Journal of Southeast University (English Edition) Vol. 22, No. 3, pp. 315 – 318 Sept. 2006 ISSN 1003—7985

Analysis and modeling of web services flow using π-calculus

He Tao Miao Huaikou Qian Zhongsheng

(School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

Abstract: In order to increase the effectiveness and the reliability of web services flow, the π-calculus formal method is introduced as a development language for web services flow. The π-calculus overcomes inabilities of web service flow languages in demonstrating the consistency, validating the correctness and so on. The π-calculus analysis and modeling of web services flow is presented, the dynamic actions and basic activities of WS-BPEL with π-calculus formally are described, and the mapping from π-calculus expression to WS-BPEL is built. The basic construction of web services flow with the π-calculus method after the analysis of the syntax of WS-BPEL and inter-description between WS-BPEL and π-calculus is expressed. Also discussed are the approaches to web services flow by modeling from different views, and the proposed approaches through the development and modeling of an e-commerce web service flow application are illustrated.

Key words: business process execution language (BPEL); web services; work flow; π-calculus

Web services (WS) are distributed and independent processes which communicate with each other through the exchange of messages, and the central question in WS engineering is therefore to make a number of processes work together to perform a given task[1]. But web services flow languages such as WS-BPEL, which do not offer formal analysis and modeling means, have some defaults[2]. Some formal methods proposed have emerged recently to describe WSs at abstract level, but most of them are based on transition system models such as Mealy automata, Petri nets, etc[3–6]. Compared to these formal methods, π-calculus has the following advantages:

① It can present the interactions between new services and other participants by abstracting and refining means, then encode them with executable language;

② It can refine the abstract description from the executions of services to describe their actions, and transform it by reverse engineering means;

③ It can add protocols to the ports.

The π-calculus overcomes the inabilities in presenting the consistency and validating the correctness of web services flow languages, and it can solve some problems to some degree.

1 Inter-Description between π-Calculus and WS-BPLE

π-calculus formal language and WS-BPLE can be mapped and describe each other, and π-calculus can be adopted to design layer and middle layer of web services description language. On the one hand, it can describe the system abstract specification and form the mapping from abstract specification to execution language, so that the code framework will be automatically generated; on the other hand, the reasoning theory and the model examination of the π-calculus formal method can be used to verify the system[7–8] (see Fig. 1).

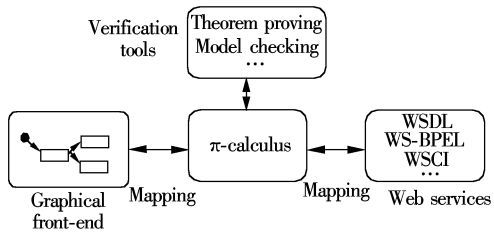

Fig. 1 Overview of the mapping between web services flow and π-calculus

1.1 π-calculus description toward WS-BPEL

The basic WS-BPEL activities include ⟨receive⟩, ⟨reply⟩, ⟨invoke⟩, ⟨assign⟩, ⟨throw⟩, ⟨terminate⟩, ⟨wait⟩, ⟨empty⟩, etc. We can use π-calculus to express the basic activities of WS-BPEL simply (see Tab. 1).

1.2 Mapping from π-calculus to WS-BPEL

We can get the basic route construction of web services flow as follows:

① Sequence $P.\ Q$; ② Choose $P + Q + R$; ③ Receive $x(a: T)$; ④ Synchronous $P \mid Q$; ⑤ Alternate $!P$; ⑥ Reply $\bar{x}\langle a: T\rangle$

The first five are used to deal with the combination process; the last one is used to deal with correspondence between processes.

Received 2006-04-25.

Biographies: He Tao (1973—), male, graduate; Miao Huaikou (corresponding author), male, professor, hkmiao@staff.shu.edu.cn.

Tab. 1 Basic WS-BPEL activities expressed by π-calculus

Basic activities of WS-BPEL	Corresponding π-calculus description	Comments
Sequence	$P.\ Q.\ R$	Allow a set of activities to be executed in order
Receive	$x(a)$	
Reply	$\bar{y}\langle \text{reply} \rangle$	
While	$!(li(\text{LoopRequest}).\ \bar{gi}\langle \text{MethodRequest} \rangle \mid \bar{gi}\langle \text{MethodRequest} \rangle.\ \text{genericTask}.\ (\bar{go}\langle \text{MethodResponse} \rangle + \bar{li}\langle \text{LoopRequest} \rangle) \mid go(\text{MethodResponse}).\ \bar{lo}\langle \text{LoopResponse} \rangle)$	The appointed activities proceed repeatedly, until the given term is no longer satisfied.
Throw	$P + \bar{f}\langle \text{processfault} \rangle \mid f(\text{processfault})$	Throw an error.
Pick	$m(\text{msgwaiting}) + t(\text{timeout}) \mid \bar{m}\langle \text{msgwaiting} \rangle + \bar{t}\langle \text{timeout} \rangle$	Allow a block in order to wait for a certain message's arrival, or report timeout.
Flow	$P \mid (A\text{“} \mid \text{”}B) \mid Q$	Allow appointing one or more synchronous proceeding activities. "Flow" is equal to a nested synchronous construction, and can be mapped into synchronous components.

We can realize the mapping from basic route construction to WS-BPEL.

1) Sequence $P.\ Q \rightarrow$

```
⟨sequence⟩
    webSeviceP
    webSeviceQ
⟨/sequence⟩
```

2) Choose $P + Q \rightarrow$

```
⟨sequence⟩
    webSeviceA
    ⟨switch⟩
        ⟨case condition = "c1"⟩
            webSeviceP
        ⟨/case⟩
        ⟨case condition = "c2"⟩
            webSeviceQ
        ⟨/case⟩
        ⟨otherwise⟩
            ⋮
        ⟨/otherwise⟩
    ⟨/switch⟩
    webSeviceB
⟨/sequence⟩
```

3) Receive $x(a: T) \rightarrow$

```
⟨receive partnerLink = "Requester"
        portType = "tns: PortTypeRqst"
        operation = "operationR"
        variable = "x"/⟩
```

4) Synchronous $P \mid Q \rightarrow$

```
⟨invoke partnerLink = "Adder"
        portType = "tns: requestResultPortType"
        operation = "operationRqst"
        inputVar = "x"
        outputVar = "y"/⟩
```

The receive/reply is composed with a service doing on the other side:

```
⟨receive partnerLink = "Requester"
        portType = "tns: requestResultPortType"
        operation = "operationRqst"
        var = "x"/⟩
        ⋮
⟨reply partnerLink = "Requester"
        portType = "tns: requestResultPortType"
        operation = "operationRqst"
        var = "y"/⟩
```

5) Alternate $!P \rightarrow$

```
⟨sequence⟩
    webSeviceA
⟨while casecondition = "c1"⟩
        webSeviceP
⟨activity webSeviceP/⟩
    ⟨/while⟩
⟨/sequence⟩
```

6) Reply $\bar{x}\langle a: T \rangle \rightarrow$

```
⟨invoke partnerLink = "Adder"
        portType = "tns: PortTypeRqst "
        operation = "operationRqst"
        inputVar = "x"/⟩
```

2 Mapping Web Services Flow Combination to π-calculus

The π-calculus is an effective expression form to web service flow combinations, and it can describe system behavior distinctly. However, there is a question of how to distinguish the services providers in the system and how to recognize the process channels used by services provider in the course that the logic relation of web services flow combination mapping to π-calculus expression. Three rules are given here to analyze the processes and channels.

Rule 1 The services provider which can provide atom service is recognized as one services process. The other complex services can be separated into a set of atom services and be executed by other services processes respectively.

Rule 2 The correspondence between services processes must pass through one channel. Two services providers cannot communicate each other if there is no channel between them.

Rule 3 Asynchronous messages are transferred through one channel between two services processes.

3 Web Services Flow Modeling Using π-Calculus

This paper takes a B2B e-commerce application as an example, which regards manufactory as the center to integrate web services among the manufactory, the dis-

tribution company, the goods-supplier and the bank, so that we can order and transfer account etc. on the web. The reached flow is the order flow of the manufactory. In the process, the manufactory sends the order form to the distribution company and prepays the bank. After the distribution company receives the order form, it will check the stock of the goods-supplier and check if it has received the money paid. If it is true, it will send the goods to the manufactory and get feedback from the manufactory. The flow is pictured by a synchronous combination pane graph of process algebra (see Fig. 2).

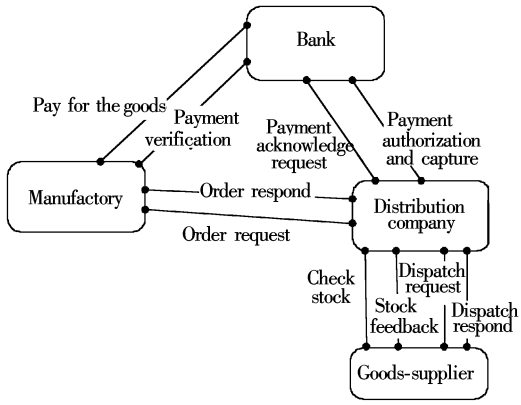

Fig. 2 E-commerce WSs flow synchronous combination pane graph

Two approaches are given here to describe WSs flow and model with π-calculus from different views.

3.1 Approach one

When describing a web services flow combination with π-calculus, we can regard the entire business flow as a process, the messages invoking services as an input to the process, and the result of services including output and fault messages generated by a service as an output of the process. This approach, which describes and models WSs flow with pure π-calculus, facilitates the interaction of each transaction and is propitious to put a hold on the entire business flow for us.

According to the approach, the transaction flow of the case above is shown in Fig. 3.

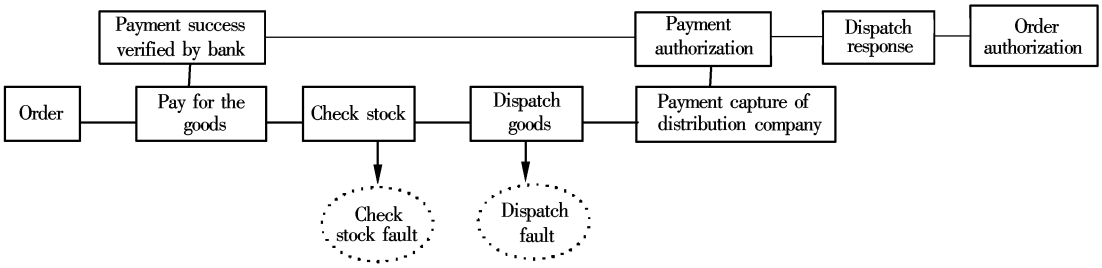

Fig. 3 Transaction flow graph

The π-calculus expression for this example is

ori(OrderRequest). (pvi⟨PayVerifyReqt⟩ | csri⟨CheckStockReqt⟩) | pvi(PayVerifyReqt). payVerifyTask. (pcri⟨PayCaptureReqt1⟩ | di⟨DispatchReqt1⟩) | csri(CheckStockReqt). checkStockTask. (di⟨DispatchReqt2⟩ + csf⟨CheckStockFault⟩) | (di(DispatchReqt1) | di(DispatchReqt2). dispatchTask. ((pcri⟨PayCaptureReqt2⟩ | do⟨DispatchResponse⟩) + df⟨DispatchFault⟩) | (pcri(PayCaptureReqt1) | pcri(PayCaptureReqt2)). payCaptureTask. pcro⟨PayCaptureResponse⟩ | (pcro(PayCaptureResponse) | do(DispatchResponse)). oro⟨OrderResponse⟩ | (csf(CheckStockFault) + df(DispatchFault)). pof⟨ProcessOrderFault⟩

3.2 Approach two

We can use approach two by the following steps: ① Distinguish the participation processes in WSs flow and take these processes as services processes of π-calculus; ② Take the solitary atom web service as an atom process, and transfer the messages of each related atom through one channel[9]. This approach, which describes and models WSs flow with type π-calculus, facilitates the expression of the relationship between processes and the external environment. In order to analyze and deduce correctly, we call $\Gamma = \{v_1: T_1, \ldots, v_n: T_n\}$ typing environment, which endow name v_i with type T_i. Each v_i presents itself in Γ only once. Typing judgment $\Gamma \vdash E$ means that the expression E is well-typed. In other words, all the variables of the expression are defined in Γ. $\Gamma \vdash x: T$ means that the type of name x is T; $\Gamma \vdash U \leqslant T$ means that the type U is a subtype of type T.

In Fig. 2, a channel x is set up between the manufactory process and the distribution company process to show the relationship between them. Channel y is the channel between the manufactory and the bank; channel z is the channel between the distribution company and the goods-supplier; Channel w is the channel between the distribution company and the bank.

$$\Gamma \vdash T_{\text{channel}} \leqslant T;\ \Gamma \vdash T_{\text{msg}} \leqslant T;\ \Gamma \vdash x, y, z, w: T_{\text{channel}}$$

$$\Gamma \vdash \text{msg}_i: T_{\text{msg}} (i \text{ is a random char string})$$

The model of the manufactory process with π-calculus is as follows:

$$\bar{x}\langle \text{msg}_{\text{orderRequest}} \rangle . \bar{y}\langle \text{msg}_{\text{pay}} \rangle . y(\text{msg}_{\text{payVerify}}) . (x(\text{msg}_{\text{orderResponse}}) + x(\text{msg}_{\text{err}}))$$

The model of the distribution company process with π-calculus is shown as follows:

$$x(\text{msg}_{\text{orderRequest}}) . \bar{z}\langle \text{msg}_{\text{checkstock}} \rangle . z(\text{msg}_{\text{stockCheckResult}}) . ([\text{msg}_{\text{stockCheckResult}} = \text{msg}_{\text{orderRequest}}] (\bar{w}\langle \text{msg}_{\text{payAckRequest}} \rangle$$

$$| \bar{z}\langle \mathrm{msg}_{\mathrm{dispatch}} \rangle) + \bar{x}\langle \mathrm{msg}_{\mathrm{err}} \rangle).$$
$$((w(\mathrm{msg}_{\mathrm{payResponse}}) | (z(\mathrm{msg}_{\mathrm{dispatchResponse}}) + \bar{x}\langle \mathrm{msg}_{\mathrm{err}} \rangle)) . \bar{x}\langle \mathrm{msg}_{\mathrm{orderResponse}} \rangle$$

In the above models, the error throw and disposal for stock checking failure and dispatch failure are carried through in the inner of sales agent, which is the inner operation. So we do not describe it in the model above, and use $\bar{x}\langle \mathrm{msg}_{\mathrm{err}} \rangle$ only to send error messages to the manufactory process.

The model of the goods-supplier process with π-calculus is

$$z(\mathrm{msg}_{\mathrm{checkStock}}) . \bar{z}\langle \mathrm{msg}_{\mathrm{stockCheckResult}} \rangle . z(\mathrm{msg}_{\mathrm{dispatch}}) . \bar{z}\langle \mathrm{msg}_{\mathrm{dispatchResponse}} \rangle$$

The model of the bank process with π-calculus is

$$y(\mathrm{msg}_{\mathrm{pay}}) . \bar{y}\langle \mathrm{msg}_{\mathrm{payVerify}} \rangle . w(\mathrm{msg}_{\mathrm{payAckRequest}}) . \bar{w}\langle \mathrm{msg}_{\mathrm{payResponse}} \rangle$$

4 Conclusion

This paper analyzes the syntactic structure of WS-BPEL and gives an approach to analyzing and modeling WSs flow with π-calculus. It describes the mapping relationship of syntactic elements of WS-BPEL and π-calculus, discusses the approaches to web services flow modeling from different views, and illustrates it by developing and modeling an e-commerce web service flow application. It is not only doable in theory but also can be verified and traced by some verification tools such as automated verification tool MWB[10], interactive verification tool PiM, etc. In a word, it provides a new kind of valid means to analyze workflow of WSs, and it is worthy of further research and study.

References

[1] Tang Yu, Chen Luo, He Kaitao, et al. SRN: an extended Petri-net-base workflow model for web service composition [A]. In: *Proc of IEEE International Conference on Web Services* [C]. San Diego, California, 2004. 591 – 599.

[2] Salaun Gwen, Bordeaux Lucas, Schaerf Marco. Describing and reasoning on web services using process algebra [A]. In: *Proc of IEEE International Conference on Web Services*[C]. San Diego, California, 2004. 43 – 50.

[3] Berardi D, Calvanese D, De Giacomo G, et al. Automatic composition of e-services that export their behavior[A]. In: *Proc of ICSOC'03, LNCS* [C]. Trento, Italy, 2003, **2910**: 43 – 58.

[4] Gao Yong, Liu Yu, Xie Kunqing, et al. A Petri nets-based model for web service composition [J]. *Computer Engineering*, 2006, **32**(6): 17 – 18. (in Chinese)

[5] Ren Zhihong, Cao Jiannong, Chan A T S, et al. Toward a formal approach to composite web service, construction and automation [A]. In: *Proc of International Conference on Parallel Processing*[C]. Kaohsiung, 2003. 436 – 443.

[6] Zhang Jia, Chang C K, Chung Jenyao, et al. WS-net: a Petri-net based specification model for web services [A]. In: *Proc of IEEE International Conference on Web Services* [C]. San Diego, California, 2004. 420 – 427.

[7] Sangiorgi D, Walker D. *The π-calculus: a theory of mobile processes* [M]. Cambridge University Press, 2001. 580.

[8] Salaun Gwen, Bordeaux Lucas, Schaerf Marco. Describing and reasoning on Web services using process algebra [A]. In: *Proc of IEEE International Conference on Web Services*[C]. San Diego, California, 2004. 43 – 50.

[9] Gong Hongquan, Zhao Wenyun, Xu Ruzhi, et al. A research on pi-calculus based component evolution [J]. *Acta Electronica Sinica*, 2004, **32**(12): 238 – 242.

[10] Victor B, Moller F. The mobility workbench-a tool for the pi-calculus [A]. In: *Proc of CAV'94, LNCS*[C]. Stanford, California, 1994, **818**: 428 – 440.

基于 π-演算的 web 服务流的分析与建模

何 涛 缪淮扣 钱忠胜

(上海大学计算机工程与科学学院,上海 200072)

摘要:为了提高 web 服务流程的有效性和可靠性,提出了 π-演算形式化方法. 该方法能克服 web 服务流语言不能表明一致性及进行验证等缺陷. 讨论了 web 服务流的 π-演算分析和建模,对 WS-BPEL 语法元素及动态基本活动进行了分析和形式化刻画,同时,对 π-演算和 WS-BPEL 的相互映射进行了描述. 最后,利用 π-演算描述 web 商业流程基本结构,并从不同角度探讨了用 π-演算进行商业建模的方法,有效地对 WS-BPEL 所述商业流程进行了 π-演算的分析和建模.

关键词:BPEL;web 服务;工作流;π-演算

中图分类号:TP301