Modelling of Ontology-based Service Compositions using Petri Net

Yunni Xia[1], Xiang Zhang[1], Xin Luo[1], Qingsheng Zhu[1]
[1]*Department School of Computers, Chongqing University, Chongqing 400030, P.R. China, phone: 86-23-65008071*
xiayunni@yahoo.com.cn

***Abstract*—OWL-S, one of the most significant Semantic web service ontologies , provides Web Service providers with a core ontological framework and guidelines for describing the properties and capabilities of their web Services in unambiguous, computer interpretable form. In this work we present a translation-based approach for modelling the semantic Web services described in the OWL-S language. This approach employs Petri net as the fundamental means of modelling and defines a set of translation rules to map OWL-S elements into equivalent Petri net representations. They capture the main aspect of service invocations and the control flow of the service model. A case study based on a real-world OWL-S example is also conducted to examine the effectiveness of the translation-based model.**

***Index Terms*—Petri nets, ontology, OWL-S, effectiveness.**

I. INTRODUCTION

Web Services are interfaces that describe a collection of operations that are network-accessible through standardized protocols, and whose features are described using a standard XML-based language. Web services in the Semantic Web are described through ontologies, which represent formally the service features by using a semantic mark-up language that follows a logical paradigm.

One of the most used ontologies to specify semantic Web service compositions is OWL-S (Ontology Web Language for Services) [1]. OWL-S is a computer-interpretable semantic mark-up language, where, for the first time, ontology-based descriptions of service functionality and of interaction service behaviour coexist. Recently, various research contributions have been proposed in formalization and functional verification of OWL-S. For example, researches based on Petri nets [2], [3], process algebra [4] and other formal models [5] are introduced to capture the behavioral patterns of the service ontologies and verify formal and functional properties (e.g., deadlockfreeness, boundedness, interface compatibility, and liveness). However, there are still many limitations in the abovementioned models, e.g., the incomplete modelling of the process model, the absence of modelling service grounding, and the absence of modelling timeout in the atomic process, .etc.

The main objective of this research is the development of a feature-completed formal model for describing OWL-S processes, thus paving the way for analyzing the ontology-based services by exploiting the formalization and reasoning power of Petri nets. This approach is based on a set of translation rules to map OWL-S elements into equivalent Petri net representations. They capture the main aspect of service invocations, the behavioural patterns for atomic/composite processes, and the control flow of the service model.

To illustrate the effectiveness of the framework, we conduct a case study based on a real-world OWL-S application, where the OWL-S document is translated into an equivalent Petri net representation.

II. OWL-S

OWL-S is defined as a W3C standard to provide a computer-interpretable description of the services, service access and service composition using OWL ontologies. Building upon SOAP (Simple Object Access Protocol) and WSDL (Web Service Definition Language), the OWL-S services can be dynamically executed on the Web.

OWL-S models the upper ontology for services from three perspectives: *Service Profile, Service Grounding, and Service Model.* The Service Profile provides a high-level description of the service entity and its provider for advertising, requesting and matchmaking. The Service Grounding defines the mapping from abstract representation to concrete specification, which specifies the details of how to access the service such as message formats, serialization, transport, and addressing. The Service Model serves as the control flow model of service interactions and the process template of service compositions. The Service Model is modelled as a workflow of processes, including atomic, simple and composite processes. Each composite process holds a control construct, which can be implemented using the sequence, parallelism, choice, and repetition patterns. The construct can contain each other constructs recursively.

III. MODEL TRANSLATION OF OWL-S

In this section, we introduce the translation rules for OWL-S. Because the syntax of OWL-S is too vast, we restrict the translation into a subset (OWL-S elements such

Manuscript received March 19, 2012; accepted May 22, 2012.
This work was supported partly by the Natural Science Foundation of China (No.61103036).

as input-binding, output-binding, data manipulation, boolean condition evaluation, preconditions and results are abstracted away and omitted), which describes the control flows of the activity executions and message exchanges. This subset is mainly specified by the Service grounding and Service model specifications. It captures the control flow evolution of the OWL-S workflow, the compositional patterns by which the processes are organized, and the invocation of external services (from abstract processes to the concrete services specified by the WSDL documents).

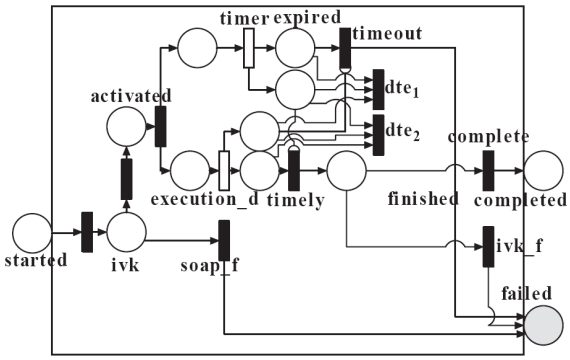

Fig. 1. Petri net model of the atomic process.

The translation starts with the atomic process. The atomic process in OWL-S is a description of a service that expects one (possibly complex) message and returns one (possibly complex) message in response. It is invoked through the <*perform*> construct. It corresponds to an action that a service can perform in a single interaction, and it can be executed in a single step by sending and receiving appropriate messages. The main operation of the atomic process is to contact and invoke the partner services through the grounding mechanism. The grounding is a mapping from the abstract specifications to actual executable services (specified by WSDL documents).

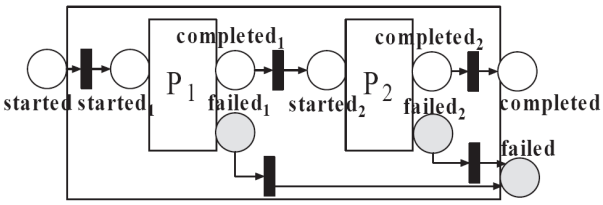

Fig. 2. Petri net model of the <*sequence*> process.

According to the discussion above, an atomic process in OWL-S can be translated into the Petri net model given in Fig. 1. In Fig. 1, the *started* and *completed* places indicate the initial and completed states of the process, respectively. The *execution_d* timed transition denotes the duration needed for the invoked service to complete execution. *timer* denotes the timeout threshold. Because the external services are invoked through SOAP messages and the SOAP connections are subject to failure (caused by message loss, for instance), the *soap_f* transition is used to capture the SOAP failure and it directly marks *failed* (in gray) to indicate the unsuccessful invocation. If no SOAP failure occurs and the service execution duration exceeds the timeout threshold, a timeout event is triggered and the *failed* place is marked. Otherwise, the *timeout* transition is prevented and the *timely* place is marked. In this case, the execution of the invoked external service can either be faulty (by firing the *ivk_f* transition and marking the failed place) or successful (by firing the complete transition and marking the *completed* place). Note that the dte_1 and dte_2 (*dte* stands for dead-token-elimination) transition ensure that no dead token exists when the atomic process normally completes or fails.

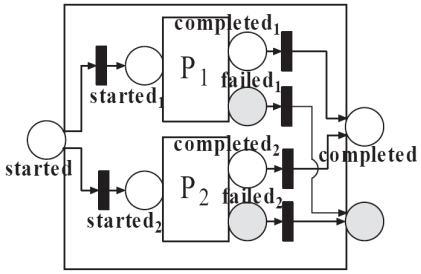

Fig. 3. Petri net model of the <*choice*> process.

The transition of composite processes is also given. All composite processes are composed of atomic processes. The <*composedOf*> property in OWL-S describes the control flow and the data flow of sub-processes within a composite process, yielding constraints on the ordering and conditional execution of these sub-processes. The composite processes can be implemented by the <*sequence*>, <*split*>, <*split-join*>, <*choice*>, <*any-order*>, <*if-then-else*>, <*repeat-while*>, and <*repeat-until*> patterns.

We start with the <*sequence*> process. In this process, a list of control constructs is executed sequentially. The equivalent Petri net model of this process is given in Fig. 2. For simplicity, we assume that there exist only two sub-processes, namely P_1 and P_2. The failure mode of <*sequence*> is simply implemented by propagating the inner failures of $P_{1,2}$ to the level of <*sequence*> itself through the immediate transitions from $failed_{1,2}$ to *failed*.

The <*choice*> process stipulates that a single one from a given bag of sub-processes (specified by the <*components*> property) is executed. As shown in Fig. 3, the choice construct includes two selective branches, P_1 and P_2. It is organized by an XOR selective construct. Selecting and completing either branch would allow the <*choice*> process to finish. The failure mode of <*choice*> is implemented in a similar way to that of the <*sequence*> process.

The <*split*> process stipulates that branches are executed in parallel. It completes as soon as all of its branches have been scheduled for execution and dose not wait for the completion of those branches. The translation of the <*split*> process is given in Fig. 4.

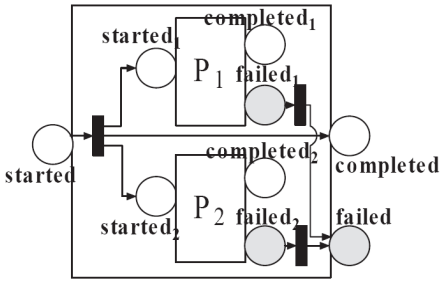

Fig. 4. Petri net model of the <*split*> process.

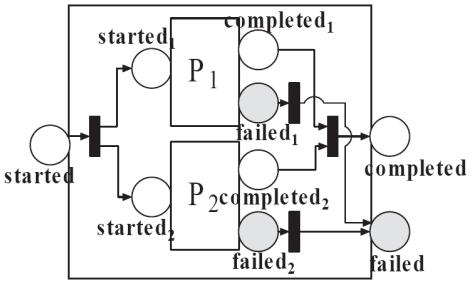

Fig. 5. Petri net model of the <*split-join*> process.

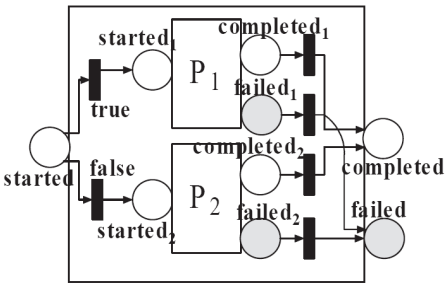

Fig. 6. Petri net model of the <*if-then-else*> process.

The <*split-joint*> process also supports concurrent execution but is intrinsically different from the <*split*> process. It consists of the concurrent execution of a bunch of processes, following a barrier synchronization style. That is, it completes when all of its sub-processes have completed. The equivalent Petri net model of the <*split-join*> process is given in Fig. 5.

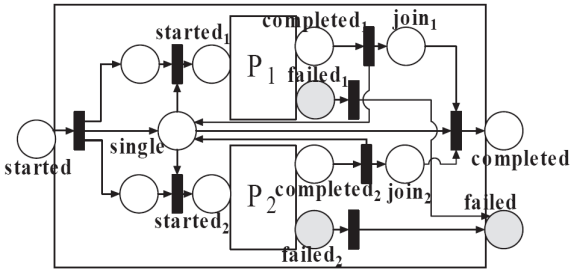

Fig. 7. Petri net model of the <*any-order*> process.

The <*if-then-else*> process is a control construct associated with a Boolean decision. If the condition is satisfied, the true branch (i.e., the <*then*> branch) is selected and executed, otherwise the false branch (i.e., the <*else*> branch). The <*if-then-else*> process is accomplished when its selected branch is completed. The equivalent Petri net model is given in Fig. 6, where the *true/false* immediate transitions denote the true/false evaluation of the Boolean condition.

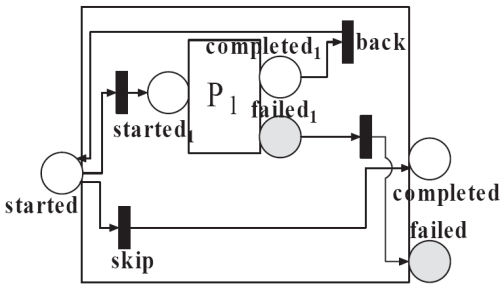

Fig. 8. Petri net model of the <*repeat-while*> process.

The <*any-order*> process allows the sub-processes to be executed in some unspecified order but not concurrently. Execution and completion of all branches are required. As shown in Fig. 7, the execution of branches in an <*any-order*> process cannot overlap and all branches must be executed before the <*any-order*> process completes. The single place and the bidirectional arcs from single guarantee that P_1 and P_2 are not executed concurrently.

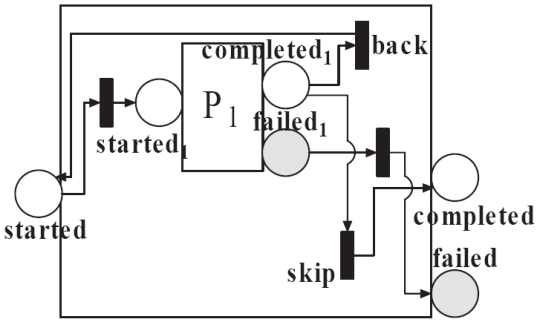

Fig. 9. Petri net model of the <*repeat-until*> process.

Both the <*repeat-while*> and <*repeat-until*> processes support iterative execution. They keep iterating until a condition becomes false or true. <*repeat-while*> tests for the loop condition, loops if the condition is true, and otherwise executes the nested process. <*repeat-until*> executes the nested process, tests for the condition, exits if it is true, and otherwise loops. Thus, <*repeat-while*> may never execute its nested process, whereas <*repeat-until*> always executes the nested process at least once. Fig. 8 and Fig. 9 show the Petri net models of the two processes. In these figures, the *back* immediate transition leads the control flow *back* to the beginning, and the *skip* immediate transition leads the control flow out

IV. A CASE STUDY

In this section, we conduct a case study to illustrate the effectiveness of the translation introduced above. The case study is based on the frequently used CongoProcess sample given in [6]. The *FullCongoBuy* process is the uppermost composite process of the sample. It is organized by an <*sequence*> process and composed of an atomic process, *LocateBook*, and a composite process, *OrderManagement*. The *OrderManagement* process implements an <*any-order*> process and includes two composite processes, namely *CongoBuyBook* and *UserInfoRetrieval*. The *UserInfoRetrieval* process implements a sequential process and includes two atomic processes, namely *LoadUserProfile* and *ValidateUserEmail*. The *CongoBuyBook* process also implements a sequential process and includes a composite process, *BuySequence*. The *BuySequence* process implements a sequential process and includes an atomic process, *PutInCart*, and a composite process, *SignInAndSpecify*. The *SignInAndSpecify* process implements a <*split-join*> process and includes two atomic processes, namely *SpecifyPaymentMethod* and *ShipmentManagement*. Based on the translation rules given in the previous section, the sample can be translated to the Petri net model given in Fig. 10.

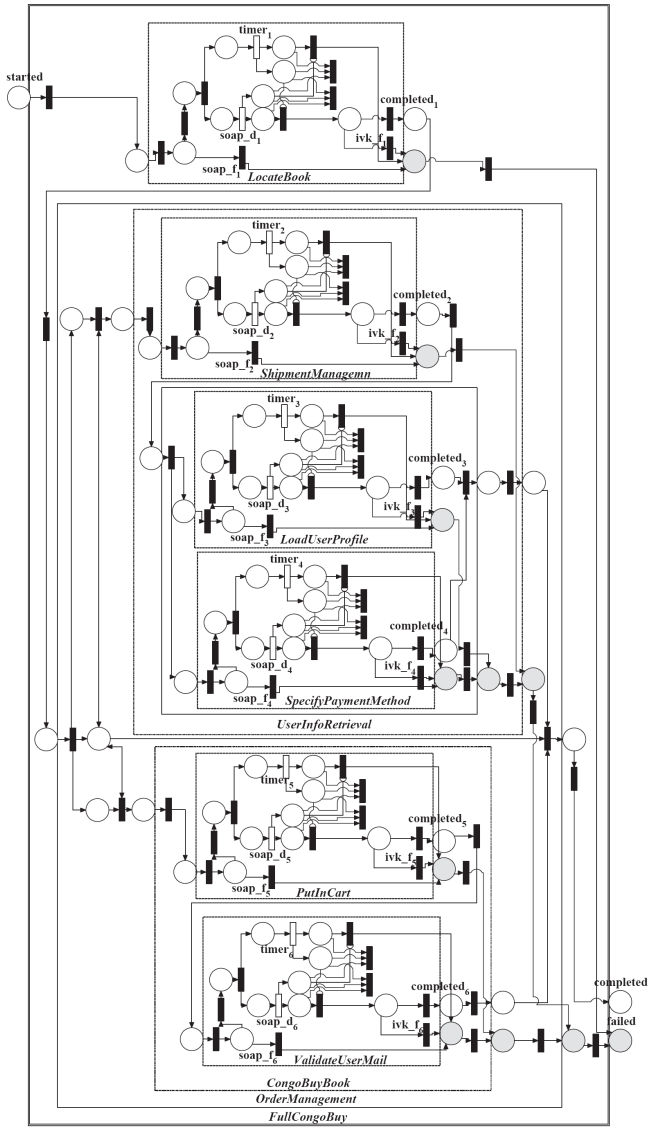

Fig. 10. Petri net model of the *FulCongoBuyl* process.

V. CONCLUSIONS

In this manuscript, we present a translation-based model for ontology-based service compositions built on OWL-S. In this model, the OWL-S elements are translated into the equivalent Petri net representations. This model captures the main aspects of service invocation and control flow evolutions. A case study based on a real-world OWL-S sample is also conducted to examine the effectiveness of the model.

REFERENCES

[1] J. Cardoso, A. Sheth, "Introduction to Semantic Web Services and Web Process Composition", *Lecture Notes in Computer Science,* vol. 3387, pp. 1–13, 2005. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30581-1_1

[2] G. Dai, X. Bai, C. Zhao, "A Framework for Time Consistency Verification for Web Processes Based on Annotated OWL-S", in *Proc. of the GCC -IEEE Computer Society,* 2007, pp. 346–353.

[3] A. Brogi, S. Corfini, S. Iardella, "From OWL-S Descriptions to Petri Nets", in *Proc. ICSOC -IEEE Computer Society,* 2007, pp. 427–438.

[4] B. Norton, S. Foster, A. Hughes, "A Compositional Operational Semantics for OWL-S", in *Proc. EPEW/WS-FM -IEEE Computer Society,* 2005, pp. 303–317.

[5] S. Narayanan, K. Sievers, S. J. Maiorano, "OCCAM: Ontology-Based Computational Contextual Analysis and Modeling", in *Proc. of the CONTEXT -IEEE Computer Society,* 2007, pp. 356–368.

[6] *The CongoProcess sample.* [Online]. Available: http://www.daml.org/services/owls/1.1/CongoProcess.owl