterns with Event-B

Imen Tounsi[1], Mohamed Hadj Kacem[1], Ahmed Hadj Kacem[1], and Khalil Drira[2,3]

[1] ReDCAD-Research unit, University of Sfax, Sfax, Tunisia,
[2] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[3] Univ de Toulouse, LAAS, F-31400 Toulouse, France,
{imen.tounsi,mohamed.hadjkacem}@redcad.org, ahmed.hadjkacem@fsegs.rnu.tn, khalil@lass.fr

Abstract. This paper introduces a formal architecture-centric approach, which allows first to model message-oriented SOA design patterns with the SoaML standard language, and second to formally specify these patterns at a high level of abstraction using the Event-B method. These two steps are performed before undertaking the effective coding of a design pattern providing correct by construction pattern-based software architectures. We implement our approach under the Rodin platform which we use to prove model consistency.

Keywords: Design patterns: SoaML modeling: Event-B method

1 Introduction

Service-oriented architectures (SOA) is a technology that offers a model and an opportunity to solve problems related to the communication and the integration between heterogeneous applications [Erl, 2009]. Nevertheless these architectures are subject to some quality attribute failures (e.g., reliability, availability, and performance problems). *Design patterns*, as tested solutions to common design problems within a context, have been widely used to solve these weaknesses.

Most design patterns are presented in an informal way that can raise ambiguity and may lead to their incorrect usage. Patterns, proposed by the SOA design pattern community, are described with informal visual notations [Erl, 2009]. The intent of our approach is to model and formalize message-oriented SOA design patterns. These two steps are performed before undertaking the effective coding of a design pattern, so that the pattern in question will be correct by construction. Our approach allows to reuse correct SOA design patterns, hence we can save effort on proving pattern correctness.

In this paper, we introduce a formal architecture-centric approach. The key idea is to model SOA design patterns with the semi-formal Service oriented architecture Modeling Language (SoaML) in order to attribute a standard notation to SOA design patterns, then to formally specify them with the Event-B method. We have tested our approach with pattern examples. From these patterns we quote the *Asynchronous Queuing* pattern proposed by the SOA design pattern community. We implement the specifications under the Rodin platform which we use to prove model consistency. We provide both structural and behavioral features of SOA design patterns in modeling and formalizing steps. Structural features of a design pattern are generally specified by assertions on the existence of entity types in the pattern. The configuration of the entities is also described, in terms of the static relationships between them. Behavioral features are defined by assertions on the temporal orders of the messages exchanged between the entities [Zhu and Bayley, 2010].

The rest of this paper is organized as follows. Section 2 gives an overview of our proposed approach. Section 3 discusses related work. Section 4 concludes and gives future work directions.

2 Approach overview

The proposed approach mainly consists of two steps. In the first step, SOA design patterns are modeled graphically with the SoaML language [OMG, 2012]. In the second step, the obtained graphical models are formalized with the Event-B method [Abrial, 2010] (Fig. 1).

2.1 Pattern Modeling

We provide a modeling solution for describing SOA design patterns using a visual notation based on the graphical SoaML standard language. To specify structural features, we use Participant diagram that allows modeling entities that make up the pattern's architecture, their types and their dependencies (connections). We also use ServiceInterface and MessageType diagrams to respectively model interfaces of entities and exchanged messages. To specify behavioral features, we use UML2.0 sequence diagram that provides a graphical notation to describe dynamic aspects of design patterns.

Entities, that make up the architecture of an SOA design pattern, can be either *Participants* or *Agents*. It is a *Participant* when it provides and/or consumes services and it is an *Agent* when it can adapt to and interact with its environment. *Agents* are also participants, providing and using services. Entities can have *ports* that constitute interaction points with their environment. The *port* type can be either service or request. The communication path between Services and Requests within an architecture is called *ServiceChannel*.

2.2 Pattern Formalization

For the formalization of SOA design patterns, we use the *Event-B* method.We use the Rodin Platform [Abrial et al., 2010] in order to prove the correctness of the pattern specification.

A design pattern is described with structural features and behavioral features. Structural features are specified with one or several contexts PCi and behavioral features are specified with one or several machines PMi. To reduce the complexity of pattern formalizations, we define specification levels. In the first level, we create a very abstract model (a context $PC0$ and a machine $PM0$). In the next levels, we use the refinement techniques to gradually introduce detail and complexity into our model until obtaining the final pattern specification. When we move from Level(i) to Level(i+1), we add a new entity and its connections to the model. In Level(i+1), the context PCi is extended with the context $PC(i+1)$ and the machine PMi is refined with the machine $PM(i+1)$. The refined machine sees the extended context.

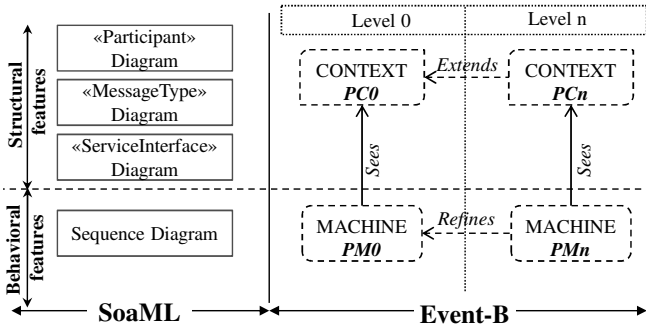

Fig. 1. Approach overview

3 Related work

Research connected to design patterns in the field of software architecture are mainly classified into three branches of work according to their architectural style. The first is about design patterns for Object-Oriented Architectures [Gamma et al., 1995], the second is about design patterns for Enterprise Application Integration (EAI) [Gregor and Bobby, 2003], and the third is about design patterns for Service Oriented Architectures (SOA) [Erl, 2009]. Most proposed patterns are presented in an informal way, using a combination of textual description and a graphical notations in order to make them easy to read and understand [Gamma et al., 1995,Gregor and Bobby, 2003,Erl, 2009]. However, using these descriptions makes patterns ambiguous and may lack details. Therefore, several researches have proposed the formalization of these patterns. Since the most famous ones are those proposed by Gamma [Gamma et al., 1995], most researches refer to these patterns [Zhu and Bayley, 2010]

[Taibi and Ngo, 2003,Dong et al., 2007,Kim and Carrington, 2009]. These researches use several formal techniques to define pattern specifications like the Balanced Pattern Specification Language (BPSL) [Taibi and Ngo, 2003], TLA [Dong et al., 2007] and Object-Z [Kim and Carrington, 2009] [Dong et al., 2007]. All these research address object-oriented design patterns, however in our research work we are interested in *SOA design patterns* and we are based on patterns defined by Erl [Erl, 2009]. Erl presents his patterns with an informal proprietary notation. So, in our work, we propose to model SOA design patterns with the SoaML standard language and we focus on their structural and behavioral features.

4 Conclusions

In this paper, we introduce a formal architecture-centric design approach supporting modeling and formalizing message-oriented SOA design patterns. The modeling phase allows to represent SOA design patterns with a graphical standard notation using the SoaML language. The formalization phase allows to formally specify both structural and behavioral features of these patterns at a high level of abstraction using the Event-B method. So far, we have implemented the elaborated specifications under the Rodin platform. We also illustrated our approach through a pattern example within the "Service messaging patterns" category. Currently, the transition from the SoaML modeling to the formal specification is achieved manually, we are working on automating this phase by implementing transformation rules.

5 Acknowledgments

This paper is done with the support of the Ministry of Higher Education and Scientific Research of Tunisia within the Tunisian-French scientific cooperation (DGRS/CNRS).

References

[Abrial, 2010] Abrial, J.-R. (2010). *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, New York, NY, USA, 1st edition.

[Abrial et al., 2010] Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T. S., Mehta, F., and Voisin, L. (2010). Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *Int. J. Softw. Tools Technol. Transf.*, 12(6):447–466.

[Dong et al., 2007] Dong, J., Alencar, P. S. C., Cowan, D. D., and Yang, S. (2007). Composing pattern-based components and verifying correctness. *J. Syst. Softw.*, 80:1755–1769.

[Erl, 2009] Erl, T. w. a. c. (2009). *SOA Design Patterns (The Prentice Hall Service-Oriented Computing Series from Thomas Erl).* Prentice Hall PTR, 1 edition.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R. E., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA.

[Gregor and Bobby, 2003] Gregor, H. and Bobby, W. (2003). *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions.* Addison Wesley.

[Kim and Carrington, 2009] Kim, S.-K. and Carrington, D. A. (2009). A formalism to describe design patterns based on role concepts. *Formal Asp. Comput.*, 21(5):397–420.

[OMG, 2012] OMG (2012). Service oriented architecture Modeling Language (SoaML) Specification. Technical report.

[Taibi and Ngo, 2003] Taibi, T. and Ngo, D. C. L. (2003). Formal specification of design pattern combination using BPSL. *Information and Software Technology*, 45(3):157 – 170.

[Zhu and Bayley, 2010] Zhu, H. and Bayley, I. (2010). Laws of pattern composition. In *Proceedings of the 12th international conference on Formal engineering methods and software engineering*, ICFEM'10, pages 630–645, Berlin, Heidelberg. Springer-Verlag.