TOWARDS FORMAL VERIFICATION OF WEB SERVICE COMPOSITION

Xinxin Bai[1], Yushun Fan[1]

[1]Department of Automation, Tsinghua University, China

ABSTRACT

Correct notation for specifying the composite service alone is not sufficient to guarantee reliability, thus comes a strong demand on modeling and verifying web services composition languages. This paper presented a formal method of web service composition verification based on Petri net. Focus is given on BPEL4WS specification, of which both behavior semantics and communication semantics are covered. Meta-model transformation rules are given and automatic verification process is also introduced.

Keywords. Web Service Composition Verification, BPEL4WS, Workflow, Petri Net.

INTRODUCTION

Currently, service-oriented architecture (SOA) is becoming the most popular software architecture of modern enterprise applications, and one important technique of its implementation is Web Services. Individual service offered by some service provider may represent limited business functionality, however, by composing different individual services from different service providers, a composite service describing the entire business process of an enterprise can be made. Thus web service composition has been attached great importance in enterprise application integration (EAI).

Many Web Service workflow languages, such as WSCI, BPML, BPEL4WS, BPSS, etc., have emerged to support long running and multi-service transactions in the case of web service composition, among which two languages are representative, that is, BPEL4WS [Tony et al (1)] and DAML-S [Ankolenkar et al (2)]. The former comes from the industrial world while the latter from the semantic web. However, correct notation for specifying the composite service alone is not sufficient to guarantee reliability. It is also important that the specification representing the web services composition translates into an error-free flow when the composite service executes [Muhammad (3)]. Thus comes a strong demand on modeling and verifying web services composition languages. In this paper, we propose a formal approach based on Petri Net to contribute to the verification work of web services composition. BPEL4WS language is concentrated on.

RELATED WORK

Several works have been done to verify web service workflow languages. They cover different source languages and use different techniques:

In [Srini and Sheila (6)], DAML-S semantics is described in terms of a first-order logical language, situation calculus (SC), and then service descriptions are encoded in Petri Net formalism. In this case, web services can be simulated, verified and composed.

While, in the industry world, Petri Net based method [Rachid and Roualem (5), Thomas et al (7), Aalst et al (8)] is also introduced to capture semantics of WSDL and XRL. Besides, model-checking techniques have been adopted in WSFL verification [Shin (4)]. In [Howard et al (9)], specification of design in the form of Message Sequence Charts (MSCs) and corresponding workflow implementation in BPEL4WS are both translated into finite state processes (FSP), then trace equivalence between them is verified by labeled transition system analyzer (LTSA). Further research taking into account distributed process interactions from multiple parties is conducted in [Howard et al (10)] based on (9).

Web services composition languages in the semantic realm (such as DAML-S) requires detailed portrayal of resources used and goals achieved by services that cannot be easily caught, therefore, we focus on BPEL4WS, the most emerging standard comes from industry domain representing the merging of WSFL and XLANG. Petri Net, as both a formal and graphic modeling language with abundant and mature analysis techniques, is rather proper for workflow modeling and analysis; what's more important, it is one of the roots of BPEL4WS inheriting from that of WSFL, hence we use it to model and verify BPEL4WS. Recent proposed Petri net semantic for BPEL4WS [Karsten and Christian (15)] does not address the composite process verification, automatic transformation is not covered either.

OVERVIEW OF BPEL4WS

Business Process Execution Language for Web Services (BPEL4WS) is defined for specifying business process behavior based on Web Services (1). It builds upon IBM's WSFL (Web Services Flow Language) and Microsoft's XLANG (Web Services for Business Process Design). Thereby, it integrates the features of a block structured process language (XLANG) rooted in Pi-calculus with those of a graph-based process language (WSFL) rooted in Petri Net.

BPEL4WS is layered directly on top of WSDL, in which format messages, operations, port types, partners and other information like service link types and correlations referred by process definition are represented. A concrete WSDL example is shown in Fig.1 (1). Message describes the data being exchanged abstractly; port type defines a collection of operations, whose name represents the method that is called; operation defines a combination of input, output, and fault messages; binding explains the protocol being used to carry the Web Service communication; service consists of a collection of ports each of which specifies an address for a binding. And a partner link type characterizes the conversational relationship between two services.

```
<definitions
    <message name="riskAssessmentMessage">
        <part name="level" type="xsd:string"/>
    </message>
    <portType name="loanServicePT">
        <operation name="request">
          <input message="lns:creditInformationMessage"/>
          <output message="lns:approvalMessage"/>
          <fault name="unableToHandleRequest"
                message="lns:errorMessage"/>
        </operation>
    </portType>
    <plnk:partnerLinkType
        name="riskAssessmentLinkType">
      <plnk:role name="assessor">
          <plnk:portType name="lns:riskAssessmentPT:/>
      </plnk:role>
    </plnk:partnerLinkType>
    <binding …> … </binding>
    <service name="LoanAssessor">…</service>
</definitions>
```

Figure 1: An example for process reference definition in WSDL

In BPEL4WS, the partnerLink declarations specify the shape of the relationship that BPEL4WS process will employ in its behavior. In service interaction, BPEL4WS process takes on the "MyRole" role, while its partner bears the "PartnerRole" role.

In general, BPEL4WS is grouped into four important parts: basic activities, structured activities, data handling and scopes, as illustrated in Table 1. Since BPEL4WS adopts most of the functionality present in workflow system, as illustrated in structured activities, such as <Sequence>, <Switch>, <while> and <Flow>, it is just "Old Wine in New Bottles" [Aalst et al (11)]. At the same time, it also provide more explicit support for the basic communication patterns, as basic activities representing, for example: <Invoke> (invoking a web service operation), <Receive> (receive a message from its partner) and <Reply> (send a response to a request previously accepted through <Receive>). <Variable>s hold messages that constitute the state of a business process. Further more, BPEL4WS introduces systematic mechanisms for dealing with business exceptions and processing faults.

Table 1: Core concepts of BPEL4WS

Basic Activities	Structured Activities	Data Handling	Scopes
<Invoke> <Receive> <Reply> <Assign> <Throw> <Wait> <Empty> <Terminate>	<Sequence> <Switch> <While> <Pick> <Flow>	<Expression> <Variable>	<Compensation Handler> <Fault Handler> <Event Handler>

PETRI NET AND ITS ANALYSIS ABILITY

Petri Net is both a formal and graphic modeling language. Despite new concepts, such as color, hierarchy, stochastic concept etc., are fused into Petri Net in succession, the essence of syntax and dynamic behavior properties never changes. The basic properties which can be studied with Petri Net can be divided into two groups: the behavioral properties, including reachability, boundedenss, liveness, reversibility, coverability, persistent, synchronic distance and fariness, etc.; and the structural properties, such as structural liveness, controllability, structural boundedness,. conservativeness, repetitiveness, consistency and so on. So within the transformation process, the content to be verified in the BPEL4WS models should be translated into one property or the combination of several properties of the corresponding Petri Net that can be analyzed by existing approaches.

VERIFICATION METHOD

This section presents an approach to the verification of BPEL4WS. After translating the operational semantics of BPEL4WS into Petri Net, mature Petri Net analysis techniques can be used to prove some certain properties of the initial model.

Motivation of verification

The industry has high performance and availability requirements on Web Services when they are deployed in the real network environment. These requirements are grounded on the correctness of composed business processes. Some efforts are reflected in the BPEL4WS specifications; however, there is currently no evidence that BPEL4WS is based on a formal semantics. Also, as far as a concrete process model described in BPEL4WS is concerned, different partners may have different expected functions towards the whole composed web services; more specific, each partner concerns not only the correct behaviors of its corresponding partners, but the desired functions of its own process. To sum up, there are two different types of semantics at different levels needs to be captured:

- process description language semantics: This kind of semantics deals with the essential operational

semantics of BPEL4WS. It depicts pre-condition, behavior and post-condition of those basic elements of BPEL4WS in a formal way. Transition rules from BPEL4WS into Petri Net upon this level falls into metal-model scope.

- model semantics: It lies at the model level and has a close relation with concrete verification contents. For instance, whether an individual or composed process is safe and non-deadlocking, whether there are unreachable activities in process definition, whether the entire composition supplies functions to different partners as desired, such as trace equivalence or no missing reply message.

The chief aim of BPEL4WS translation in this paper is to find out the metal-model translation rules, which are influenced to some extent by model semantics to be verified. More specifically, whether a BPEL process (including individual process and composed process) is logically correct and act in sequence as desired is our verification target.

Rather than considering BPEL4WS in its full complexity, this work starts from a subset of it. First, compensation and fault handlers are ignored. Second, we do not look about time concept. Thirdly, data abstraction is conducted, that is, only data type not data value communicated between partners are concerned.

Semantics of BPEL4WS in terms of Petri Net

Basic activity and link semantics

Basic activities are activities at the bottom level, which cannot be divided into sub-parts. In the specification, each basic activity (BA) has the form showed in Fig.2. Standard attribute (SA) consists of a name, a join condition and a suppressJoinFailure indicator. When a BA is the target of other BAs, join condition explains its logical relation (for example: AND-JOIN) with those source BAs. Standard elements (SE) interpret whether a BA is the source or target of an existing control link.

<BA partnerLink=’’ portType=’’ operation=’’
Variable=’’, standard_attributes>
standard- elements
/BA>

Figure 2: Basic Activity

In our translation, <Throw>, <Wait>, <Empty> and <Terminate> activities are not involved, because <Throw> and <Wait> are about error-handling and time respectively, and the last two are solely for implementation and do not represent an action. Both control flow and data flow are covered, and the former reveals the state transition of a process while the latter displays the message transmission. The unit of data token is message wrapped in variable. When a BA is independent of other links, its semantics has the simplest form as illustrated in Fig.3: activity state and data resource (variable) are both translated into place, a token in a specific place represents activity being at the specific state or certain variable being available, and operation is translated into transition. State places have prefix “Activity Name” used for distinguishing between different activities, and no prefix is attached to variables and transitions for the reason that these elements belong to the same process and do not need to be differentiated. A BA cannot be conducted until it is in the “Ready” state and obtains the “Input” variable; after performing, it turns to the “Done” state and put a token in the “Output” place. Here, place “Input” and “Output” corresponds to the input variable and output variable of a BA, for those BAs like <Receive> and <Reply>, having only a “Variable” attribute, these variables are directly translated into output places since they represent service product acquiring. As for the <Assign> activity, place “Input” and “Output” corresponds separately to its <from> and <to> element. It has to be mentioned that, the arc connecting “Input” place and transition is bidirectional, because variables in BPEL representing business object information which may be needed many times in the long running business process and should stay available after certain operation step.

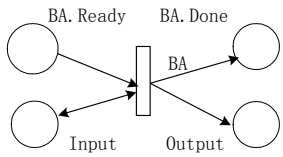

Figure 3: Semantics of BA without SE

At this time, we just capture the common semantics of basic activities in individual process specification. When considering process interaction, <Invoke>, <Receive> and <Reply> have different semantics expressed in PN, which will be discussed later. As stated, this translation adjustment is influenced by content we want to verify.

Links is used to keep synchronization between concurrent activities. Each link links a source activity (SA) to a target activity (TA). Transition condition is evaluated after SA’s termination. Till all links to a TA is evaluated and the join condition (JC) becomes true, the TA can be triggered. When it comes to the death path elimination (DPE) situation (with suppressJoinFailure attribute set to “yes”) where all links is typed false value, the TA should be skipped. If a BA does have some connection with other links, it takes on a different look from that of Fig.3. Fig.4, 5 and 6 addresses the semantics of a source BA and a target BA respectively: in Fig.4, a SA is “Done” after it output a message to another service and put a token in the “TC” (Transition Condition) place to trigger one of the TCEvaluation (TCEva for short) transitions; if the transition condition of the SA is put to default value, Fig.4 can be simplified into Fig.5, where transition condition evaluation step is omitted and no token will be put into the “TCFalse” place. Fig.6 illustrates the semantics of a target activity with default join condition and death path elimination.

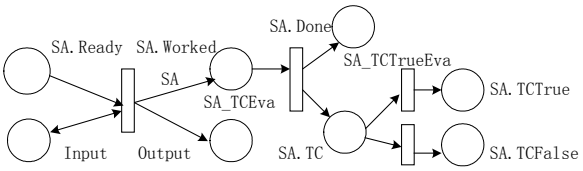

Figure 4: Semantics of SA

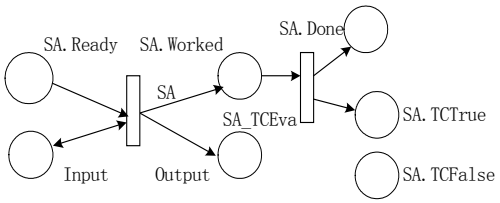

Figure 5: Semantics of SA with default TC

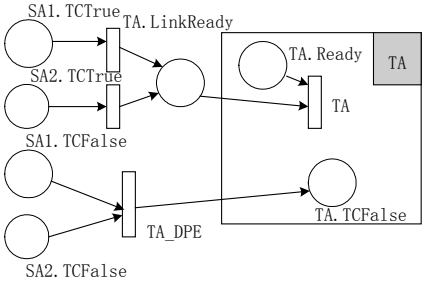

Figure 6: Semantics of TA with default JC and DPE

Structured activity semantics

In BPEL4WS, structured activities prescribe the order in which a collection of activities take place. It is very similar to the "route" node in traditional business process modeling languages.

<Sequence>: A sequence activity contains one or more activities that are performed sequentially, as Fig.7 shows, when a SA is ready to go, the first child activity is able to perform at once, after the first child is done, "Next" transition will enable the next child to perform, finally, after the last child is completed, the whole SA will be in the "Done" state. For simplicity, only two child BAs are involved in a <sequence> activity.

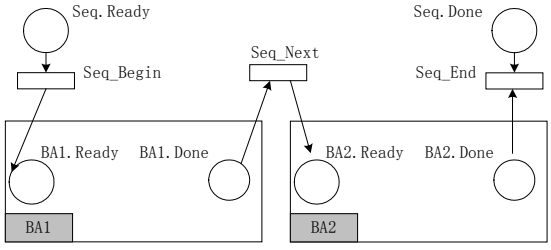

Figure 7: Semantics of <Sequence>

Due to limited space, other structured activities including <Switch>, <While> and <Flow> are skipped here.

Communication semantics

In this study, communication activities (CA) refer to those special activities interacting with other services, including <Invoke>, <Receive>, <Reply> and <Pick>. Simply put, service A and service B are interacting with each other, for acquiring some proper function provided by B, A send a message to invoke B, after receives the message, B triggers a <Reply> activity to responds to A, then A obtains the output through a <Receive> activity. Thus two pairs exist: <Invoke>-<Receive> and <Reply>-<Receive>. Without considering time concept, <Pick> has the same semantics with <Receive>.

Aiming at verifying the composed service, CA's semantics should be expressed in a way that activity execution path can be recognized and services combined easily. Here a specific naming approach is adopted: Transition is named *ProcessName _ Activity Name _ OperationName*, place is called *Process Name_Viriable Name*, and communication place, which represents call or return message transferred between partners, is labeled *PartnerRole Name.MyRole Name.Struct Name.Operation Name.Variable Name*. When an <Invoke> activity is "Ready", it transmits an input variable to its partner with an output token called *PartnerRole Name.MyRole Name.Invoke.Operation Name.Variable Name*, as Fig.8 shows. As for <Receive> and <Reply> activity, similar semantics can be expressed. When a pair of communication activities does not differ in the *MyRole.PartnerRole* content, corresponding two communication places can be combined into one place labeled with *Variable Name* (Fig.9 shows).

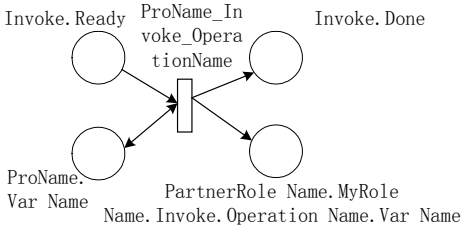

Figure 8: Semantics of <Invoke>

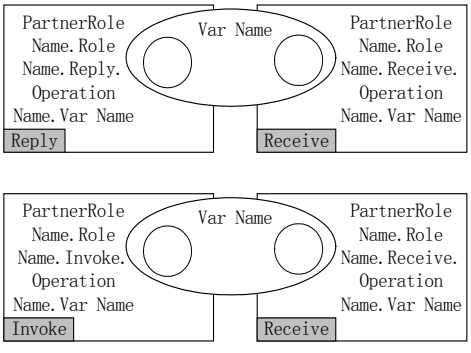

Figure 9: Communication places combining semantics

VERIFICATION PROCEDURE

With semantics specified in terms of Petri net, described before, we can use existing Petri net analysis tools or

develop our own tools to verify interested semantics of the original business process. Petri net is described in PNML format. The Petri Net Markup Language (PNML) is a newly proposal of an XML-based interchange format for Petri nets [Jonathan et al (12)]; since it gets increasingly accepted and is gaining support from many Petri net tools, it makes for the automation of verification.

First, business process specification files, including BPEL files and WSDL files, are input into Transformation Engine, which is in charge of execution of meta-model transformation rules, and are transformed to Petri net model described in output PNML files. A rule expresses how the operational semantics of a BPEL element is translated into corresponding Petri net structure. These rules are defined in XSLT files. XSLT, as the pattern based language to transform a XML file in one form to another form, is rather proper for the transformation from BPEL to PNML, which is introduced by (8) for transformation from XRL to PNML. Transformation Engine is actually an XSLT processor, such as XT, LotusXSL, SAXON, JAXP and etc; its function comprises not only the basic transformation but also later refinement operations, for instance, communication places combining.

Then, PNML files are loaded into Petri net engine for simulation and verification. Petri Net Kernel (PN Kernel) is chosen as the Petri net engine. It provides an infrastructure for bringing ideas for analyzing, for simulating, or for verifying Petri Nets into action [(13)]. It can be seen as a Document Object Model for PNML [(12)], and enables easy plugging of new user-defined semantics verification tools.

CONCLUSIONS

In this paper, we proposed a formal verification method of web service composition based on Petri net. This method is strongly influenced by our verification content, which focuses on process logic and execution trace of business processes in BPEL4WS specification, as is obvious in those meta-model operational semantics in terms of Petri net for BPEL, such as special naming strategy of transitions and places. Communication semantics is used for tackle the verification of composite process. Transformation rules from BPEL to Petri net are described in XSLT language, and target model is expressed in PNML form and analyzed in PN Kernel.

As in an early stage, we abstract from complete BPEL4WS. With more verification needs emerging, there would be adjustment of meta-model transformation rules. However, this can be easily achieved by enriching inner state transformations of each BPEL activity ((14) will be a good reference.), and also, verification method and process presented in this paper can still work.

ACKNOWLEDGEMENT:

This work is funded by Grant 2003AA4Z2020 from the National High-tech. R&D Program for CIMS of China and by Grant 60274046 from the National Natural Science Foundation of China.

REFERENCES

1. Tony A., Francisco C., Hitesh D., et al., 2003, “Business Process Execution Language for Web Services”, Version 1.1, Technical report, BEA, IBM, Microsoft, et al.

2. Ankolenkar A., et al., 2002, “DAML services”, http:/www.daml.org/services.

3. Muhammad F. K., 2003, “A Classification Framework for Approaches and Methodologies to make Web Services Compositions Reliable”, http://www.cs.ucl.ac.uk/staff/g.piccinelli/ eoows/documents/slides-kaleem.pdf.

4. Shin N., 2002, “Verification of Web service flows with model-checking techniques”, CW’02, 378-386.

5. Rachid H., Roualem, B., 2003, “A Petri Net-based Model for Web Service Composition”, ADC2003, 17.

6. Srini N., Sheila M., 2003, “Analysis and Simulation of Web Services”. Computer Networks, 42(5), 675-693.

7. Thomas P., Thomas M., Ghinea G., 2003, “Modeling of Web Services Flow”, CEC03, 391-398.

8. Aalst W.M.P., Verbeek H.M.W., Kumar, A.., 2001, “XRL/Woflan: Verification of an XML/Petri-net-based language for inter-organizational workflows”, CIST2001, 30-45.

9. Howard F., Sebastian U., Jeff M., and Jeff K., 2003, “Model-based verification of Web service compositions”, CASE2003, 152-161.

10. Howard F., Sebastian U., Jeff M., and Jeff K., 2004, “Compatibility Verification for Web Service Choreography”, http://www.doc.ic.ac.uk/~hf1/phd/papers/compatibilityforchoreography-icws2004.pdf.

11. Aalst W.M.P., Dumas M., Hofstede A.H.M., 2003, “Web Service Composition Languages: Old Wine in New Bottles?”, EROMICRO’03, 298-307.

12. Jonathan B., et al., 2003, “The Petri Net Markup Language: Concepts, Technology, and Tools”, ICATPN 2003.

13. Petri Net Kernel Team, 2002, Petri Net Kernel. http://www.informatik.hu-berlin.de/top/pnk/.

14. Karsten S. and Christian S., 2004, “A Petri net semantic for BPEL4WS-validation and application”, AWPN 2004.