Applied Mathematics & Information Sciences 5 (2) (2011), 25S-31S
– An International Journal

A Linear Logic Representation for BPEL Process Protocol*

Jian Wu[1] and Lu Jin[2]
[1]Department of Computer Science and Technology, Zhejiang University, Hangzhou, China
[2] Department of Computer Science and Technology, Zhejiang University, Hangzhou, China
Email Address: jinluzju@gmail.com

Received June 22, 2010; Revised December 21, 2010

Business Process Execution Language (BPEL) is a powerful tool for describing web services compositions. The protocol of a BPEL process indicates the order of messages, in which it sends or receives messages, as well as the structure of the internal logic. Although formal methods have been used to model BPEL, the purposes of these researches are to verify BPEL process and eliminate ambiguity. Recent research advances in dynamic adaptation and reconfiguration of service processes require a new formal method to express BPEL process with the ability of problem solving. This paper presents a linear logic based representation for BPEL process. Our approach expresses both basic and structured activities in BPEL. With the help of proof-searching tools, our approach set up a formal foundation for (semi)automatically solving more challenging issues of service computing.

Keywords: Web Service, Service Adaptation, Linear Logic, BPEL.

---

1 Introduction

Web services are software entities capable of exchanging XML based data. With well-defined interfaces (Web Service Description Language, WSDL) [1], web services can be quickly composed as service processes to complete more complex tasks. One of the well-received specifications for describing service composition is the Business Process Execution Language (BPEL) [2]. BPEL is supported by most major software vendors and applied to various fields.

BPEL is a complex programming language for describing sophisticated business processes. A business process performs numerous actions to complete a business transaction. The order in which actions are executed is called business protocol. The same idea of protocol has been embedded into the BPEL language. With structured activities and other intrinsic mechanisms, such as fault and exception handling, a BPEL process is capable of describing and executing complex business processes.

* This research is partially supported by National Sci & Tech Research Program of China (No. 2009ZX01043-003-003), the 863 Program (No. 2008AA01Z141), the National NSF of China (No. 60873224, No. 60873045, No.60803004), Zhejiang Provincial NSF (No. Y106598), and Zhejiang Provincial Sci &Tech Program (No. 2008C03007).

Since BPEL is not equipped with formal semantics, the protocol of a BPEL process is very difficult to be formally reasoned. Several formal methods [3] have been used to model BPEL processes. R. Lucchi extended pi-calculus to describe activities and advanced structures, such as compensation handler, in BPEL [4]. N. Lohmann provided a Petri-net based model for BPEL and a corresponding tool, BPEL2oWFN [5]. These models are then used to eliminate ambiguity and verify various properties.

Recent advances in service adaptation [6] require formal methods with ability to provide solutions to mismatch problems. Although existing approaches are still used, they are more often used as an intermedial model to bridge different process definition languages and avoid ambiguity. The adaptation process is usually constructed by an algorithm proposed by the authors. Without sufficient test cases to prove their algorithms, the methods are less convincing.

Linear logic (LL) [7] is a branch of logic system. In combination with proof-searching tools, such as Coq [8] and llprover [9], linear logic is capable of providing solutions to particular problems (semi)automatically. In our previous work [10] we proposed a linear logic-based automatic method for process adaptation. In this paper we propose a formal representation of BPEL process. Both basic activities and structured activities are specified using LL sequents. As a result, our approach is capable to capture fully the BPEL process behavior and to generating a linear logic-based representation of process protocol, in which the message orders and internal structures are defined.

The rest of this paper is organized as follows: Firstly we introduce some background information on linear logic. Then we present the LL representation of BPEL basic activities and structured activities. After that an example is given. Finally we summarize our work and give an outlook into our future work.

2 Linear Logic Background

Linear logic (LL) was introduced by Girard [7] to provide a logical way for coping with resources. The fundamental notion in LL states that "*A is consumed while producing B*". As a result, the number of formulae is aware to the logic, that is to say one copy of *A* is different than two or more copies of *A*. This unique feature has made LL popular among computer scientists. The LL grammar that we use in this paper is presented as follows:

$$A ::= A \otimes A \mid A \oplus A \mid A \& A \mid A \multimap A \mid !A \mid u : A \mid 1.$$

The *simultaneous conjunction* $A \otimes B$, also called *multiplicative connective*, suggests the possession of both *A* and *B* at the same time. The *disjunction* $A \oplus B$, also called *external choice*, suggests that either *A* or *B* is available. The *alternative conjunction A&B*, also called *internal choice*, represents that either *A* or *B* is produced. The *linear implication* $A \multimap B$ states that *A* is consumed while achieving *B*. The "*of course*" modality can be applied to resource *A* if *A* could replicate itself without any resource. The *label u*:*A* means that *A* is labeled with *u*. The *trivial goal 1* represents a goal that requires no resources to achieve.

In this paper we use LL formulae to represent messages exchanged by different services and processes. Since LL does not distinguish data types, we use different names for formulae to represent different types of messages. In this way we are able to reduce the complexity of reasoning process.

3 Linear Logic Model for BPEL Protocol

A BPEL process implements its business logic by performing *activities*. There are two major categories of activities in the BPEL specification: *basic activity* and *structured activity*. Basic activities describe elemental steps of the process behavior, while structured activities encode control-flow logic.

Basic activities in BPEL are capable of exchanging messages, manipulating data, controlling internal state etc. They are defined as follows:

Definition 1. A *basic activity* is defined as an 8-tuple *BA=<N, I, O, P, E, F, CT, ST>*.

In the definition, *N* represents the name of this activity; *I* and *O* represent incoming messages/input variable, outgoing message/output variable; *P*, *E* and *F* represent precondition, effect and fault respectively; *CT* is the collection of control tokens, which functions as a control mechanism for the execution of each activity; *ST*, state transition, is the collection of LL sequents that describe the basic activity's behavior.

There are three notions within this definition that are not originally from the BPEL specification: *effect*, *precondition* and *control token*. *Precondition P* and *effect E* can be used to represent internal states of a BPEL process. Furthermore we use *preconditions* and *effects* to represent the <*source*> and <*target*> elements. They are standard elements in basic activity to specify the *source* and the *target* of a *link*, which is a synchronization mechanism for parallel processes. Under the same *link*, the *source* activity must finish before the execution of the *target* activity. Thus it is possible for activities in different processes to synchronize. In our model the *source* activity generates a formula as the *effect* while the *target* activity requires that particular formula as the *precondition*.

Control tokens are used to control the execution of each activity. There are three kinds of control tokens in the definition of activity: CT_{in}, CT_{next}, and CT_f. CT_{in} represent the control token that activates the activity. CT_{next} represents the control token that activates the next activity while CT_f represents fault. By assigning one activity's CT_{next} to the next activity's CT_{in}, we are able to embed sequence structure into the definition of activity. Furthermore, since the proof search of linear logic is undetermined, the application of control tokens can reduce the cost of proving theorems.

The logic sequent *ST* represents the behavior of a basic activity. Due to their different functions not every basic activity shares the same composition and LL representation.

- The *invoke* activity is capable of calling web services. It is defined as $CT_{in} \otimes P \ddot{} \quad (CT_{temp} \otimes O) \& CT_f$ and $CT_{temp} \otimes I \ddot{} \quad (E \otimes CT_{next}) \& CT_f$.

- The *receive* activity is used to provide services to partners through inbound message activities. It is defined as $CT_{in} \otimes P \otimes I$ ¨ $(E \otimes CT_{out})\,\&\,CT_f$.
- The *reply* activity is used to send a response to a previously accepted request. It is defined as $CT_{in} \otimes P$ ¨ $(O \otimes E \otimes CT_{out})\,\&\,CT_f$.
- The *assign* activity is capable of manipulating variables and data within the business process. The *ST* of the *assign* activity is defined as $CT_{in} \otimes P \otimes I$ ¨ $(O \otimes E)\,\&\,CT_f$.
- The *throw/rethrow* activity is used to signal internal fault explicitly or to propagate existing faults. It is represented as $CT_{in} \otimes P$ ¨ $E\,\&\,CT_f$.
- The *wait* activity delays the process for a specific period of time or until a certain deadline is reached. It is defined as Δ گ$CT_{in} \otimes P$ ¨ $E\,\&\,CT_f$. Δ represents the time consumed by *wait*.
- The *empty* activity represents activities that do nothing. It is defined as $CT_{in} \otimes P$ ¨ $E\,\&\,CT_{next}$.
- The *exit* activity is used to terminate the business process instance. It is defined as $CT_{in} \otimes P$ ¨ F .

Structured activities describe the order in which a collection of activities is executed. By the composition of activities, structured activities are able to express the control patterns, handle faults and external events, and coordinate message exchanges between process instances involved in a business protocol. The definition of structured activities is described as follows:

Definition 2. A *structured activity* is defined as an 8-tuple *SA=<N, I, P, E, F, EA, CT, TT* >, in which

- *N, I, P, E, F* represent name, incoming message, precondition, effect and fault respectively;
- $EA=\{EA_1,EA_2,\ldots,EA_n\}$ is the collection of embedded activities. If only one activity is embedded, EA is used to represent the embedded activity;
- $CT=\{CT_{in}, CT_{next}\}$ is the collection of control tokens for this structured activity;
- *TT*, token transition, is the collection of LL sequents that describe the activity's structure.

The definition of structured activities is similar to that of basic activities. *Token transitions, TT* for short, are used to describe the control flow encoded into the structured activities. Structured activities include: *sequence, if, pick, flow, while, repeat Until, for Each*. The token transitions for each structured activity are defined as follow:

- The *sequence* activity allows activities to be executed in the lexical order in which they are defined. With CT_{out} and CT_{in} in every basic activity it is possible to control the sequence of activities by assigning one activity's CT_{out} as another activity's CT_{in}.

- The *if* activity offers the conditional branch structure that represents internal choice. It is defined as $CT_{in}\blacklozenge EA_1.CT_{in}\&EA_2.CT_{in}\&\ldots\&EA_n.CT_{in}$.
- The *pick* activity offers the ability to respond to external events. It is represented as $CT_{in}\blacklozenge(I_1\blacklozenge EA_1.CT_{in})\oplus(I_2\blacklozenge EA_2.CT_{in})\oplus\ldots\oplus(I_{n-1}\blacklozenge EA_{n-1}.CT_{in})\oplus EA_{n.}CT_{in}$. Each sequent $I_m\blacklozenge E_m.CT_{in}$ ($0<m<n$) corresponds to an <*onMessage*> element and depicts that the *pick* activity consumes an incoming message and execute one activity EA_m. The last EA_n represents the activity in the <*onAlarm*> branch, if defined.
- The *flow* activity offers concurrency and synchronization. The *TT* of the *flow* activity consists of two parts: start and termination. The start is represented as $CT_{in}\blacklozenge E_1.CT_{in}\otimes E_2.CT_{in}\otimes\ldots\otimes E_n.CT_{in}$. The termination is represented as $E_1.CT_{next}\otimes E_2.CT_{in}\otimes\ldots\otimes E_n.CT_{next}\blacklozenge CT_{next}$.
- The *while* activity provides the ability to execute a contained activity repeatedly. The *TT* of the *while* activity also contains two sequents: $CT_{in}\blacklozenge E.CT_{in}\&CT_{next}$ and $E.CT_{next}\blacklozenge CT_{in}$.
- The *repeatUntil* activity also furnishes the function for repeated execution of a contained activity. The *TT* of the *repeatUntil* activity is defined as $CT_{in}\blacklozenge E.CT_{in}$ and $E.CT_{next}\blacklozenge E.CT_{in}\&CT_{next}$.

After both basic activities and structured activities are defined, the BPEL process can be quickly defined as a union of all activities.

Definition 3. A BPEL process is defined as a 4-tuple $P=<IV, CT, BA, SA>$, where

- $IV=<N, I, O, P, E, F>$ is the interface view of the service process, which consists of canonical name, incoming and outgoing messages, preconditions, effects, and faults.
- $CT=\{CT_1, CT_2,\ldots, CT_n\}$ is the collection of control tokens;
- $BA=\{A_1, A_2,\ldots, A_m\}$ is the collection of basic activities;
- $SA=\{S_1, S_2,\ldots, S_k\}$ is the collection of structured activities.

4 Example Walkthrough

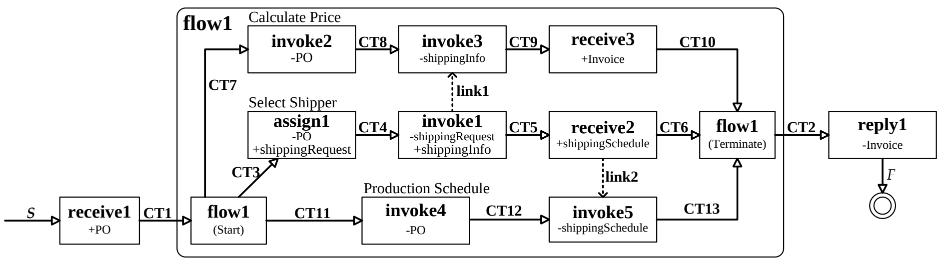

Fig. 1. Linear Logic Model for Purchase Order Process

Figure 1 demonstrates the linear logic model for the purchase order process from the section 5.1 of BPEL 2.0 specification[2]. Activities are represented using rectangles. Each activity is labeled its message directions, '+' for incoming message and '-' for outgoing message. After a purchase order from the client is received, the process initiates three parallel subprocesses: i) “Calculate Price” calculates the final price of the order and receives the invoice produced by a third party service; ii) “Select Shipper” arranges transportation and calculates the shipping price; iii) “Production Schedule” schedules the production and shipment for the order. Dotted arrows represent control links used for synchronization across concurrent activities. *Link1* indicates the shipping price is required to finalize the price calculation. *Link2* shows that the shipping date is required for the complete fulfilment schedule. After the three concurrent paths have completed their execution, an invoice is returned to the customer.

The process definition is listed as follows.

$$P=\langle IV, CT, BA, SA\rangle$$

- *IV*=<’Purchase Order Process’, *POMesage*, *InvMessage*, Null, Null, cannotCompleteOrder>
- $CT=\langle CT_1, CT_2, \ldots CT_{13}, S, \Phi\rangle$
- *BA*=<*invoke1, invoke2, invoke3, invoke4, invoke5, receive1, receive2, receive3, reply1, assign1,*>
- *SA*=<*flow1*>

The detailed linear logic representation of each activity in the purchase order process is listed in Table 1. Note that initials are used for clear representation. *IN* stands for *Invoice*. *SI* stands for *shippingInfo*. *SR* stands for *shippingRequest*. *SS* stands for *shippingSchedule.*

Activities	I	O	P	E	F	CT	ST
receive1	*PO*	*N/A*	*N/A*	*N/A*	*N/A*	S, CT_1	$S \otimes PO \blacklozenge CT_1$
flow1	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	$CT_{1,2,3,6,7,11,13}$	$CT_1 \blacklozenge CT_3 \otimes CT_7 \otimes CT_{11}, CT_6 \otimes CT_{10} \otimes CT_{13} \blacklozenge CT_2$
reply1	*N/A*	*IN*	*N/A*	*N/A*	*N/A*	CT_2, Φ	$CT_2 \blacklozenge IN \otimes \Phi$
invoke2	*N/A*	*PO*	*N/A*	*N/A*	*N/A*	$CT_{7,8}$	$CT_7 \blacklozenge PO \otimes CT_8$
invoke3	*N/A*	*SI*	*link1*	*N/A*	*N/A*	$CT_{8,9}$	$CT_8 \otimes link1 \blacklozenge SI \otimes CT_9$
receive3	*IN*	*N/A*	*N/A*	*N/A*	*N/A*	$CT_{9,10}$	$CT_9 \otimes IN \blacklozenge CT_{10}$
assign1	*PO*	*SR*	*N/A*	*N/A*	*N/A*	$CT_{3,4}$	$CT_3 \otimes PO \blacklozenge SR \otimes CT_4$
invoke1	*SI*	*SR*	*N/A*	*link1*	*CCO*	$CT_{4,5}, CT_{inv1}$	$CT_4 \blacklozenge (SR \otimes CT_{inv1}) \& CT_{CCO}, CT_{inv1} \otimes SI \blacklozenge link1 \otimes CT_5$
receive2	*SS*	*N/A*	*N/A*	*link2*	*N/A*	$CT_{5,6}$	$CT_5 \otimes SS \blacklozenge link2 \otimes CT_6$
invoke4	*N/A*	*PO*	*N/A*	*N/A*	*N/A*	$CT_{11,12}$	$CT_{11} \blacklozenge PO \otimes CT_{12}$
invoke5	*N/A*	*SS*	*link2*	*N/A*	*N/A*	$CT_{12,13}$	$CT_{12} \otimes link2 \blacklozenge SS \otimes CT_{13}$

4 CONCLUSION & FUTURE WORKS

In this paper a linear logic-based formal model for BPEL process is proposed. The contributions of this work include: i) Linear logic is incorporated into our method. With the help of proof-searching tool and proper problem setup, more challenging service computing problems, such as service adaptation, can be solved (semi)automatically. ii) A linear logic-based formal semantic for basic activities is proposed, which is capable of expressing message exchanges and the *link* mechanism. iii) A linear logic-based semantic for structured activities is proposed. Structures, such as sequence, choice, parallel and loop, can be easily described. In our future work more mechanisms, such as fault handling, event handling and compensation, will be included. We are currently working on an automatic tool that can transform XML based BPEL description into LL sequents.

References

[1] Web Service Description Language (WSDL) Version 2.0, W3C Recommendation (2007).

[2] Web Services Business Process Execution Language (BPEL) Version 2.0, OASIS Standard (2007)

[3] F. v. Breuge and M. Koshkina, Models and Verification of BPEL, *http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf* (2006).

[4] R. Lucchi and M. Mazzara. A pi-calculus based semantics for WS-BPEL. *Journal of Logic and Algebraic Programming.* 70 (2007) 96-118.

[5] Niels Lohmann. A Feature-Complete Petri Net Semantics for WS-BPEL 2.0, *In Web Services and Formal Methods, Forth International Workshop*, Milan, 2007.

[6] M. Dumas, B. Benatallah and H.R.M. Nezhad, Web Service Protocols: Compatibility and Adaptation, *IEEE Data Engineering Bulletin.* 31 (2008) 40-44.

[7] J.-Y. Girard, Linear logic, *Theoretical Computer Science*, 50 (1987) 1-101.

[8] Coq, a formal proof management system. *http://coq.inria.fr/*

[9] llprover, A Linear Logic Prover, *http://bach.istc.kobe-u.ac.jp/llprover/*

[10] L. Jin and J. Wu, A Linear Logic Based Approach for Generating Deadlock Adapters, *Proceedings of 2010 Asia Pacific Service Computing Conference*, Hangzhou, China.

Jian Wu received his B.Sc and Ph.D degree from Zhejiang University in 1998 and 2004, respectively. He is now an Associate Professor in the Department of Computer Science and Technology. His research interests include Service Computing, Data Mining, Grid Computing and Semantic Web.

Lu Jin received his B.Sc from Zhejiang University in 2005. He is now a Ph.D student in the Department of Computer Science and Technology in Zhejiang University. His research interests include Service Computing and Semantic Web.