Model-Checking Verification for Reliable Web Service

Shin NAKAJIMA
Hosei University and PRESTO, JST
nkjm@i.hosei.ac.jp

Abstract

Model-checking is a promising technique for the verification and validation of software systems. Web service, an emerging technology in the Internet, is an autonomous server that may offer an individual service. It sometimes requires to combine more than one to meet our requirements. WSFL(Web Services Flow Language) is proposed as a language to provide means to describe Web service aggregation. We are interested in how much the software model-checking technique can be used as a basis for raising reliability of Web service, Web service flow descriptions in particular. Our experience shows that faulty flow descriptions can be identified with the proposed method. The method is also very helpful in studying an alternative semantics of the WSFL in regard to the handling of dataflows.

1 Introduction

Model-checking is a promising technique for the verification and validation of software systems [2][5]. The technique is applied to software requirement specification and design specification and aims to increase the reliability and productivity from early stages of the software development. As the number of the success cases increases, the technique becomes one of the basic tools for the use in the development of a wide variety of software [9][13].

As *the Internet* becomes an essential tool for our daily activities, Web service, a new form of software technology to use remote services, is emerging [15]. In order to use Web service, we find appropriate Web service providers that are located somewhere in the network environment. And we initiate service interactions with the providers. Since each provider may offer an individual service, it sometimes needs to combine more than one to meet our requirements.

Service aggregation is a key concept of the technology to combine existing Web services, and it needs a language to describe how various Web services are composed. The description is essentially a collaboration of distributed autonomous computing entities. And writing correct flow descriptions is not a easy task. The verification of the Web service flow prior to its execution in *the Internet* is mandatory [10].

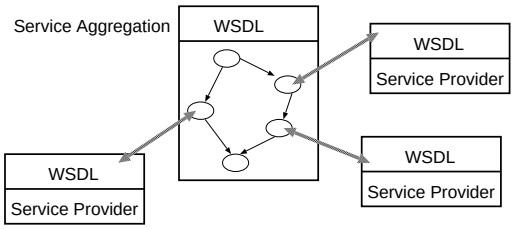


Figure 1. Web Service Aggregation

We are interested in how much the software model-checking technique can be used as a basis for raising reliability of Web service, Web service flow descriptions in particular. We have conducted some experiments to model-check the description written in WSFL (Web Services Flow Language) [8]. Since WSFL is a net-oriented specification language, any WSFL descriptions, though syntactically correct, sometimes show faulty global behaviours. Our experience shows that such faulty descriptions can be identified with the method. And the method is very helpful in studying an alternative semantics of the WSFL in regard to the handling of dataflows.

2 Web Service Flows

2.1 Web Service Aggregation

A concept of service aggregation, combining existing Web services, is an important feature of the Web service technology. Figure 1 presents an illustration of the Web service aggregation. When we enter the world of *the Internet*, we see a lot of the Web services ready for us to use. We want to combine flexibly some of the existing Web services as our own needs change, which is conducted in a somewhat ad-hoc manner. In its essence, the aggregation is a collaboration of many Web service providers. Each service provider is a self-contained software system having its

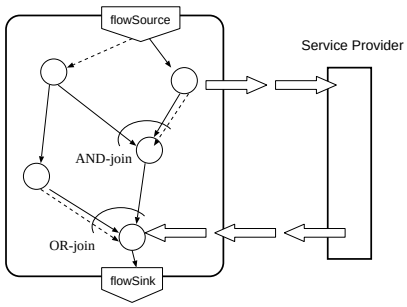


Figure 2. WSFL Basis

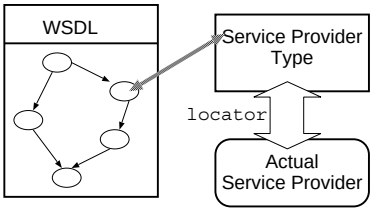


Figure 3. Late Binding

own threads of control. The description for the aggregation needs an explicit notion of both control and data flows between the Web services, that faithfully reflects the causal structure of the aggregation. Two languages, WSFL [8] and XLANG [14], for describing Web service flows have been proposed. The aim of both languages is to augment WSDL [1] with the behavioural aspect of the flows.

By using one of the languages, we can express our own requirements as "programs" or flow descriptions. It, however, needs a great care on submitting the descriptions to execute in *the Internet*. The flow descriptions may contain "bugs" because it is probable to write such flow descriptions in an ad-hoc manner. Executing a buggy flow description, consumes tremendous amount of network resources that are shared publicly. Thus, verifying the Web service flow prior to its execution in *the Internet* is mandatory [10].

2.2 WSFL

WSFL (Web Services Flow Language) [8] is a net-oriented specification language, originated from a model of a workflow description language [7]. Each service invocation corresponds to a "task" of workflow and both control and data flows are represented as "arcs" relating task nodes. WSFL is meant to be a behavioural extension of WSDL, and is equipped with an XML-based concrete syntax.

The core part of the behavioural aspect is a WSFL flow model. Figure 2 shows a schematic illustration of a simple example. Essentially the WSFL flow model shows how

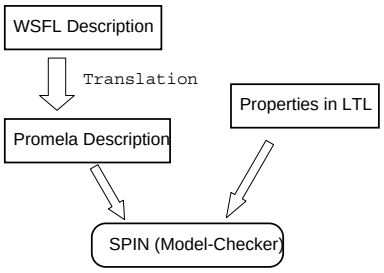


Figure 4. Verification Process

the description combines the necessary existing Web service providers. An invocation of Web service, called a flow activity depicted as a circle in Figure 2, corresponds to a workflow node. Each activity may have an event of invoking some external service provider, accepting notification from the outside, or performing an internal action. Here, an internal action is assumed to be implemented in the platform that the flow model is executing.

WSFL allows the late-binding of the actual service provider. Figure 3 illustrates the situation where a WSFL activity provides a binding information in the form of the `locator`. The binding of the service provider type and the actual provider software is done during the execution of the flow. And this allows a room for the flexibility necessary for the Web service in that selecting the actual service provider can be delayed as late as possible. On the other hand, the activity uses the information on the service provider type to invoke the provider, and the flow assumes that the actual provider obeys the type information. Thus, the logical correctness of the flow can be checked irrespective of the actual service provider entity.

The specification document [8] describes the operational semantics of the WSFL flow model. The semantics is based on the PM-flow [7] proposed by the same author as a workflow schema language. In a word, WSFL can be considered as an XML-based workflow schema language following the PM-flow model. The operational semantics is essentially a set of rules that select activities to be fired. More than one activity are simultaneously able to fire, which is the source of concurrency.

3 Verification Method

3.1 Overview

WSFL is essentially a workflow schema description language that is based on the net-oriented specification paradigm. The verification problem, thus, turns out to be one similar to the verification of workflow or business flow

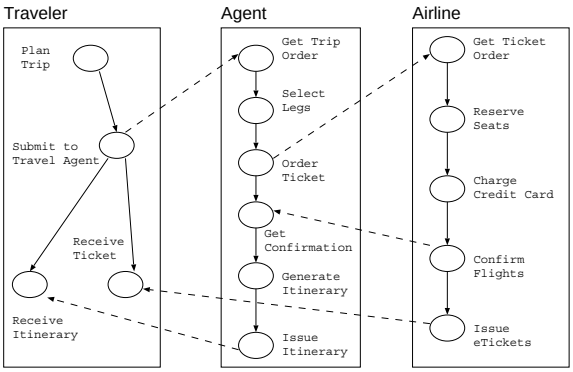


Figure 5. Ticket Order Example

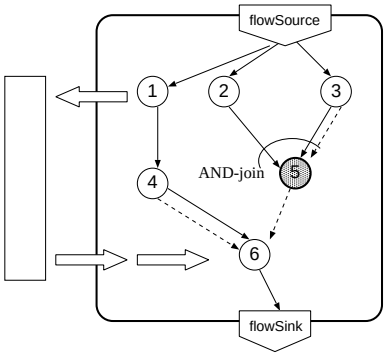


Figure 6. Deadlocked Flow Model

[4][6]. We use SPIN model-checker in the following discussion.

SPIN is an automaton-based model-checker developed by G. J. Holzmann [5]. SPIN provides a specification language Promela that describes the target system to be a collection of Promela processes (automata) with channel communications.

Figure 4 shows the verification process. The flow description written in WSFL is translated into Promela, the input specification language of SPIN. The formalization mainly concerns with the translation of a WSFL flow model into Promela processes [12]. The Promela processes should faithfully encode the operational semantics of WSFL. The properties to be checked are reachability, deadlock-freedom, or application specific progress properties. The application specific properties are expressed as formulas of LTL (Linear Temporal Logic), which are also fed into SPIN. An LTL formula can have `[]` (always), `<>` (eventually) and `U` (strong until) as the temporal operators.

3.2 Property Checking Examples

Figure 5 is a schematic illustration of the example derived from the specification document [8]. The example has three WSFL flow models, `Traveler`, `Agent`, and `AirLine`. The Promela description is obtained by following the translation method reported in [12]. The code size is about 700 lines of Promela source text. The aggregate flow turns to be a large transition system with about 280 thousands of states and about 470 thousands of transitions. It is because three flow models are analyzed as if they execute concurrently, while each flow executes almost sequentially as Figure 5 suggests. Actually the size of each flow model is small. `Airline`, for example, has only 201 states and 586 transitions.

As an example of application-specific progress properties, the formula below for the case of `Traveler` is checked. It says that it is always (`[]`) the case that if there is an event of `SubmitToTravelAgent` then eventually (`<>`) `ReceiveTicket` and `ReceiveItinerary` are followed.

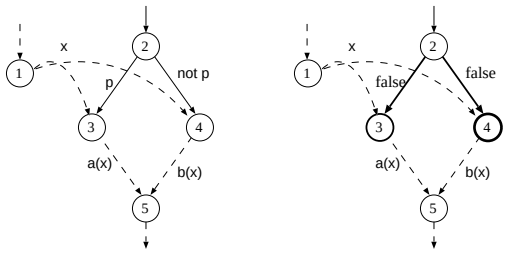
```
[](SubmitToTravelAgent ->
     (<>ReceiveTicket && <>ReceiveItinerary)
```

Next, consider an example WSFL model that is syntactically correct but shows a faulty behaviour (Figure 6). The flow model consists of six activities numbered from `1` to `6`. The activity `5` has two input control links, coming from the activities `2` and `3`, and a join condition of `AND`. It further has an output data link to the activity `6`. The activity `6` has a control link from the activity `4` and two data links from `4` and `5`. According to the operational semantics, the activity `5` can be fired when the two input control links have definite values and also the join condition becomes $true$. The activity `5`, after the completion of its internal activity body, puts out some data on the data link to the activity `6`.

Imagine a situation that one of the input control links to the activity `5` is $false$ and thus the join condition is not satisfied. The activity `5` is not put into execution, and the data link input to the activity `6` does not have a definite value. It results in a faulty situation where the activity `6` waits for the value indefinitely. The model-checker can detect the faulty situation as a *deadlock*.

3.3 An Alternative Semantics for Verification

WSFL is a net-oriented specification language and allows two syntactic elements, control and data links to combine in an independent manner. It results in a flow graph not having consistent control and data links. WSFL, further, excludes some flow graphs that are logically correct. It is because WSFL does not have operational semantics to handle dataflows properly, which it is related to DPE (Dead-Path Elimination) discussed in [8].



(a) Valid (b) Invalid

Figure 7. Conditional Expressions

Figure 7 shows a simple example to discuss the issue. The flow model is a diagram notation of the following conditional expression.

```
if p then a(x) else b(x) endif
```

Consider a case where `p` is $true$ (Figure 7 (a)). We expect that the activity `3` becomes able to fire and that the resultant value `a(x)` flows into the activity `5`. On the other hand, the activity `4` is not fired because its input control link is $false$ (`not p`). Since the flow is an implementation of the conditional expression above, the activity `5` is supposed to consume the data from the activity `3` even if the data link from `4` is not definite. According to the operational semantics of WSFL, the activity `5`, however, is not fired at all. It is because the activity `5` does not have any input control link and thus is not selected as a candidate for the execution.

Figure 7 (b) illustrates a case where both control links from the activity `2` become $false$, which is actually a "bug." The bug causes the activity `5` will never be fired. The operational semantics of WSFL starts DPE from the activities `3` and `4` because the two are not fired due to the $false$ value of their input control links. The DPE propagates downward through the control links. It never reaches the activity `5`, since the activity `5` is linked via data links only. Therefore, the activity `5` is not salvaged by means of DPE, and will never be fired. In sum, according to the WSFL semantics, the two different situations in Figure 7 shows the same behaviour in that the activity `5` will never be fired although (a) is valid and (b) is invalid.

From the examples in Figure 6 and Figure 7, the issues with the current WSFL specification can be summarized from two viewpoints. First, syntactically, WSFL provides control and data links as two independent language constructs. We can construct WSFL flow models that have control and data links related in an arbitrarily manner, while the two kinds of links are closely related from the operational meanings. Second, two different things on "propagation" are mixed up in the operational semantics of WSFL; the propagation of some concrete value and the meta-level propagation of an event that a value becomes definite at some point are confused.

We have proposed a generalized DPE that deals with meta level propagation along data links as well as control ones [11].

4 Discussions

In WSFL and PM-graph, the control link is the major player in the navigation algorithm, and the data link is an auxiliary one. Workflow schema languages, including WPDL standardized in WfMC [16], generally put emphasis on the control flows between the activities. Data are assumed to be maintained outside of the flow model. For the case of WSFL, a typical example is shown in Figure 2. An activity invokes a service provider and expects it to calculate and store some data values. Then, another activity at a downstream side obtains the result data value from the service provider. A major concern of the flow model is the control flow between the two activities. Further, the service provider may maintain the values and offer them to the requesting activities at anytime. Alternatively, we explicitly add control links to data links because dataflow generally has implicit control flow. The Ticket Order Example (Figure 5) is written in this manner [8].

The WSFL specification document, however, does not mention anything about the rules on the use of control and data links. Thus, we, in a careless manner, construct flow models that behave differently dependent on whether a data link has a definite value or not. This leaves a large chance of introducing faults in the WSFL flow descriptions.

Ideally we guarantee that any syntactically correct flow model should show no faulty global behaviour. However, it is difficult for the case of WSFL and other net-oriented specification languages. It needs a high level concurrent language that is more abstract than the net-oriented one. If we use a net-oriented specification language like WSFL, we must take a great care on using control and data links. And a tool is needed for a global behavioural analysis to identify anomalies in syntactically correct WSFL descriptions. The SPIN model checker can be used as the engine of such tools.

In August 2002, BPEL4WS (Business Process Execution Language for Web Services) [3] has been announced as a convergence of the ideas in WSFL [8] and XLANG [14]. BPEL4WS provides flow activities (`<flow>`) to express concurrency and synchronization. Links attached to the flow activities (`<link>`) represent control flows. And data are transferred via containers (`<container>`). Container is a new syntactic construct to explicitly represent data store outside the flow actvities. BPEL4WS provides further types of activities that include a switch (`<switch>`) to represent conditional flow controls. In sum, the flow model in BPEL4WS considers problems of the specification of the

WSFL flow model, that Section 4 refers to. The BPEL4WS specification document [3], however, does not have descriptions in regard to the operational semantics, but provides explanations through several examples only. Our method to use model-checkers to study the operational semantics of the flow-based specification language is supposed to be useful to work out the precise operational semantics of BPEL4WS.

5 Position Summary

Our position is that the software model-checking technique is useful in the case of Web service flows as well as the EJB [9][13]. Here, EJB is often chosen as the middleware for the implementation of the Web service server. We think that the method of using model-checkers is valuable to study and clarify behavioral semantics of BPEL4WS.

References

[1] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Service Description Language (WSDL). W3C Web Site, 2001.

[2] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, 1999.

[3] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services (v1.0). July 2002.

[4] H. Eertink, W. Janssen, P.O. Luttighuis, W. Teeuw, and C. Vissers. A Business Process Design Language, In *Proc. FME FM'99*, pages 76–95, September 1999.

[5] G.J. Holzmann. The Model Checker SPIN. *IEEE Trans. Soft. Engin.*, Vol.23, No.5, pages 279–295, May 1997.

[6] C. Karamanolis, D. Giannakopoulou, J. Magee, and S.M. Wheater. Model Checking of Workflow Schemas. In *Proc. IEEE EDOC 2000*, pages 170–179, September 2000.

[7] F. Leymann and W. Altenhuber. Managing Business Processes as an Information Resource. *IBM System Journal*, vol.33, no.2, pages 326–348, 1994.

[8] F. Leymann. Web Services Flow Language (WSFL 1.0). IBM Corporation, May 2001.

[9] S. Nakajima and T. Tamai. Behavioural Analysis of the Enterprise JavaBeansTM Component Architecture. In *Proc. 8th SPIN Workshop*, pages 163–182, May 2001.

[10] S. Nakajima. On Verifying Web Service Flows. In *Proc. SAINT 2002 Workshop*, pages 223–224, January 2002. Presented at WebSE 2002 [15].

[11] S. Nakajima. Behavioural Analysis of Web Service Flows with Model-Checking Techniques. TR-DSE-02-002, Hosei University, August 2002.

[12] S. Nakajima. Verification of Web Service Flows with Model-Checking Techniques. In *Proc. CW 2002*, IEEE, to appear, November 2002.

[13] S. Nakajima. Behavioural Analysis of Component Framework with Multi-Valued Transition System. In *Proc. APSEC 2002*, IEEE, to appear, December 2002.

[14] S. Thatte. XLANG – Web Services for Business Process Design. Microsoft Corporation, May 2001.

[15] WebSE 2002. International Workshop on Web Service Engineering. Nara, February 2002.

[16] Workflow Management Coalition. Interface 1: Process Definition Language Process Model. WfMC TC-1016-P (v1.1), October 1999.