Architectural Characterization of Web Service Interaction Verification

Gopal N. Rai and G.R. Gangadharan

Abstract Web service interaction utilizes disparate models as it still does not have its own model for verification process. Adaptation of a different model is not always beneficial as it may prune several significant characteristics that worth considering in verification. The primary reason behind this adaptation is that the primitive characteristics are not well identified, standardized, and established for Web service interaction model. In this article, we therefore investigate the primitive characteristics of Web service interaction model that need to be well considered in verification. Further, we study the appropriateness and effectiveness of two modeling and verification phenomena namely *model checking* and *module checking* with respect to investigated primitive characteristics.

Keywords Web services composition · Formal methods · Modeling · Model checking · Module checking

1 Introduction

Web services are self contained, self-describing modular applications that can be published, located, and invoked across the web [1]. Since Web services interact with each other through messages (synchronous and asynchronous), concurrency related bugs and/or inconsistency problems are possible in the interaction patterns (communication patterns). In order to overcome the problem, Web service interaction verification is required.

G.N. Rai (✉) · G.R. Gangadharan
IDRBT, Castle Hills, Masab Tank, Hyderabad 500 057, India
e-mail: gopalnrai@gmail.com

G.R. Gangadharan
e-mail: geeyaar@gmail.com

G.N. Rai
SCIS, University of Hyderabad, Gachibowli, Hyderabad 500 046, India

A. Nagar et al. (eds.), *Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics*, Smart Innovation, Systems and Technologies 44, DOI 10.1007/978-81-322-2529-4_47

Classical testing techniques are inadequate to verify the interaction among services [2, 3]. A Web service interaction scenario resembles with reactive and concurrent systems, with component-based systems [4, 5], and with multi-agent systems [6] up to a certain extent. Therefore, various Web service interaction modeling and verification techniques are proposed on the basis of these approaches [6, 7]. However, composition among services tangles the task of verification as it poses unique primitive verification requirements that are not covered by either classical tests or other verification approaches.

Many efficient and industry wide accepted formal verification tools are available that could verify Web service interaction. However, transformation of a service interaction scenario into an input model for any existing verification tool may loose the architectural originality and significant native primitive notions, thus, restricting the verifiable properties (requirements). In this article, we characterize the model of Web service interaction for verification, a novel effort towards establishing *Web service interaction verification model*.

The rest of the article is structured as follows. Section 2 investigates fundamental characteristics of a Web service interaction model. We study the appropriateness and effectiveness of model checking and module checking in Sect. 3. Section 4 provides conclusion and future directions.

2 Primitive Characteristics of Web Service Interaction Model

In this section, primitive characteristics of the Web service interaction model are discussed as follows.

2.1 Modular and Hierarchical Architecture

All Web services basically fall into two categories: either basic or composite. A basic Web service does not take help of other Web services to accomplish the job whereas a composite Web service does.

Modular architecture of Web services interaction model refers to the design of a system composed of independent Web services that could interact with each other. The advantage of the modular architecture is that a module could be added or substituted with another suitable module without affecting the rest of the system.

Hierarchical architecture of Web services interaction model comes in the existence only when a composite service comes in the existence. A composite service depends on the other basic or composite services. A composite service is a higher level abstraction and basic Web services are lowest level abstraction. Web service interaction possess hierarchical architecture but nesting of Web services are not

allowed. Indeed, nesting of Web services does not make any sense as basic Web services work as independent modules.

Therefore, Web service interaction model has both shades of hierarchical and modular architectural archetype. It requires modeling a Web service in such a way that it preserves its identity and supports composite services at a time. Consequently, a Web service can serve as an independent module and can serve as an building block of a composite service.

2.2 Open System

Each basic Web service works as an independent module that fundamentally yields an open system from a verification perspective. An open system is one which is designed to interact with an environment [8]. A web service cannot anticipate all behaviors of its interacting partner in advance, thus enabling the open system phenomenon. In order to be able to model the open system, *environment modeling* is inevitable. Classical modeling approaches and many other new verification techniques does not support environment modeling. They typically apply to closed systems whose behavior is fully specified.

2.3 Trace Modeling

Informally, a trace (in the context of Web services) is a linear, unidirectional Web service composition workflow path in which each node represents a Web service and the directed edges indicate the flow from one service to another. Trace modeling and computation are among the most primitive modeling requirements for Web service interaction as the computation of trace related phenomenons (such as trace inclusion, trace crossing, and trace merging) reduce the time and space complexity for verification.

Formal definition of a Web service trace is given as follows.

Definition 1 (*Trace*) A trace is a tuple $\mathcal{T} = (\mathcal{W}, I, w_i, w_n)$, where $\mathcal{W} = \{w_1, \cdots, w_m\}$ is a finite set of the Web services, $I : \mathcal{W} \rightarrow \mathcal{W}$ is an invocation function such that $w_i I w_j$ if and only if w_i invokes w_j, $w_i \in \mathcal{W}$ is a service from which trace generation begin and $\nexists w_j \in \mathcal{W} : w_j I w_i$, and $w_n \in \mathcal{W}$ is a service on which trace ends up and $\nexists w_j \in \mathcal{W} : w_n I w_j$.

Let w_i be a Web service then $\mathcal{T}_{w_i}$ represents a set which contains all the traces generated by the service w_i. The concept of trace is utilized mainly while studying behavioral equivalence of services.

2.4 Asynchronous Messaging

A Web service consists ports and a port consists sets of input and output messages. Messages consisting activities are unit of interaction. There are two principal messaging models used in Web services namely synchronous and asynchronous model. The two Web service messaging models are distinguished by their way of request-response operation handling mechanism.

Synchronous messaging and asynchronous messaging among less number of services can be handled in a fair manner without much complexity. If involved number of services are high, asynchronous messaging complicates the verification process.

2.5 Recursive Composition

Composition and recursive composition are fundamental characteristics of Web service interaction model. Composition of services is aggregation of facilities provided by services. Recursive composition refers recursive aggregation of services. The difference between recursive composition and non-recursive composition is explained as follows.

Let $\langle A, B\rangle$ represents the knowledge that a Web service A is composed of a Web service B. Let $\{\langle A, B\rangle, \langle B, C\rangle, \langle C, D\rangle\}$ represents a composition scenario. If a composition dependency holds among tuples such as $\langle A, B\rangle$ depends upon $\langle B, C\rangle$ and $\langle B, C\rangle$ depends upon $\langle C, D\rangle$ then it forms a recursive composition scenario otherwise non-recursive composition.

2.6 Dynamic Reconfiguration

There are two crucial factors that make consideration of dynamic reconfiguration in the case of Web service inevitable. First, a Web service resembles a module (a basic Web service resembles an independent module whereas a composite Web service resembles a dependent module). Second, dynamic availability of services. Web services are accessible through the Web and a Web service could become unavailable/removed at any time or a new Web service could be introduced at any time. Ethically, a composition designer or verifier must be ready for substitution, replacement, and introduction of services. Introduction or unavailability of a service could make complete chaos if not handled properly. Automatic dynamic service composition is a rapidly emerging paradigm and research topic that is based on dynamic reconfiguration phenomenon.

2.7 Hierarchical Concurrency

Classical model-based verification approaches do not consider hierarchy among Web services while verifying the interaction. They keep transmitting all variables that are being considered for verification through every state in their state transition diagram. All variables need not be considered at a time. A hierarchy must be found among services and must be considered in verification process. We explain a hierarchical concurrency scenario among Web services as follows.

Let $\langle A, B\rangle$ represents the knowledge that A is composed of B and $\langle A, B\rangle \rightarrow \langle B, C\rangle$ represents the knowledge that $\langle A, B\rangle$ depends upon $\langle B, C\rangle$. Let us consider the following scenario

$$\begin{array}{ccccc} \langle A,B\rangle & & & & \langle I,J\rangle \\ \downarrow & & & & \downarrow \\ \langle B,C\rangle & \leftarrow & (B,J) & \rightarrow & \langle J,K\rangle \\ \downarrow & & & & \downarrow \\ \langle C,D\rangle & & & & \langle K,L\rangle \end{array}$$

Consider that concurrency has to be resolved between services B and J for a given specification. If $\langle B, C\rangle$ and $\langle J, K\rangle$ do not affect $\langle B, J\rangle$ regarding concurrency then concurrency will be resolved between B and J only (B and J are first level services). No need to involve the services C and K (C and K are second level services). If the concurrency is not resolved at first level then only second level services will be considered. Again, if second level services are also not sufficient to resolve the concurrency, third level services (D and L are third level services) will be considered.

We introduce a conceptual term *sphere of influence* to ease the process of hierarchical concurrency verification in the context of Web service interaction. Sphere of influence is a set of Web services computed for an Web service such that either each member of the set is directly invokable by the center service or invokes the center service. Figure 1 is a pictorial representation of the sphere of influence for a Web service namely WS1. This diagram infers that services WS2, WS3, WS4, and WS5 constitutes the sphere of influence set for the service WS1. Figure 2 is an

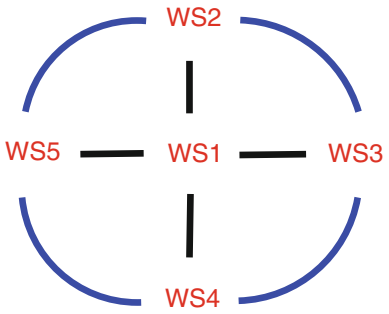

Fig. 1 Sphere of influence for a Web service WS1

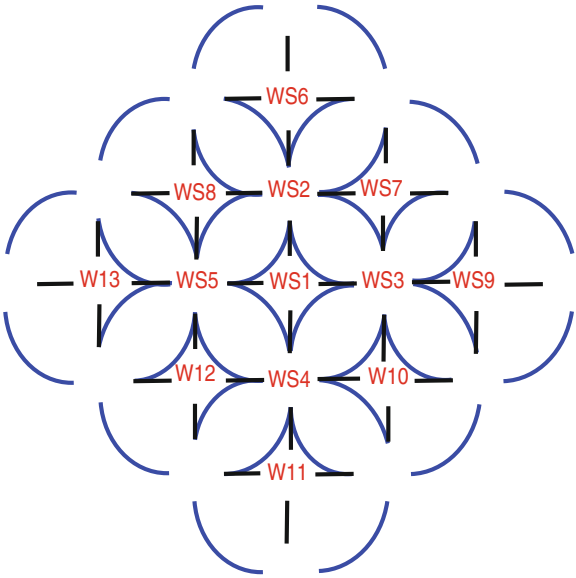

Fig. 2 Evolved version of sphere of influence depicted in Fig. 1

evolved version of sphere of influence depicted in Fig. 1. Sphere of influence serves as a basic model to verify hierarchical concurrency.

2.8 Verification of Adversarial Specification

Classical temporal logics such as linear temporal logic (LTL) and computation tree logic (CTL) are unable to specify collaborative as well as adversarial interactions among different Web services. Alternating Temporal Logic (ATL) [8] is designed to write requirements of open system [8] and is able to express collaborative as well as adversarial interaction specifications. ATL models each Web service as an agent. Let Σ be a set of agents corresponding to different Web services, one of which may correspond to the external environment. Then, the logic ATL admits formulas of the form $\langle\langle A\rangle\rangle\Diamond p$, where p is a state predicate and A is a subset of agents. The formula $\langle\langle A\rangle\rangle\Diamond p$ means that the agents in the set A can cooperate to reach a p-state no matter how the remaining agents resolve their choices.

3 Web Service Interaction Verification

In this section, we study and compare two different model-based formal verification approaches that are suitable for service interaction verification: model checking and module checking. Further, we study the working mechanisms of their representative

tools and its input languages whether there are native language constructs or not to support the desired characteristics discussed earlier in this paper.

Model Checking Model checking has been widely used as one of the formal techniques for Web service composition and interaction verification [7, 9–12]. In order to be able to model check a Web service interaction scenario, the scenario has to be modeled as an input model of model checking. The transformation of service interaction into an input model of model checking yields a monolithic structure. This transformed structure collapses several significant characteristics such as modular cum hierarchical architecture. Further, monolithic structure increases computation overhead and is not adequate for reasoning. The biggest drawback one faces with model checking is that it does not support open system modeling that is primary requirement for modeling of Web service interaction. Model checking does not have any provision for *interface* and *private* variable types. Consequently, a chaos develops among variables in verification.

SPIN [13] and NuSMV [14] are representative tools of model checking. SPIN supports only linear-time temporal logic whereas NuSMV supports both linear-time temporal logic as well as branching-time temporal logic. We study NuSMV (NuSMV 5.0) as a representative tool for model checking.

Module Checking Module checking is also an model-based formal verification technique. It could be considered as an alternative of model checking. Module checking differs from model checking in various means as follows. From modeling point of view, module checking supports the heterogeneous modeling framework of reactive modules whereas model checking supports unstructured state transition graphs. From specification writing point of view, module checking employs ATL to write specification about the system whereas model checking verifies the specifications written in LTL/CTL. From architecture point of view, module checking facilitates with hierarchical design and verification along with modular design whereas model checking provides only modular designing and verification.

We study MOCHA [15] (jMOCHA 2.0) as a representative tool for module checking. MOCHA is an interactive verification environment that supports a range of compositional and hierarchical verification methodologies.

A comparative analysis between NuSMV 5.0 and jMOCHA 2.0 with respect to investigated characteristics is shown in Table 1.

4 Related Work

To the best of our knowledge, a paper merely focusing on architectural characterization of Web service interaction is not available in literature. However, some of the studied characteristics are discussed individually in articles. Hierarchical

Table 1 Verification requirements versus verification approaches

Verification requirements	Model checking (NuSMV)	Module checking (jMOCHA)	Property type
Modular architecture	Yes	Yes	Modeling
Hierarchical architecture	No	Yes	Modeling
Variables support	Global	External, interface, private	Modeling
Lazy evaluation	No	Yes	Modeling
Dynamic reconfiguration	No	No	Modeling
Composition	No	Yes	Modeling
Recursive composition	No	No	Modeling
Open system modeling	No	Yes	Modeling
Trace containment verification	No	Yes	Modeling
Adversarial specification	No	Yes	Specification

architecture of Web services is discussed in [16]. Hnetynka et al. [4] discusses dynamic reconfiguration of Web services and collaboration among them. Roglinger [17] presents a requirements framework for verification. However, the proposed requirements in [17] corresponds to model checking and not characterizing the Web service interaction. Pistore et al. [18] present a requirement model for verification of Web services. However, the authors focus on business requirements not architectural characterization in the paper [18].

We do not find any evidence of Web service interaction verification using module checking tool (jMOCHA) in literature. However, ATL is used for Web service verification along with Petri net in [19].

5 Conclusion and Future Work

On the basis of our study, we conclude that Web service interaction is a modular cum hierarchical architecture. Each Web service works as an open system. Recursive composition is a fundamental characteristic of Web service interaction. And a verification approach for Web services must support hierarchical concurrency verification with adversarial specification facility.

Though module checking is an efficient verification technique for reactive, concurrent, and open systems, it is not widely accepted among industry and academia. Reasons might be lack of popularity and complex specification writing scheme. However, on the basis of our study, we advocate use of module checking rather than model checking for verification of Web services interaction.

There are many improvement suggestions that could be considered as parts of future work. First, we want to investigate hierarchical structure of Web service interaction in more depth. Both verification approaches model checking and module checking do not support recursive composition and dynamic introduction or removal of a service. Therefore, realization of a verification approach based on investigated characteristics and sphere of influence will be our second future work. Discovering all fundamental and advanced verification requirements for Web service interaction is also a part of our future work.

References

1. Wang, H., Huang, J.Z., Qu, Y., Xie, J.: Web services: problems and future directions. Web Semantics: Sci. Serv. Agents World Wide Web **1**(3), 309–320 (2004)
2. Bozkurt, M., Harman, M., Hassoun, Y.: Testing Web Services: A survey. Department of Computer Science, King's College London, Tech. Rep. TR-10–01 (2010)
3. Mei, L., Chan, W., Tse, T., Jiang, B., Zhai, K.: Preemptive regression testing of workflow-based web services. IEEE Trans. Services Comput. (2014)
4. Hnetynka, P., Plášil, F.: Dynamic reconfiguration and access to services in hierarchical component models. In: Component-Based Software Engineering, pp. 352–359. Springer (2006)
5. Lau, K.K., Wang, Z.: Software component models. IEEE Trans. Softw. Eng. **33**(10), 709–724 (2007)
6. Walton, C.: Model checking multi-agent web services. In: AAAI Symposium of Semantic Web Services (2004)
7. El Kholy, W., Bentahar, J., El Menshawy, M., Qu, H., Dssouli, R.: Modeling and verifying choreographed multi-agent-based web service compositions regulated by commitment protocols. Expert Syst. Appl. **41**(16), 7478–7494 (2014)
8. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM (JACM) **49**(5), 672–713 (2002)
9. Bentahar, J., Yahyaoui, H., Kova, M., Maamar, Z.: Symbolic model checking composite web services using operational and control behaviors. Expert Syst. Appl. **40**(2), 508–522 (2013)
10. Boaro, L., Glorio, E., Pagliarecci, F., Spalazzi, L.: Semantic model checking security requirements for web services. In: 2010 International Conference on High Performance Computing & Simulation, pp. 283–290 June 2010
11. Marques, A.P., Ravn, A.P., Srba, J., Vighio, S.: Model-checking web services business activity protocols. Int. J. Softw. Tools Technol. Transf. **15**(2), 125–147 (2013)
12. Sheng, Q.Z., Maamar, Z., Yao, L., Szabo, C., Bourne, S.: Behavior modeling and automated verification of Web services. Inf. Sci. **258**, 416–433 (2014)
13. Holzmann, G.J.: The SPIN model checker: Primer and reference manual, vol. 1003. Addison-Wesley Reading (2004)
14. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: Computer Aided Verification, pp. 359–364. Springer (2002)
15. De Alfaro, L., Alur, R., Grosu, R., Henzinger, T., Kang, M., Majumdar, R., Mang, F., Meyer-Kirsch, C., Wang, B.: Mocha: Exploiting Modularity in Model Checking. Tech. Rep, DTIC Document (2000)
16. Kreger, H.: Web services conceptual architecture (wsca 1.0). IBM Software Group 5 (2001)

17. Rglinger, M.: Verification of web service compositions: an operationalization of correctness and a requirements framework for service-oriented modeling techniques. Business & Inform. Syst. Eng. **1**(6), 429–437 (2009)
18. Pistore, M., Roveri, M., Busetta, P.: Requirements-driven verification of web services. Electronic Notes Theor. Comput. Sci. **105**, 95–108 (2004)
19. Schlingloff, H., Martens, A., Schmidt, K.: Modeling and model checking web services. Electron. Notes Theoret. Comput. Sci **126**, 3–26 (2005)