Modeling Web Services Composition with Transaction Extension for Performance Evaluation

Yanxiang He[1], Liang Zhao[1], Zhao Wu[2], Fei Li[1]
[1]*School of Computer, Wuhan University, Wuhan, 430079, P. R. China*
[2]*Department of Computer Science and Technology, Xiangfan University, Hubei, 441053, P.R. China*
Email:zhl_mars@hotmail.com

Abstract

Web Services can be composed to build domain-specific application and solution. The standards of several Web services composition (WSC) are proposed, for example, WS-BPEL and WS-CDL. Moreover, there is a great demand for the performance optimization of WSC recently. However, WS-BPEL lacks formal semantics, so it is very difficult to verify WSC and evaluate the performance of WSC. Therefore, considering such situation and Long-Running Transaction (LRT) in WS-BPEL, in this paper, we adopt General Stochastic High-Level Petri Net (GSHLPN) as basic formal description tool of WSC based on WS-BPEL and model the context of LRT. Our approach can provide a more real environment for evaluating and optimizing the performance of WSC based on WS-BPEL.

1. Introduction

Service-oriented computing (SOC) is an emerging paradigm that is changing the way systems are designed, architected, deployed, and used. SOC decomposes computation into a set of loosely-coupled, abstract services, and emphasizes document-centric interactions through the exchange of messages. Web Services and WSC play the key role in SOC field. Some standards of WSC are proposed recently, for example, WS-BPEL [1] and WS-CDL. However, WS-BPEL lacks formal semantic, so the validity of WSC can not be verified and analyzed, including conformance check, deadlock, unreachable activities in WS-BPEL. The existing methods used for modeling WSC include Petri net [2] [3], π calculus, graph grammar, process algebra, state diagram, activity diagram etc.

Unfortunately, above formal approaches lack the description for time variable required by performance evaluation and can not simulate for LRT because of the limitation of itself. Therefore, there are few researches on performance evaluation of WSC based on WS-BPEL. Consequently, we propose a model to support the simulation in such environment through GSHLPN [4] [5] and then evaluate its performance.

The rest of this paper is organized as follows. Related work is introduced in Section 2. State pattern of composite web service is presented in Section 3. Section 4 discusses the description of interrupt event in GSPHLN for WS-BPEL. Section 5 illustrates business process model and the computing of state. Section 6 draws conclusions and future work。

2. Related work

Several approaches [6] [7] have been proposed in literatures that deal with transformation from WS-BPEL to Petri Net. In [6] and [7], they all propose Petri net semantics for BPEL and want to formally analyze and verify BPEL processes. Owe to the related works, we research on WS-BPEL specification, and think that the approach based on pattern is better for describing WS-BPEL construct, because it can hierarchically describe Petri Net so as to reduce the complexity of graphic description, we prefer to adopt the approach that is similar to [7].

However, there are several differences between [7] and our method. Firstly, [7] addresses checking the consistence of business process based on WS-BPEL, thus it uses Petri Net as the formal analysis tool. But Petri net lacks the simulation of time and interrupt. Secondly, in contrast to [7], we use the state of composite web service as the interface of activity in WS-BPEL. Considering LRT's feature, we can give a simple description of LRT by means of GSHLPN and model defined by section 5. In term of several researches [9] [3] on performance evaluation of WSC, they all do not consider LRT's impact on performance. Therefore, this paper aims at the limitations of their

978-0-7695-3473-2/08 $25.00 © 2008 IEEE
DOI 10.1109/APSCC.2008.104

researches and proposes a model to mostly simulate the behavior of LRT in WS-BPEL.

3. State pattern of composite Web Service

In [8], a web service can be in one of the following states: NotInstantiated, Ready, Running, Suspended, or Completed. However, in WS-BPEL specification, except for structured relations, synchronization dependence relation also plays a key role in the relation between activities. For performance analysis and optimization, time is a very important variable, so we must consider such relations and can not ignore the dead-path-elimination problem [1]. WS-BPEL provides the capability of FCT-handler, and then in the performance analysis, the simulation for the exception behaviors is necessary. As a result, we propose that web service own the following states in WS-BPEL: **NotInstantiated**, **Skipped**, **Ready**, **Running**, **Failed**, **Terminated** and **Completed**.

- **NotInstantiated**: Web service is not instantiated;
- **Skipped**: It is the need of dead-path-elimination. Web service will be skipped though it can have been instantiated;
- **Ready**: web service is initializing;
- **Running**: Web service has finished initialization and is running;
- **Failed**: Web service has the fault during its running;
- **Terminated**: Web service completes unsuccessfully or is terminated by *context*;
- **Completed**: Web service completes successfully.

In WS-BPEL, we regard that a composite web service is equivalent to an activity, and consider *place* represents the state of composite web service or exchanged messages, while *transition* represents the process of web service in Petri net. So for a composite web service we may use seven places to represent the interfaces of composite web service state. But because NotInstantiated and running states may be judged through others states, they are optional. Generally we do not use the places to describe them. Then a composite web service may be depicted in Figure 1.

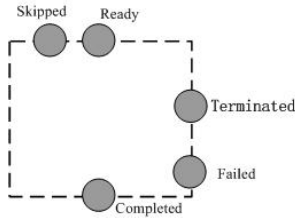

Figure 1: state pattern of composite Web service

It is very important for Business processes to do with fault, canceling operation and termination event in WS-BPEL. When FCT events happen, system will catch these events and interrupt the running process and execute the corresponding operation. So we need to simulate such handlers in order to obtain analysis results after failures happen. Figure 2 represents it. In this figure, we add a place marked as “Catching” in order to catch the event that may be generated by system or manual control. In this paper, we use dashed circle to represent such place.

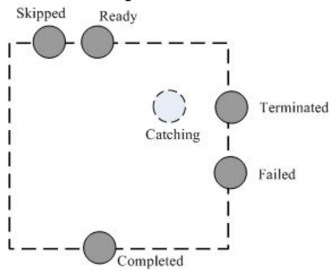

Figure 2: “Catching” place in composite web service

4. Describing interrupt event through GSHLPN

In Petri net, a transition reflects an event in the real system and the fire of transition reflects the state change in system. There are two reasons for state change: the certain logic conditions of verification and the completion of certain activities. For the later, it can simulate activity. Compared to common Petri net, transition is divided into two different classes: *timed* transition and *immediate* transition in GSHLPN. *Immediate* transitions fires within zero time once they are enabled. *Timed* transitions fire after a random, exponentially distributed enabling time. The priority of *immediate* transitions is higher than *timed* transition, when several transitions may be simultaneously enabled, if the set of enabling transition, H, comprises *timed* transition and *immediate* transition, *immediate* transition may be enabling and but *timed* transition can not be enabling.. In Figure 3, the token in place P_1 starts the activity simulated by T_1, but if P_2 obtains a token before T_1 will be fired, because the priority of T_2 is higher than T_1, then T_1 will become unenforceable while T_2 will become enforceable and be fired. So the fire of T_2 can interrupt the fire of T_1.

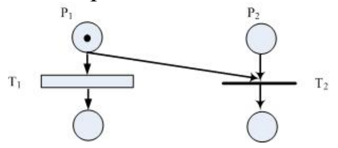

Figure 3: Use immediate transition to interrupt activity

For example, <receive> activity in WS-BPEL may be described as following figure.

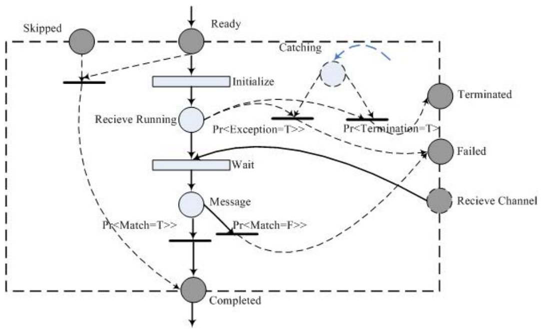

Figure 4: Receive Activity

White solid circle represents inner place in a pattern, and ***grey solid circle*** represents the interface of a pattern, while ***grey dashed circle*** represents this place is shared by other activities, and ***white dashed circle*** represents that when this place owns a token, it needs the trigger of outer or WSC context. The dashed arc on the "Catching" place means that this place owns the token by the trigger of the outer condition or WSC Context, in every figure two *immediate* transition are connected with "Catching" place and the place under running state also has an arc with this two places, and Pr<Exception = T> and Pr<Termination = T> describe the logical condition verification. It can be see that only failure or terminated event from the trigger of context has happened, for example, the failure in *Channel*, the activity in running state will be interrupted or terminated and the state of activity also is modified into terminated or failed.

5. Simulation model of WSC for LRT

5.1. Business process model

A business process may be regarded as a composition of web services, which are coordinated to achieve a certain business goal. With the purpose of performance evaluation, we adopt GSHLPN as basic formal description tool of WSC based on WS-BPEL and model the context of LRT. Therefore, business process may be defined as the following tuple:

$$BP = (PN, CS, CR, I, O, CT)$$

Where:

- $PN=(P,T;F,A,V,D,X,W,M_0,\lambda)$ represents business process is described through GSHLPN. Its detail description may see [4];
- *CS* is a set of composite web services;
- *CR* is a set of relations among web services will be introduced in the section 5.2;
- *I* is a set of input port of *CS*;
- *O* is a set of output port of *CS*;
- *CT* is a *context* of web service composition and will be introduced in the section 5.3.

Accordingly, we apply *BP* model into the performance evaluation for business process under the following scenario:

- Owing to the works in Section 4, interrupt behavior, which is triggered by the transaction in business process, can be manually controlled. In consequence, the behavior of exceptional canceling transaction can be simulated under the probability function;
- The impact of FCT handler can be simply simulated because of the description of context of WSC.

5.2. Modeling a composite Web service

In a business process, multiple Web services may be required to collaborate with each other to form a composite Web service, while multiple composite Web services can be constructed into business process. Thus, a composite web service is a tuple:

$$CWS = (PN, CS, I, O, PRE, POST, CR, state)$$

Where:

- *PN*, *CS*, *CR*, I and *O* are same above description.
- *PRE* is a set of pre-condition that CWS can be executed;
- *POST* is a set of post-condition that impacts on the other CWS that have the synchronization dependencies on it.
- *state* is the state of web service in section 3.

The paper [8] proposes the set of services can be defined by s the syntax and informal semantics of the service algebra operators. In here, we mainly care about structured (including sequence, choice, loop and parallelism) and synchronization dependency relation. Hence a composite web service and CR may be defined by as fellow:

$$S := \varepsilon \mid X \mid S \to S \mid S + S \mid S \parallel S \mid \nabla_c S \mid \overline{\nabla}^n_c S \mid S \mapsto S$$

Where:

- ε is empty composite web service, for example, <empty> activity in WS-BPEL;
- X is an atomic web service, for example, a web service wrapped by WSDL in WS-BPEL;
- $\to$ is a *sequence* binary operator, and $S_1 \to S_2$ represents the web service S_2 follows immediately the web service S_1;
- + is a choice binary operator, and $S_1 + S_2$ represents a composite service that behaves as either service S_1 or service S_2 [8];
- $\parallel$ is a parallel binary operator, and $S_1 \parallel S_2$ represents a composite service that performs the

services S_1 and S_2 independently from each other [8];

- ∇_c is an iteration single operator, and $\nabla_c S$ represents the web service S performs repeatedly until condition c is satisfied;
- $\overline{\nabla}_c^n$ is an parallel iteration single operator, $\overline{\nabla}_c^n S$ represents a composite service create n copies and every copy will performs in parallel until condition c is satisfied;
- $\mapsto$ is a synchronization dependency operator;

Consequently, (*CS, CR*) construct a mapping space, $CS \times (I \cup O) \times CS \subseteq CR$ is *structured*-relation-set. In-addition,-we consider the relations among *PRE*, *POST* and *CR* as following case:

- $S_1 \mapsto S_m, S_2 \mapsto S_m, \cdots, S_n \mapsto S_m \equiv (S_1 | S_2 | \cdots | S_n) \mapsto S_m$ (Set $\{S_1, S_2, \cdots, S_n, S_m\}$ forms an "*And-Join*" structure.)
  $\Rightarrow$ S_m*.PRE* is partially decided by *S₁.POST*, *S₂.POST*,···, *Sₙ.POST*;
- $S_m \mapsto S_1, S_m \mapsto S_2, \cdots, S_m \mapsto S_n \equiv S_m \mapsto (S_1 | S_2 | \cdots | S_n)$ Set $\{S_1, S_2, \cdots, S_n, S_m\}$ forms an "*And-Fork*" structure.
  $\Rightarrow$ *S₁.PRE*, *S₂.PRE*,···, *Sₙ.PRE*; are partially decided by S_m*.POST* .

Meanwhile, we extend the above relation of WSC because of the convenience to describe the cluster of composite web services.

- $\xrightarrow{l}$: $S_1 \xrightarrow{l} S_2$ represents S_1 and S_2 only lie in the same *sequence block*.
  $(S_1 \rightarrow S_2, S_2 \rightarrow S_3, \cdots, S_{n-1} \rightarrow S_n)$
  $\wedge (\forall j \neq i+1 \;\; \neg\exists((S_i, S_j) \in (CR \setminus \{\mapsto\}) \wedge (S_n, S_1) \in CR)$
  $(i = 1, \cdots, n-1)$
  $\Rightarrow \{S_i\}$ construct and only lie in *the same sequence block*.
  $(i = 1, \cdots, n)$

- $\xrightarrow{p}$: $S_1 \xrightarrow{p} S_2$ represents S_1 and S_2 construct only lie in the same *parallel block*.
  $S_1 \| S_2, S_2 \| S_3, \cdots, S_{n-1} \| S_n$
  $\wedge \exists r = (S_i, S_j) \in CR \quad r = \| \; or \mapsto \quad (i = 1, \cdots, n, j \in N)$
  $\Rightarrow \{S_i\}$ construct and only lie in *the same parallel block*.
  $(i = 1, \cdots, n)$

So for composite relation like $\xrightarrow{l}$ and $\xrightarrow{p}$, we commonly regard the set of web service formed by such relation as the whole when need tracing the execution path and obtain *reverse path* in case *Context* need to send compensation event to its child along with reverse path. Generally, the approach of construction of our simulation model is *top-down*, and thus this model is hierarchical.

5.3 Modeling a context of WSC

"Context" (CT) simply describes the behavior of Long-Running Transactions (LRT's) [1], which use compensation to handle failures, potentially aggregate smaller ACID transactions. In contrast to rollback in ACID transactions, compensation restores the original state, or an equivalent, and is business-specific. Compensation may be defined as the most logical change applied to the resource to maintain data consistency and integrity. Then when compensating, the effects of some logic related web services need to be negated, these logic associated web services form the structure of *SCOPE*, and *SCOPE* may be nested. The state of *SCOPE* is decided by all immediately enclosing scopes and web services, and will trigger the immediately enclosing web service which state is running to interrupt their running when *SCOPE* catches the exception and termination event. Hence, *SCOPE* is a tuple:

$$SCOPE = (SCS, CS, IP, ST, i, o, state)$$

Where:

- *SCS* is a set of scopes;
- *ST* is set of state immediately of enclosing composite web services and scopes;
- *IP* is a set of the place of *interrupt* triggers ("Catching" place) in all composite web services;
- *i* is the first scope or composite web service;
- *o* is the last scope or composite web service.

In BP model, *Context* of web service composition (*CT*) is actually a whole scope. It may be regarded as directory service, and can describe and control the transaction behaviors of nested scopes and web services, for example, compensation handler and fault handler.

5.4 State computing

5.4.1 State of composite web service

Composite web service *S* is composed of *S₁* (and *S₂*) by the relations defined by section 5.2.

***Sequence* composition:**

Ready:

$S_1.state = Ready \Rightarrow S.state = Ready$;

Running:

$S_1.state = Running \Rightarrow S.state = Runing$;

Failed:

$S_1.state = Failed \vee S_2.state = Failed \Rightarrow S.state = Failed$;

Terminated:

$S_1.state = Terminated \vee S_2.state = Terminated$

$\Rightarrow S.state = Terminated;$

Completed:

$S_2.state = Completed \Rightarrow S.state = Completed;$

Skipped:

$S.state = Skipped$

$\Rightarrow S_1.state = Skipped \text{ and } S_2.state = Skipped$

NotInstantiated:

$S_1.state = NotInstantiated \Rightarrow S.state = NotInstantiated;$

Choice composition:

Ready:

$S_1.state = Ready \vee S_2.state = Ready \Rightarrow S.state = Ready;$

Running:

$S_1.state = Running \vee S_2.state = Running$

$\Rightarrow S.state = Running;$

***Failed*:**

$S_1.state = Failed \vee S_2.state = Failed \Rightarrow S.state = Failed;$

Terminated:

$S_1.state = Terminated \vee S_2.state = Terminated$

$\Rightarrow S.state = Terminated;$

Completed:

$S_1.state = Completed \vee S_2.state = Completed$

$\Rightarrow S.state = Completed;$

Skipped:

$S.state = Skipped$

$\Rightarrow S_1.state = Skipped \text{ and } S_2.state = Skipped$

$S.state \neq Skipped \wedge S_1.state = Ready \Rightarrow S_2.state = Skipped$

$S.state \neq Skipped \wedge S_2.state = Ready \Rightarrow S_1.state = Skipped$

NotInstantiated:

$S_1.state = NotInstantiated \wedge S_2.state = NotInstantiated$

$\Rightarrow S.state = NotInstantiated;$

Parallel composition:

Ready:

$S_1.state = Ready \wedge S_2.state = Ready \Rightarrow S.state = Ready;$

Running:

$S_1.state = Running \vee S_2.state = Running$

$\Rightarrow S.state = Running;$

***Failed*:**

$S_1.state = Failed \vee S_2.state = Failed \Rightarrow S.state = Failed;$

Terminated:

$S_1.state = Terminated \wedge S_2.state = Terminated$

$\Rightarrow S.state = Terminated;$

Completed:

$S_1.state = Completed \wedge S_2.state = Completed$

$\Rightarrow S.state = Completed;$

Skipped:

$S.state = Skipped$

$\Rightarrow S_1.state = Skipped \text{ and } S_2.state = Skipped$

NotInstantiated:

$S_1.state = NotInstantiated \wedge S_2.state = NotInstantiated$

$\Rightarrow S.state = NotInstantiated;$

Loop composition:

Ready:

$S_1.state = Ready \Rightarrow S.state = Ready;$

Running:

$S_1.state = Running \Rightarrow S.state = Running;$

***Failed*:**

$S_1.state = Failed \Rightarrow S.state = Failed;$

Terminated:

$S_1.state = Terminated \Rightarrow S.state = Terminated;$

Completed:

$condition = true \Rightarrow S.state = Completed;$

Skipped:

$S.state = Skipped \Rightarrow S_1.state = Skipped$

NotInstantiated:

$S_1.state = NotInstantiated \Rightarrow S.state = NotInstantiated;$

5.4.2 State of context

We suppose as following scenario:

- $scope[m]$: m is the times of actually entering the same scope;
- U is a set of uncompleted immediate scopes and web services in $scope[m]$
- C is a set of completed immediate scopes and web services in $scope[m]$;

Consequently, the state of a scope may be computed by the following method:

Ready:

$i.state = Ready \Rightarrow scope[m].state = Ready;$

***Running*:**

$\exists u \in U \quad u.state = Running \Rightarrow scope[m].state = Running;$

***Failed*:**

$\exists u \in U \quad u.state = Failed \Rightarrow scope[m].state = Failed;$

Terminated:

$\neg \exists u \in U$

$u.state \neq Terminated \Rightarrow scope[m].state = Terminated;$

Completed:

$|U| = 0 \Rightarrow scope[m].state = Completed$

Skipped:

$\neg \exists c \in C$

$c.state \neq Skipped \wedge |C| = |scope[m].SCS| + |scope[m].CS|$

$\Rightarrow scope[m].state = Skipped$

NotInstantiated:

$m = 0 \Rightarrow scope.state = NotInstantiated$

Construction of *U* and *C*:
Step1. $scope[m].state = Ready \;\Rightarrow$
Let $U := SS \cup SCP, C := \phi$
Step2.
$\forall u \in U$ When $u.state = Completed$ or $u.state = Skipped$
Do $U := U - \{u\}$, $C := C + \{u\}$

SCOPE will catch exception, termination and compensation events in its whole lifetime and do corresponding handlers.

Exception **event in a scope** ($scope[m]$):
$\forall u(u \in U \wedge u.state = Running)$
Step1.If u is a *scope*, *Send* (termination, u);
Step2. If u is a *web service*, $scope[m]$ searches the *interrupt place* ("Catching" place) of u and fires it with 'Terminated' notation;
$\forall v(v \in C \wedge v.state = Completed)$
Step3. *Send* (compensation, v) along with *reverse* path.
Step4. Update the state of $scope[m]$

Termination **event in a scope** ($scope[m]$):
$\forall u(u \in U \wedge u.state = Running)$
Step1. If u is a *scope*, *Send* (termination, u)
Step2. If u is a *web service*, $scope[m]$ searches the *interrupt place* ('Catching' place$\in$ IP) of u and fires it with 'Terminated' notation;
Step3. Update the state of $scope[m]$

Compensation **event in a scope** ($scope[m]$):
Step1. $\forall v(v \in C_i \wedge v.state = Completed)$
Send (compensation, v) along with reverse path.
Step2. $scope[m].state =\sim ready$ ("*~Ready*" represents scope or web service has been instantiated but not enter scope)
Step3. m := m-1;
Step4. Update the state of $scope[m]$

6. Conclusion and future research

Our goal is to establish simulation-based on performance evaluation system approximated to the environment of transaction in WS-BPEL. Because of LRT in WS-BPEL, we need to take advantage of interrupt event to emulate transaction handler. Furthermore, we consider that GSHLPN is used to formally describe web service composition consisted of the activities in WS-BPEL. Meanwhile, we also need to consider the context of transaction handler. Therefore, we take into account the above consolidated cases. We build the simulation model with the characteristics in order to evaluate the performance of WSC based on WS-BPEL in more reasonable environment. Our future work is to make use of this simulation model to implement the visual simulation-based on the performance evaluation system.

7. Acknowledgement

This research has been supported by the National High-Tech Research and Development Program (863 programs) of China under Grant No. 2007AA01Z138

8. References

[1] OASIS (2007) Web Services Business Process Execution Language (WS-BPEL version 2.0) http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf
[2] Murata T. Petri nets: properties, analysis and application. Proceedings of the IEEE, 1989, 77(4), pp. 541-580
[3] A. N. Silva, F. A. A. Lins, J. C. Santos, N. S. Rosa, N. C. Quental, and P. R. M. Maciel. Performance Evaluation of Web Service Composition Using Petri Nets. In Brazilian Symposium on Computer Networks, Curitiba, Parana,
[4] Lin Chuang. *Stochastic Petri nets and System Performance Evaluation*, Beijing, Tsinghua University Press, 2005.
[5] Marco Ajmone Marsan , Andrea Bobbio , Susanna Donatelli, Petri Nets in Performance Analysis: An Introduction, Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, September 01, 1996, pp. 211-256.
[6] Ouyang, C., Verbeek, E., Aalst, W.M.P.v.d., Breutel, S., Dumas, M., Hofstede, A.H.t.: Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-15, Queensland University of Technology, BPMcenter.org (October 2005) http://www.BPMcenter.org/reports/2005/BPM-05-15.pdf, accepted for publication by Science of Computer Programming.
[7] S. Hinz, K. Schmidt, C. Stahl, Transforming BPEL to Petri nets, in: W.M.P. van der Aalst, B. Benatallah, F. Casati, F. Curbera (Eds.), Proceedings of the International Conference on Business Process Management, BPM 2005, in: Lecture Notes in Computer Science, vol.3649, Springer-Verlag, Nancy, France, September 2005, pp. 220–235.
[8] Rachid Hamadi , Boualem Benatallah, A Petri net-based model for web service composition, Proceedings of the fourteenth Australasian database conference, , Adelaide, Australia , February 01, 2003, pp. 191-200,
[9] Shan Zhiguang, Lin Chuang, Marinescu D C, Yang Y. Modeling and performance analysis of Qos aware load balancing of Web server clusters. Computer Networks, Elsevier Science, 2002, 40(2), pp. 235-256 Brazil, May 2006.