# A Generic Approach for the Verification of Static and Dynamic Behavioral Properties of SCDL/WS-BPEL Service-Component Architectures

Taoufik Sakka Rouis[1(✉)], Mohamed Tahar Bhiri[2], Mourad Kmimech[3], and Layth Sliman[4]

[1] LIPAH Laboratory, FST, University of Tunis El Manar, Tunis, Tunisia
srtaoufik@yahoo.fr
[2] MIRACL Laboratory, ISIMS, University of Sfax, Sfax, Tunisia
[3] UR-OASIS Laboratory, ENIT, University of Tunis El Manar, Tunis, Tunisia
[4] Efrei - École d'Ingénieur Généraliste en Informatique, Villejuif, France

**Abstract.** Web systems verification is a crucial activity throughout the systems development life cycle, especially in the phase of service-component architectural design. Indeed, this activity allows the detection and consequently the correction of errors early in Web systems development life cycle. In this paper, we discuss the behavioral verification problem on the SCDL/WS-BPEL service-component architectures. To do so, the Wright formal ADL and the Ada concurrent language were used as a target models. To achieve this, a set of systematic translation rules are proposed. This allows the verification of the standard behavioral properties using the Wr2fdr tool. In addition, using an Ada dynamic analysis tool, we could detect the potential behavioral properties such as the deadlock of an Ada concurrent program.

**Keywords:** Verification · Concurrent program · Model-Checker
Static behavioral properties · Dynamic behavioral properties
Service-Component architecture

## 1 Introduction

Today, we witness a growing interest in the domain of service-component architecture technologies. This interest is mainly motivated by the reduction of cost and development time of complex Web systems. In the context, the Service Component Definition Language (SCDL) and the Web Service Business Process Execution Language (WS-BPEL) are the standards de-facto used in the modeling and implementing of Service-Component Architecture (SCA). This SCDL language [14] is an XML based formatted language which allows expressing all the relations in an SCA architectural element. The WS-BPEL language (abbr. BPEL) [6] offers a standard based approach to build flexible business processes by the orchestrating and choreographing of multiple Web services. In addition, it aims to model the behavior of component processes by specifying both abstract and executable business processes. It also defines an interoperable


J. H. Park et al. (Eds.): PDCAT 2018, CCIS 931, pp. 381–389, 2019.
https://doi.org/10.1007/978-981-13-5907-1_41

integration model that should facilitate the expansion of automated process integration both within and between businesses.

For several years now, these SCDL and WS-BPEL technologies appear as the powerful complementary models for the development of service-component architectures. However, they lack a formal foundation for the specification and verification of their structural, behavioral and non-functional properties. As solutions for this problem, several works have been proposed to translate these source models into another which supports specific analyzers. For example, in our previous work [7] we proposed to map SCA to Acme/Armani for the verification of the structural and non-functional properties of SCA software architecture. For the verification of the WS-BPEL behavioral specifications, numerous works proposed to translate the WS-BPEL activities into a formal technique. For example, the works presented in [3], [16] and [12] propose, respectively, to translate the WS-BPEL activities into D-LOTOS, BPMN and PetriNet.

In this work, we target a formal verification of the behavioral properties of SCDL/WS-BPEL service-component architectures. To achieve this, the model transformation approach is used to translate the source architecture to an Ada concurrent program. In this study, the Wright formal ADL is used as an intermediate modeling language. The choice of these two languages in our verification approach is justified mainly by the following three factors:

- The Wright ADL defines eleven standard properties related to the consistency of software architecture among which four -assimilated to behavioral contracts- are automated by the Wr2fdr tool [15]. The latter contracts can be checked with the FDR2 model-checker.
- The semantics similarity of the Wright process and the Ada task favors the formalization of the Wright configuration by an Ada concurrent program.
- The presence of different analysis tools related to the detection of the dynamic and specific behavioral problems of an Ada concurrent program. For example, using FLAVERS, INCA or SPARK [11], we can detect the potential behavioural properties such as the deadlock of an Ada concurrent program. In addition, using an Ada dynamic analysis tool such as GNAT Programming Studio (GPS) or its extension GNATprove [10].

The remainder of this paper is structured as follows: Sect. 2 proposes an overview of the main related works. Section 3 deals with our systematic rules allowing the translation of SCDL/WS-BPEL source software architecture to the Wright target software architecture; Sect. 4 exhibits the translation rules of the Wright software architecture into an Ada concurrent program. An overview of the main results is discussed in Sect. 5. Finally, Sect. 6 provides a conclusion and possible future work.

## 2 Related Work

The approach shared by most of the existing works in the field of Web services architectures consistency verification is the use of techniques and general tools such as B, LTS, CSP and FSP. To do so, numerous works offer more or less systematic translations of

source architecture to the target model. In this section, only the works related to the behavioral verification of the SCDL/WS-BPEL architectures are mentioned.

Yeung proposes in [18] to translate the WS-BPEL web service to CSP to verify the behavior properties of web services architecture. In this paper, formal verification can be carried out based on the notion of CSP trace-refinement and can take advantage of the FDR2 model checking. The authors of [4] propose to use the FSP formal language to check if a web service composition implemented in WS-BPEL satisfies a web service composition specification captured by Message Sequence Charts (MSCs). Both the WS-BPEL process and the MSC are translated to FSPs. Each FSP represents a finite labelled transition system. Using the LTSA model checking tool, this FSP target specification can check the safety and progress properties as well as properties expressed in the LTL logic. In [5], Foster et al. use this LTSA to check the compatibility of web service compositions in WS-BPEL. Since the semantics of Petri Nets is formally defined by mapping each WS-BPEL process to a Petri net, a formal model of WS-BPEL can be obtained. This approach has been followed in several works. For example, in [12] Verbeek et al. formalize some WS-BPEL activities used for the orchestration of Web services as a class of Petri Net called workflow nets. For this class of Petri nets, a verification tool named Wolfan has been developed. This tool can verify properties such as the termination of a workflow net and detection of nodes that can never be activated. The authors of [17] propose to map most of the basic and structured activities of WS-BPEL and the Web Service Choreography Interface (WSCI) to Coloured Petri Nets (CPN). In [8] Hamel et al. propose to use the Event-B method to check the structural and behavioural properties of an SCA component assembly. To achieve this, the B-Invariant and B-Event are profitably used to formalize the patterns proposed by Barros [1]. Using the ProB animator, Hamel et al. [8] validate their formal approach on an event-B specification.

## 3 Translation of SCDL/WS-BPEL to Wright

In this section we propose a set of rules allowing the translation of SCA software source architecture to a Wright target architecture. This allows the verification of standard behavioral properties supported by the Wr2fdr tool accompanying the Wright ADL.

Regarding the structural aspect, an SCA software architecture is generally described in an XML SCDL file. The latter expresses all the relations in a composite. In this context, a composite is an assembly of heterogeneous components. Each SCDL component is based on a common set of abstractions such as services, references and properties. In the context, services and references describe, respectively, what a component provides and what a component requires from its external environment. These services and references can be matched with bindings. Hence, each SCDL markup can be specified in Wright as follows:

- An SCDL composite can be translated to a Wright configuration;
- An SCDL component can be translated to a Wright component;

- An SCDL component's reference can be translated to a Wright port with the same name;
- An SCDL component's service can be translated to a Wright port with the same name;
- An SCDL wire connects two SCA components. Hence, we propose to translate an SCDL wire to a Wright connector that proposes two roles.

Concerning the WS-BPEL behavioral descriptions, we propose to translate each WS-BPEL process by a CSP process. In this translation, each primitive activity is translated to a CSP event. Since WS-BPEL provides three kinds of activities, we suggest translating each activity by a specific event as follows:

- An <invoke> activity is used to initialize an appeal of an operation Oper. This activity can be modeled in CSP by an initialized event as follows: _invokeOper
- A <receive> activity is used to wait for a message from an external operation Oper. This observed activity can be modeled in CSP by an observed event as follows: receiveOper
- A <reply> activity is used to initialize a response to an external operation Oper. This activity can be modeled in CSP by an initialized event as follows: _replyOper

In addition, WS-BPEL provides typical structured activities such as: <sequence>, <flow>, <terminate>, <if>, <switch>, <while>, <repeatUntil>, etc. These control structures can express a causal relationship between multiple invocations by means of control and data flow links. For the WS-BPEL control structures, we propose the following translation rules:

- The <sequence> construct is used in WS-BPEL wherever a series of activities needs to occur sequentially, although they may be contained one or more times within looping or concurrent construct activities. This <sequence> construct can be modeled in CSP by a set of events separated by the prefixing operator (->).
- Concurrency in WS-BPEL permits us to model the concurrent transitions in the message sequence charts. In WS-BPEL, this is specified using the <folw> construct. However, the concurrency in CSP is modeled by the parallel composition operator (| |). Hence, using the CSP parallel operator (| |), we can model the WS-BPEL flow activities by a set of concurrent processes.
- In WS-BPEL, the conditional branching introduces decision points to control the execution flow of a process. Each conditional structure such as <if> or <switch> can be modeled in CSP by the adequate choice operator:
  - ([ ]) deterministic choice operator: if the choice between these activities is an external choice. In other words, if these activities are observed (receive activity).
  - (|~|) nondeterministic choice operator: if the choice between these activities is an internal choice. In other words, if these activities are initialized (invoke or reply activity).
- In WS-BPEL, as in most programming languages, loops are used to repeat activities. Each looping structure such as <forEach>, <while> or <repeatUntil> can be modeled in CSP by a recurrent process as follows: P = … ->P.

## 4 Translation of Wright to Ada

The main structural concepts treated in this section are: configuration, component, connector, port, role, computation, glue, attachments, process, initialized event, observed event, successfully terminated event, prefixing operator, deterministic choice operator and nondeterministic choice operator. To achieve this, we proposed an Ada package called ArchWright allowing the representation in Ada of the main structural concepts coming from the Wright ADL. Hence, a Wright configuration can be translated to an Ada concurrent program using the ArchWright package. For traceability reasons, we keep the same identifiers used in the Wright specification.

The Wright ADL is one of the first approaches allowing the description of the behavioral aspect of architectural elements. Indeed, the behavior of a Wright component (respectively of a connector) is described locally through the ports (respectively roles) and, generally, through a computation (glue respectively) using a CSP process algebra. Hence, in this work, we propose to implement each:

- Wright configuration by an Ada concurrent program using the ArchWright package.
- Wright component by an Ada record compound with two fields: (1) Ports that represents the component's ports. It is modeled by an array of CSPTask, where the CSPTask is the task type proposed to implement in Ada the CSP process; (2) Computation that represents the component computation. This field can be modeled by a single CSPTask.
- Wright connector by an Ada record compound with two fields: (1) Roles that represents the connector roles. It is modeled by an array of CSPTask; (2) Glue that represents the glue of this connector. This field can be modeled by a single CSPTask.

Table 1 illustrates the principle of the translation of the main Wright architectural elements into Ada. For traceability reasons, we keep the same identifiers used in the Wright specification.

Regarding the CSP behavioral concepts, a simple CSP process can be compound with a set of observed and initialized events separate with the specific CSP prefixing operator. However the Ada concurrent language defines a powerful behavioral tasking model. An Ada task represents the basic element of each Ada concurrent program. It consists of two parts: task specification (declaration) and task body. This task specification can provide a set of services (called entries). These entries can have in, out or in out formal parameters. Each entry exported by an Ada task indicates the possibilities of an Ada rendezvous. Based on the similarity between the CSP process and the Ada task, we offer an intuitive correspondence provided below for translating a CSP process to an Ada task:

- A CSP process leads to an Ada task;
- A CSP event naturally corresponds to an Ada entry. In order to differentiate between an observed and an initialized event, we propose to use the same prefixed notation used in CSP: an observed event is denoted by (e) and an initialized event is denoted by (_e)
- The recursion operator can be translated by an Ada loop;

**Table 1.** Ada formalization of the Wright concepts

| Wright | Ada |
|---|---|
| Component | `type CompWright (portsNumber : natural) is record Ports: array (portsNumber) of CSPTask; Computaion: CSPTask; end record;` |
| Connector | `type ConnectorWright (rolesNumber : natural) is record Roles: array (rolesNumber) of CSPTask; Glue: CSPTask; end record;` |
| Configuration | `with ArchWright; use ArchWright; procedure Configuration is … begin … end Configuration ;` |

- The CSP prefixing operator (->) can be specified by the Ada sequential instruction;
- The CSP successfully terminated event (denoted by TICK or §) can be implemented with the Ada "terminate" instruction.
- The CSP internal choice operator (denoted by Π or | ∼|) allows the future evolution of a process to be defined as a choice between two sub-processes, but does not allow the environment any control over which one of the component processes will be selected. This internal choice can be implemented in Ada with a simple conditional structure (if). If we have multiple composite processes, the Ada conditional structure (case) can be used.
- The CSP external choice operator ([ ]) allows the future evolution of a process to be defined as a choice between two sub processes, and allows the environment to resolve the choice by communicating an initial event for one of the processes. This deterministic choice can be implemented with the Ada "select" instruction.

Table 2 illustrates these proposed rules allowing the translation of a CSP specification to Ada.

Each attachment of a component's port with a connector's role can be implemented in Ada by a sequence of entries call of the events specified in the interconnected port/role. The general form of the entries call (event) is specified as follows:

- If the port entry corresponds to an initialized event: we call this entry, then we call the similar entry of role (Component.port._event; Connector.role._event;).
- Else, if this entry corresponds to an observed event, we call this entry after the call of the similar entry in the role (Connector.role.event; Component.port.event;).

**Table 2.** Ada formalization of the CSP process

| CSP | Ada |
|---|---|
| P = _request -> result -> $ | task P is<br>entry _request; entry result;<br>end P ;<br>task body P is<br>accept _request; accept result;<br>end P; |
| P1 \|~\| P2 | if internCondition then P1<br>else P2<br>end if; |
| P1 [ ] P2 [ ] TICK | select P1; or P2; or terminate ;<br>end select; |

## 5 Discussions and Results Summary

The means to establish connections between software architecture and a concurrent language like Ada are limited. For example, Naumovich et al. [13] offer a manual translation of Wright into Ada without explanation rules. In our previous work [2] we established a set of simple rules allowing translating Wright software architecture into Ada. In this study, a significant improvement of our translation rules was proposed.

Our verification approach is validated on several uses cases available at our SourceForge repository [9]. The main advantage of our verification approach can be summary as follows:

- The first translation (SCDL/WS-BPEL to Wright): allows the verification of eleven standard properties related to the consistency of software architecture, among which four, assimilated to behavioral contracts, are automated by our Wr2fdr tool [15]. The latter contracts can be checked with the FDR2 model checker.
- The second translation (Wright to Ada): can be used in the verification of the specific behavioral properties of the source description using static and dynamic analysis tools associated with Ada such as test data generator and debugging. In addition, the refinement of the abstract architecture, allows step by step obtaining a coherent concrete architecture vis-à-vis the verified abstract architecture. The correction of each refinement step can be ensured by the static analysis tools associated with Ada: the refined concurrent program must keep the same properties checked on the abstract concurrent program.

## 6 Conclusion

In this paper, an approach for verifying the behavioral coherence of SCDL/WS-BPEL component-service architectures has been proposed. To achieve this, it has been proposed to map SCDL/WS-BPEL to the Wright ADL, thus allowing the checking of the

standard properties supported by this ADL. As a second step, a set of exogenous translation rules allowing the translation from the Wright specification to an Ada concurrent program has been put forward. The choice of this Ada language is justified by the presence of many Ada analysis tools able to detect several error types. Versus the properties to be checked, we must choose the adequate Ada analysis tool. For example, FLAVERS and INCA tools promote the property-oriented trace, while the SPIN and SMV tools promote the property-oriented state.

Currently, we are extending this work by an automation of these translation rules using the Xtext, ATL and Xpand model transformation languages.

## References

1. Barros, O.: Business process patterns and frameworks: reusing knowledge in process innovation. Bus. Process Manag. J. **13**(1), 47–69 (2007)
2. Bhiri, M.T., Fourati, F., Kmimech, M., Graiet, M.: Transformation exogène de Wright vers Ada. Technique et Science Informatiques **31**(7), 839–868 (2012)
3. Chama, I.E., Belala, N., Saïdouni, D.E.: Formalizing timed BPEL by D-LOTOS. IJERTCS J. **5**(2), 1–21 (2014)
4. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based verification of web service compositions. In: Proceedings of 18th IEEE International Conference on Automated Software Engineering, Montreal, Canada, pp. 152–163. IEEE, October 2003
5. Foster, H., Uchitel, S. Magee, J., Kramer, J., Magee, J.: Compatibility verification for web service choreography. In: Proceedings of the IEEE International Conference on Web Services, San Diego, CA, USA, pp. 738–741. IEEE, June 2004
6. Maâlej, A.J., Lahami, M., Krichen, M., Jmaïel, M.: Distributed and resource-aware load testing of WS-BPEL compositions. In: ICEIS (2), pp. 29–38 (2018)
7. Haddad, I., Kmimech, M., Sakka Rouis, T., Bhiri, M.T.: Towards a practical approach to check service component architecture. In: 11th International Conference on Semantics, Knowledge and Grid, pp. 65–72. IEEE (2015)
8. Hamel, L.M., Graiet, G.M., Kmimech, M.: Formal modeling for verifying SCA composion. In: RCIS Conference, pp. 193–204 (2015)
9. https://sourceforge.net/projects/SCA2WrightToAda (2018)
10. Hoang, D., Moy, Y., Wallenburg, A., Chapman, R.: SPARK 2014 and GNATprove - A competition report from builders of an industrial-strength verifying compiler. Int. J. Softw. Tools Technol. Transf. **17**(6), 695–707 (2015)
11. Maalej, Maroua, Taft, Tucker, Moy, Yannick: Safe dynamic memory management in ada and SPARK. In: Casimiro, António, Ferreira, Pedro M. (eds.) Ada-Europe 2018. LNCS, vol. 10873, pp. 37–52. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92432-8_3
12. Mateo, J.A., Ruiz, V.V., Macià, H., Díaz, G.: A coloured petri net approach to model and analyse stateful workflows based on WS-BPEL and WSRF. In: SEFM Workshops, pp. 389–404 (2014)
13. Naumovich, G., Avrunin, G.S., Clarke, L.A., Osterweil, L.J., Applying static analysis to software architectures'. In: ACM SIGSOFT 1997, Software Engineering Notes, vol. 22(6), pp. 77–93 (1997)
14. OASIS. Service Component Architecture Assembly Model Specification Version 1.1. Oasis, July 2017. https://www.oasis-open.org/standards

15. Sakka Rouis, T., Bhiri, M.T., Kmimech, M., Moussa, F.: Wr2Fdr Tool Maintenance for models Checking. In: SoMeT Conference, pp. 425–440 (2017)
16. Strobl, S., Zoffi, M., Bernhart, M., Grechenig, T.: A tiered approach towards an incremental BPEL to BPMN 2.0 Migration. In: ICSME Conference, pp. 563–567 (2016)
17. Yang, Y., Tan, Q., Xiao, Y., Liu, F., Yu, J.: Transform BPEL workflow into hierarchical CP-nets to make tool support for verification. In: APWeb Conference, pp. 275–284 (2006)
18. Yeung, W.L.: Mapping WS-CDL and BPEL into CSP for behavioural specification and verification of web services. In: ECOWS, IEEE Computer, pp. 297–305 (2006)