A Simulation-driven Approach for Composite Web Services Validation

Vadym Shkarupylo
Computer Systems and Networks Dept.
Zaporizhzhya National Technical University
Zhukovsky 64, 69063 Zaporizhzhya, Ukraine
vadshkar@yandex.ua

Abstract. *The approach for Composite Web Services validation has been proposed. It's based on DEVS-formalism usage, which provides the ability to conduct validation by way of simulation. The WS-BPEL-description has been considered as an input data. The Temporal Logic of Actions (TLA) has been chosen to check the description.*
The proposed approach provides the ability to perform validation with no need to actually deploy the Composite Web Service.
To check the proposed approach the case study has been conducted. For this purpose the testing-driven and the simulation-driven validations have been conducted. The results obtained have shown the approach applicability.

Keywords. Composite Web Service, Simulation, Validation, WS-BPEL, TLA, DEVS

1 Introduction

Nowadays the principles of Service-oriented architecture (SOA) are ubiquitously considered as a key principles in different distributed software systems engineering processes. Such principles are reusability, composability, loose coupling, etc. (Papazoglou et al., 2007). Typical representatives of the appropriate systems are the Web Services, the Composite Web Services (built on reusability and composability principles) in particular.

Today the engineering process itself is tightly coupled with extensive formal methods usage. Good example of this are the Amazon Web Services (Newcombe et al., 2015). Here the Temporal Logic of Actions (TLA) and the appropriate Model Checker TLC (TLA Checker) are utilized to check the design of the system. The TLA is based on TLA+ formalism (Lamport, 2002). Another promising way of TLA usage is the C code checking – to prevent possible runtime errors (Methni et al., 2014).

In this paper the Composite Web Service (CWS) is considered as a distributed SOA-based software system with both – functional and non-functional – properties. It's supposed that system functioning is based on an orchestration model, where all the components of the system (the atomic web services) are coordinated (invoked) in a centralized manner ("Web Services Business Process Execution Language Version 2.0", 2007).

The proposed approach is aimed at the necessity to perform the model-based formal verification to previously check the consistency of WS-BPEL-description given. Only after that the simulation-driven validation has to be conducted.

To build the simulation model of CWS the DEVS formalism (Discrete Event System Specification, by Bernard P. Zeigler) has been chosen (Wainer & Mosterman, 2010). This formalism provides the convenient way to represent the system and its components – because of the existence of the concepts of atomic and coupled DEVS-models. In given paper the concept of atomic DEVS-model is used as a structure template for the atomic web services models in particular, the concept of coupled DEVS-model – as a structure template for CWS-model.

One of the key advantages of simulation over the testing in SOA-environment is the reduction of risks involved (Kaur & Ghumman, 2015). A considerable analysis of existing simulation tools with a particular accent on the advantages of Java-based ones has already been conducted (King & Harrison, 2010).

The distinctive feature of DEVS-formalism is that the appropriate DEVS Suite toolkit provides the ability to visualize simulation model architecture and the simulation process itself, which is not the case for SimJava for instance (Rahman et al., 2015).

Both aforementioned formalisms – the TLA+ and the DEVS-formalism – have been created for a completely different purposes. The TLA+ provides the ability to rigidly specify the required properties of system developed to find out by way of model checking, whether the properties meet the requirements on a chosen level of abstraction. Unlike the TLA+, The DEVS-formalism has been created with simulation in mind – to answer the question,

whether the system developed is applicable to a given usage scenario. If represent the engineering process as iterative one, where each iteration is a sequence of requirements analysis, design, implementation and testing phases (Larman, 2004), the TLA+ formalism has to be used during the design phase. If consider the testing as a way of validation implementation, the forth (testing) phase can be swapped with more abstract phase – validation. The validation itself can be implemented by way of testing or by way of simulation. That means that DEVS-formalism should be used during the final (validation) phase.

The main contribution of the proposed paper consists in an attempt to couple TLA+ and DEVS-formalisms to successfully conduct the validation of CWS by way of simulation – with no need to actually perform the time-consuming deployments and redeployments of CWS and its components. The TLA here plays the auxiliary role – as a remedy to prevent the unnecessary DEVS-driven simulations in case of inconsistencies in given WS-BPEL-description. The named description here is the representation of certain CWS functional property. The DEVS-validation should be conducted as a subsequent step, that has to be accomplished only in case of successful verification of synthesized TLA+ specification. During such validation both – functional and non-functional – properties of CWS have to be checked.

To check the practical applicability of the proposed approach the results of simulation-driven DEVS-based validation are compared with the results of testing-driven alternative. For this purpose the atomic web services are implemented and deployed with JAX-WS technology (Vohra, 2012). Java-based implementation makes it possible to represent the functional properties of services in atomic DEVS-model one-to-one. It is essential to notice that non-functional properties of atomic web services can be represented in atomic DEVS-models as it is (for instance, when talking about the cost of service usage, obtained from certain service provider, or about the time costs, connected with functional property implementation – the computational process, that can be measured directly) or as estimated values – when talking about the communicational time costs.

A completely different approach has been proposed earlier – the TLA has been used to check the DEVS-models (Cristia, 2007).

2 Approach Description

Given the WS-BPEL-description of CWS, which consists of two types of activities – the Basic Activities and the Structured Activities. There will be used only one representative from the Basic Activities group, represented with <invoke> tag. Such tags will be utilized to form a state variables set of formal TLA+ specification to be verified. Each <invoke> tag is a representation of the appropriate atomic web service to be invoked in accordance with the scenario given in WS-BPEL-description. The invocations scenarios determine the architecture of CWS. For this purpose the <sequence> and <flow> constructs from the Structured Activities group have been considered.

In given paper the WS-BPEL-description will be considered as a representation of some functional property of CWS. From this description the formal TLA+ specification has to be synthesized. In case of multiple WS-BPEL-descriptions of certain CWS the TLA+ to WS-BPEL artefacts relation has to be "one-to-many".

To analytically represent the formal TLA+ specification, synthesized from the given WS-BPEL-description, the Kripke structure on a set of atomic prepositions AP is used (Clarke, Grumberg & Peled, 2001):

$$\langle S, \{s_0\}, R, L \rangle, \quad (1)$$

where S – finite set of states, $s_0 \in S$ – initial state, $R \subseteq S^2$ – set of transitions, $L : S \to 2^{AP}$ – states labelling function.

The AP set is formed as follows:

$$AP = V \times D, \quad (2)$$

where $V = \{v_i | i = 1,2,\ldots,n \in N\}$ – set of state variables, $D = \{0,1,2\}$ – set of allowable state variables values. Each $v_i \in V$ is a representation of i-th CWS component – atomic web service, figuring in WS-BPEL-description as an <invoke> tag.

The elements of AP set should be interpreted as following (Shkarupylo, Tomičić & Kasian, 2016):

- $(v_i, 0) \in AP$ – the i-th component hasn't yet been invoked;
- $(v_i, 1) \in AP$ – the i-th component has already been invoked and is functioning;
- $(v_i, 2) \in AP$ – the i-th component has already been invoked and its functioning has been finished.

Generally, in accordance with reusability SOA principle, the i-th component can be invoked more than once (Fig. 1).

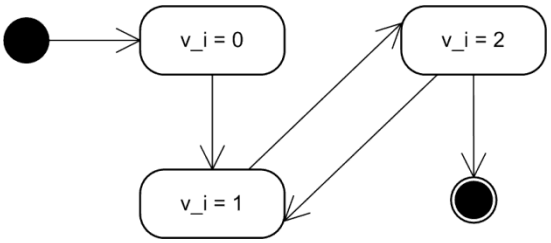

Figure 1. Atomic web service state chart diagram

Generally, some another CWSs can be considered as the components too, but for the purpose of simplicity only the atomic web services are represented in V set.

The proposed approach consists of the following steps:

- Obtain the WS-BPEL-description of CWS;
- Synthesize the formal TLA+ specification – in conformity with the aforementioned concepts;
- Verify the synthesized TLA+ specification in an automated manner with TLC model checker to check the functional property (properties) of CWS from the consistency viewpoint;
- If the verification has been successful, synthesize the atomic and coupled DEVS-models from TLA+ specification;
- Conduct the simulation-driven DEVS-validation, checking both – functional and non-functional – properties.

The WS-BPEL to TLA+ translation rules are the following:

- Form a set of state variables of formal specification from the <invoke> tags, given in WS-BPEL-description;
- Reproduce the structure of WS-BPEL-description in TLA+ specification.

There have been distinguished three types of atomic DEVS-models (Shkarupylo, Kudermetov & Polska, 2015). All the distinguished types of DEVS-models have typical structure, but are intended to be used for a completely different purposes:

- The Generator model – MG – to model (generate) client requests to the CWS, and to model the requests (jobs) distribution in particular;
- The Coordinator model – MC – to model the WS-BPEL-engine – an engine for centralized Atomic Web Services coordination – in conformity with WS-BPEL-description;
- The model of i-th component – MA_i – which directly performs the assigned part of calculations.

The aforementioned atomic DEVS-models of three types and the coupled one discussed further have to be synthesized directly from the successfully verified TLA+ specification. For instance, in case of TLA+ specification with m state variables, where $m \le n$ is a number components utilized for certain functional property implementation, there should be synthesized exactly $m+2$ atomic DEVS-models: m of those are the CWS components models; another couple of models are MG and MC. The total number of models to be synthesized is $(m+3)$ – plus the single resulting coupled DEVS-model.

The Generator atomic DEVS-model has the following structure:

$$MG = \left\langle \{act\}, \{job_j\}, ST, \delta_{ext}^{MG}, \delta_{int}^{MG}, \lambda^{MG}, ta \right\rangle, \quad (3)$$

where act – single input port for model activation; $\{job_j\}$ – set of output ports for client requests (represented as messages) generation; $ST = \{"busy", "passive"\}$ – set of states labels; $\delta_{ext}^{MG} : (act, st, e) \mapsto st'$ – external transition function, where $st = "passive" \in ST$, e – time, elapsed since last transition, $st' = "busy" \in ST$; $\delta_{int}^{MG} : st' \mapsto st$ – internal transition function; $\lambda^{MG} : st' \mapsto job_j$ – output function; $ta : ST \to R_{0,\infty}^{+}$ – time advance function.

The Coordinator atomic DEVS-model:

$$MC = \left\langle IP, \{req_i\}, ST, \delta_{ext}^{MC}, \delta_{int}^{MC}, \lambda^{MC}, ta \right\rangle, \quad (4)$$

where $IP = \{job_j\} \cup \{res_i\}$ – set of input ports, where $\{job_j\}$ – set of ports for taking client requests; $\{res_i\}$ – set of ports for taking MA_i models functioning results; $\{req_i\}$ – set of output ports for computational task parts transferring to satisfy j-th client request; $\delta_{ext}^{MC} : (job_j, st, e) \mapsto st'$ – external transition function; $\delta_{int}^{MC} : st' \mapsto st$ – internal transition function; $\lambda^{MC} : st' \mapsto req_i$ – output function.

Atomic DEVS-model of i-th atomic web service:

$$MA_i = \left\langle \{ev\}, \{rs\}, ST, \delta_{ext}^{MA_i}, \delta_{int}^{MA_i}, \lambda^{MA_i}, ta \right\rangle, \quad (5)$$

where ev – input activation port – to retrieve the appointed part of computational task; rs – output port – to send the result of computation; $ST = \{st_1, st_2, st_3\}$ – set of states labels – the DEVS-representation of $\{var\} \times D$ set – a subset of $V \times D$ (eq. 2): $var \in V$; $st_1 = "passive" \in ST$ – the representation of $(v_i, 0) \in AP$; $st_2 = "busy" \in ST$ – of $(v_i, 1) \in AP$; $st_3 = "finalized" \in ST$ – of $(v_i, 2) \in AP$; $\delta_{ext}^{MA_i} : (ev, st, e) \mapsto st'$ – external transition function; $\delta_{int}^{MA_i} : st' \mapsto st$ – internal transition function; $\lambda^{MA_i} : st' \mapsto rs$ – output function.

The atomic DEVS-models (eq. 3–5) are grouped within the coupled DEVS-model (Fig. 2).

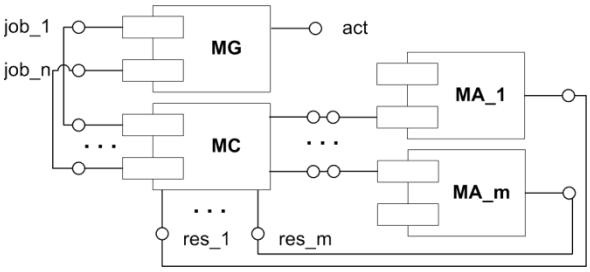

Figure 2. The architecture of coupled DEVS-model

3 Approach Checking

To check the proposed approach applicability the results of simulation-driven validation (based on the approach) have been compared to the results of testing-driven validation.

The atomic web services have been implemented with JAX-WS technology.

The testing has been conducted as following:

- The unit testing of atomic web services methods has been accomplished;
- atomic web services have been deployed on a GlassFish application server. The testing of deployed services has been conducted;
- The CWS has been deployed and tested.

As a case study the distributed π value calculation has already been considered (Shkarupylo, 2016).

In this paper the approximation problem solving has been contemplated. As an input data the time costs of BFS-driven TLC-verification of TLA+ specifications have been considered (Shkarupylo, Tomičić & Kasian, 2016). The dependence of time costs from state variables number has been represented with the following polynomial:

$$g(x) = a + b \cdot x + c \cdot x^2 + d \cdot x^3, \qquad (6)$$

where x – the amount of state variables; a, b, c, d – coefficients: $a = 0{,}866$, $b = 0{,}0265$, $c = 0{,}00071$, $d = 1{,}269 \cdot 10^{-5}$.

Let's consider the CWS, which consists of four atomic web services. Let's represent such components with $\{aws_1, aws_2, aws_3, aws_4\}$ set, where aws_1 calculates the $b \cdot x$, aws_2 – $c \cdot x^2$, aws_3 – $d \cdot x^3$, aws_4 – gathers the results from aws_1, aws_2, aws_3 and calculates the $g(x)$ value. Thus, aws_1, aws_2, aws_3 can function concurrently, and aws_4 component has to wait until another three components accomplish their calculations.

The architecture of the appropriate CWS is given in the Fig. 3, where the "BPEL-Engine" atomic DEVS-model is the Java-implementation of MC model (eq. 4).

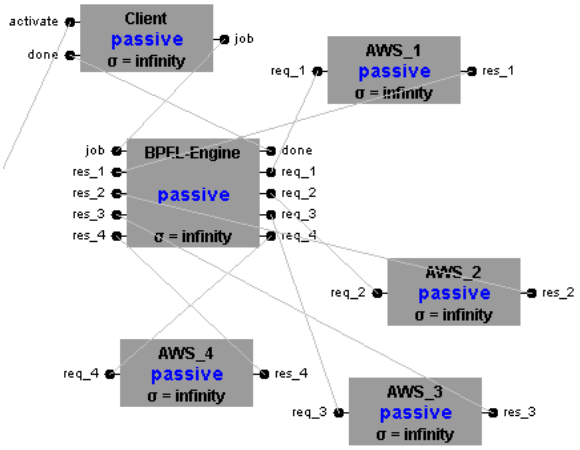

Figure 3. The resulting coupled CWS DEVS-model

In the Fig. 3 σ – time, elapsed since last transition – the DEVS Suite representation of e (eq. 3). The σ values shown mean that the simulation hasn't yet been started.

To set the non-functional properties of CWS components the Pingtest.net service has been used. As a result the following sequence of values has been formed: $0, 50, 100, 150$ ms. There are have been distinguished four cases: when components response times are equal and are set with values from the sequence (Fig. 4).

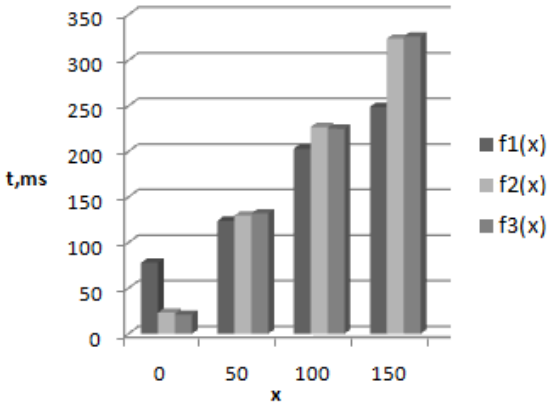

Figure 4. The results obtained

In the Fig. 4 x – values of CWS components non-functional properties; $f_1(x)$ – real time, spent on simulation-driven DEVS-based validation; $f_2(x)$ – the aggregated values of CWS non-functional property, obtained by way of DEVS-simulation; $f_3(x)$ – real time, spent on alternative testing-driven validation – the deployment/redeployment time costs haven't been taken into consideration.

It can be concluded from Fig. 4 that the bigger the values of CWS components response times

(communicational time costs) the more advantageously the proposed approach looks.

The polynomial values (eq. 6), obtained by way of DEVS-simulation were equal to the ones, obtained by way of testing. Resulting coupled CWS DEVS-model adequacy was checked with statistical t- and F-criterions.

The results obtained allow to characterize the proposed approach as applicable to the practical usage.

4 Conclusion

In this paper the simulation-driven approach for Composite Web Services validation based on DEVS- and TLA+ formalisms usage has been proposed.

The following conclusions have been made:

1. The proposed approach allows to conduct the validation of Composite Web Service with no need to deploy/redeploy the system and its components, which can potentially reduce the corresponding time costs.
2. The case study has been conducted, which has shown the applicability of the proposed approach to the practical usage. For this purpose the validation has been carried out in two ways – by way of simulation and by way of testing.
3. The bigger the orchestration-driven communication time costs the more advantageous the proposed approach looks – in comparison with testing-driven alternative way of validation.

References

Clarke, E. M., Grumberg, O., & Peled, D. (2001). Model Checking. Cambridge: The MIT Press.

Cristia, M. (2007) A TLA+ Encoding of DEVS Models. Proceedings of International Modeling and Simulation Multiconference (pp. 17–22). Buenos Aires, Argentina.

Kaur, R., & Ghumman, N. S. (2015). A Survey and Comparison of Various Cloud Simulators Available for Cloud Environment. International Journal of Advanced Research in Computer and Communication Engineering, 4(5), 605–608. doi:10.17148/IJARCCE.2015.45129

King, D. H., & Harrison, H. S. (2010). Discrete-Event Simulation in Java – a Practitioner's Experience, Proceedings of the 2010 Conference on Grand Challenges in Modeling & Simulation (GCMS 2010) (pp. 436–441). Ottawa, Ontario, Canada.

Lamport, L. (2002). Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Boston: Addison-Wesley.

Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. New Jersey: Prentice Hall.

Methni, A., Lemerre, M., Hedia, B. B., Haddad, S., & Barkaoui, K. (2014). Specifying and Verifying Concurrent C Programs with TLA+. Communications in Computer and Information Science, 476, 206–222. doi:10.1007/978-3-319-17581-2_14

Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., & Deardeuff, M. (2015). How Amazon Web Services Uses Formal Methods. Communications of the ACM, 58(4), 66–73. doi:10.1145/2699417

Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer, 40(11), 64–71.

Rahman, U., Hakeem, O., Raheem, M., Bilal, K., Khan, S. U., & Yang, L. T. (2015). Nutshell: Cloud Simulation and Current Trends, Proceedings of 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity) (pp. 77–86). IEEE. doi:10.1109/SmartCity.2015.51

Shkarupylo V. V., Kudermetov, R. K., & Polska, O. V. (2015). DEVS-oriented technique for composite web services validity checking. Radio Electronics, Computer Science, Control, 4, 79–86. doi:10.15588/1607-3274-2015-4-12

Shkarupylo V. V. (2016). A technique of DEVS-driven validation, Proceedings of 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET 2016) (pp. 495–497). Lviv, Ukraine: IEEE. doi:10.1109/TCSET.2016.7452097

Shkarupylo V. V., Tomičić, I., & Kasian, K. M. (2016). The investigation of TLC model checker properties. Journal of Information and Organizational Sciences, 40(1), 145–152.

Vohra, D. (2012). Java 7 JAX-WS Web Services: A practical, focused mini book for creating Web Services in Java 7. Birmingham-Mumbai: Packt Publishing.

Wainer, G. A., & Mosterman, P. J. (2010). Discrete-Event Modeling and Simulation: Theory and Applications. New York: CRC Press.

Web Services Business Process Execution Language Version 2.0. (2007). Retrieved from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf