Expert Systems with Applications 40 (2013) 508–522

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Symbolic model checking composite Web services using operational and control behaviors

Jamal Bentahar [a,b,*], Hamdi Yahyaoui [c], Melissa Kova [d], Zakaria Maamar [e]

[a] Concordia University, Concordia Institute for Information Systems Engineering, Montreal, Canada
[b] Khalifa University of Science, Technology and Research, College of Engineering, Abu Dhabi, United Arab Emirates
[c] Kuwait University, Computer Science Department, Kuwait
[d] Ubisoft Divertissement, Montreal, Canada
[e] Zayed University, College of Information Technology, Dubai, United Arab Emirates

ARTICLE INFO

Keywords:
Composite Web service
Behavior
Soundness
Completeness
Symbolic model checking
NuSMV

ABSTRACT

This paper addresses the issue of verifying if composite Web services design meets some desirable properties in terms of deadlock freedom, safety (something bad never happens), and reachability (something good will eventually happen). Composite Web services are modeled based on a separation of concerns between business and control aspects of Web services. This separation is achieved through the design of an operational behavior, which defines the composition functioning according to the Web services' business logic, and a control behavior, which identifies the valid sequences of actions that the operational behavior should follow. These two behaviors are formally defined using automata-based techniques. The proposed approach is model checking-based where the operational behavior is the model to be checked against properties defined in the control behavior. The paper proves that the proposed technique allows checking the soundness and completeness of the design model with respect to the operational and control behaviors. Moreover, automatic translation procedures from the design models to the NuSMV model checker's code and a verification tool are reported in the paper.

1. Introduction

Web services are widely used for developing Business-to-Business applications whose performance spreads by default over organizations' boundaries. Indeed, Web services rely on a set of platform-independent and vendor-neutral specifications that offer the necessary means for their description, discovery, invocation, and mainly composition (Yang, Zhang, & Lan, 2007). Despite the great interest of the research and industry communities in Web services, some challenging issues remain pending and hence, hinder the adoption of Web services to develop robust, dynamic, and safe business applications. Examples of issues include verification and model checking (Lomuscio, Qu, & Solanki, 2008, 2011; Rouached, Fdhila, & Godart, 2010; Yeung, 2011), reliability (Bhiri, Perrin, & Godart, 2005; Zhang & Zhang, 2005), security (Yahyaoui, 2012), transaction handling (Zhao, Kart, Moser, & Melliar-Smith, 2008), service discovery (Fardin, Naser, & Ali, 2012; Tian & Huang, 2012), and last but not least context awareness of interaction management (Handorean, Sen, Hackmann, & Roman, 2006). The severity of these issues intensifies when several component Web services are put together to form composite Web services (Benslimane, Maamar, & Ghedira, 2006; Maamar, Benslimane, Cherida, & Mrissa, 2005).

In this paper, we address the "thorny" issue of verifying the design of composite Web services. Developing business applications that end-users trust requires a deep investigation of the different and independent operations and behaviors that the component Web services in a composition execute and exhibit, respectively. For instance, having a deadlock in a Web service-based business transaction is a simply disaster for all stakeholders. Although software vendors can guarantee the safety of their Web services, the development, testing, and verification of these Web services are done independently from other vendors' peers, which means a serious lack of how these Web services behave when put together. To tackle this problem, we use *model checking*, a powerful formal and fully automatic technique for the verification of system models against specified properties in an early stage in the system lifecycle. Compared to other verification techniques such as code reviewing and testing, model checking is the only technique that provides a formal proof and thus a guarantee that the system is sound with respect to the checked properties. Using this technique, the Web services composition model, which captures the behavior

* Corresponding author. Address: Concordia University, Concordia Institute for Information Systems Engineering, 1515 Ste-Catherine Street West, EV7.640, Montreal, Quebec, Canada H3G 2W1.

E-mail addresses: bentahar@ciise.concordia.ca (J. Bentahar), hamdi@sci.kuniv.edu.kw (H. Yahyaoui), melissa.kova@ubisoft.com (M. Kova), zakaria.maamar@zu.ac.ae (Z. Maamar).

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2012.07.069

of this composition, can be investigated to check if it satisfies desirable properties such as *reacheability* and *safety*, and does not satisfy undesirable properties such as *deadlock*. The problem of model checking is formally denoted by $M \models \varphi$, where M is the system model, φ a property, and $\models$ the satisfaction symbol, meaning that the model M satisfies the property φ. In model checking approaches (Cimatti et al., 2002; Clarke, Grumberg, & Peled, 1999), M is represented in a formal language capturing the system's dynamics and φ, against which the model is checked, is expressed in a given logic such as *Linear Temporal Logic* (LTL) and *Computation Tree Logic* (CTL) (Emerson, 1990). The verification technique examines all possible executions of the model. In this paper, we focus on a particular technique of model checking referred to as *symbolic*. In symbolic model checking, the model and properties are not explicitly represented as automata, but implicitly and symbolically encoded using Boolean functions and binary decision diagrams, which alleviates the state explosion problem, the main drawback of automata-based techniques (Clarke et al., 1999).

As pointed out in many proposals, for instance (Meng & Arbab, 2007; Peltz, 2003; Zhou, Huang, & Wang, 2007), two composition approaches dominate service-oriented computing: choreography and orchestration. On the one hand, choreography identifies the set of acceptable conversations for a composite Web service without allocating a specific Web service to monitor the composition progress. It captures the global view of the composition by focusing on the coordination and multi-participant perspectives. On the other hand, orchestration is an executable specification that identifies the steps of execution for the peers where one particular party monitors the execution of such a specification. It captures the local view of the composition from one participant perspective. Although the approach we propose in this paper is choreography-oriented because it stresses the coordination and conversation among Web services in a global perspective, it is still general enough and can be applied to orchestration as well, as far as the design model, which is needed for model checking, is fully available.

To better model compositions, we build upon the idea of separating concerns that model, at different levels of abstractions, two different behaviors of Web services: *operational* and *control* (Yahyaoui, Maamar, & Boukadi, 2010; Sheng, Maamar, Yahyaoui, Bentahar, & Boukadi, 2010). From the perspective of single Web services, the operational behavior is application-dependent; it defines the business logic underpinning a Web service operation at a low level of abstraction by specifying the functions the Web service should perform, and the control behavior is application-independent; it defines at a high level of abstraction a general design pattern of the Web service and guides and monitors the execution progress of the operational behavior by identifying the actions to take and constraints, if any, to satisfy. Compared to design patterns, the control behavior is not implementable, so not directly transformable to executable code, but helps design the operational behavior, which is implementable. These two behaviors implement the separation of concerns between the business logic details and the abstract control logic of a Web service. The motivation behind such a separation is to enhance the current design practices of Web services that are prone to errors since the architectural and operational perspectives are treated at the same level, which increases the design and management complexity. From the design pattern perspective, the control behavior, when synchronized with the operational behavior, guarantees that design requirements and constraints are considered and good practices are being used. Dissociating the operational behavior from the control behavior permits to assess at design time the impact of any change in the business logic on the control behavior as shown in Sheng et al. (2010). Thus, any change in the operational behavior that does not impact the synchronization with the control behavior is allowable as it reflects the continuous fulfillment of the design pattern. The control behavior controls then the flexibility of the business logic and serves as a tool for analyzing the sensibility of the Web service to design changes. For example, if a new business rule has to be incorporated in the Web service's functionality, this will not affect the control behavior. The main advantage of such a practice is to provide service engineers with a means to monitor the execution of the Web service and identify and address design problems at an early stage.

In a previous work (Yahyaoui et al., 2010), the control and operational behaviors have been thoroughly investigated in the context of individual Web services, i.e., isolated from other peers in the same composition. The authors generalized this separation to the composite scenarios extending the separation framework proposed in Kova, Bentahar, Maamar, and Yahyaoui (2009). Unlike the control behavior of isolated Web services that focuses on the internal behavior of an individual Web service, the control behavior of a composition focuses on the interactions among different Web services. The control behavior of a composition shapes the design pattern of this composition; it is application independent and hence, applicable to a wide range of compositions of Web services. Its objective is to frame, control, and monitor the business logic execution as it provides the guidelines for an appropriate composition behavior. Based on the principle of separation of concerns (Kambayashi & Ledgard, 2004), this behavior is designed without updating the operational behavior that captures the business logic of the composition. This behavior facilitates the reusability of composition scenarios as it is independent from any specific business case. The operational behavior describes then the business logic of a specific composite scenario by identifying the functions to perform and messages to exchange among the components, while the control behavior identifies, in an application-independent perspective, the valid actions and sequences that the operational behavior should follow. Similar to the case of isolated Web services, if a new composition constraint has to be added to the composition business logic, the control behavior will not be affected, but used to enable or disable such a constraint. In this paper, the formal machinery of this composition-oriented separation of behaviors is defined in order to be automatically verified. This automatic verification raises fundamental challenges. For instance, the control behavior of a Web service can be in interaction with the operational behavior of another one, which is involved in the same composition. This leads to the need of specifying valid interactions. We map the control behavior onto the operational behavior in order to verify the soundness and completeness of the composition process if the two behaviors are synchronized. We use symbolic model checking technique to implement this verification by checking if the model of the operational behavior satisfies the properties generated from the control behavior.

The contributions of this work are twofold:

1. Proposal of an automata-theoretic approach for modeling composite Web services based on control and operational behaviors. Both are linked together to check that the conversation sequences of the operational behavior, which implement the business logic, are synchronized with the valid sequences, called executions, specified by the control behavior (soundness checking) and *vise-versa* (completeness checking). We use statecharts enhanced with additional syntax to facilitate the mapping process between both behaviors.
2. Formal and automatic verification of the mapping procedure using symbolic model checking techniques. The implementation is done using a Java-based translation procedure, which is proven to be sound, and NuSMV model checker (Cimatti et al., 2002).