A Hybrid Approach for Generating Compatible WS-BPEL Partner Processes

Simon Moser[1], Axel Martens[2], Marc Häbich[1,3], and Jutta Mülle[3]

[1] IBM Böblingen Lab, Business Process Solutions,
Böblingen, Germany
smoser@de.ibm.com
[2] IBM TJ Watson Center, Component Systems Group,
Hawthorne (NY), USA
amarten@us.ibm.com
[3] University of Karlsruhe, Information Systems Group,
Karlsruhe, Germany
muelle@ipd.uka.de

Abstract. The *Business Process Execution Language for Web Services* provides an technology to aggregate encapsulated functionalities for defining high-value Web services. For a distributed application in a B2B interaction, the partners simply need to expose their behavior as BPEL processes and compose them. Still, modeling and composing BPEL processes can be complex and error-prone. With formal methods like Petri nets, it is possible to analyze crucial properties (e.g. compatibility) effectively. In this paper, we present a method that automatically generates compatible partner BPEL processes for a given BPEL processes. Our hybrid approach makes use of formal methods, but also incorporates the structure of the original BPEL process model, such that the generated partner process is easier to understand and manage.

Keywords: Business Process Modeling, Web Service, WS-BPEL, Behavioral Compatibility, Tool based Verification, Petri nets.

1 Introduction

The *Business Process Execution Language for Web Services* BPEL4WS [7] is becoming the standard for modeling Web Service based business processes. A BPEL process implements one Web Service by specifying its interactions with other Web Services (which might be BPEL processes, too). BPEL processes consist of two kinds of activities: *Basic activities* to communicate to the outside, to manipulate data or to interfere with the control flow and *structured activities* to aggregate other activities, i. e. to build the control structures of the process.

For two BPEL processes to interact, interfaces with operations and message types have to be defined separately in WSDL [2] and included into a *partner link* which specifies the interfaces of any given set of interacting BPEL processes. In the online shop example (Figure 1), the client process provides two interfaces: the StartUp Interface towards the initiating component and the Order Client Interface

S. Dustdar, J.L. Fiadeiro, and A. Sheth (Eds.): BPM 2006, LNCS 4102, pp. 458–464, 2006.

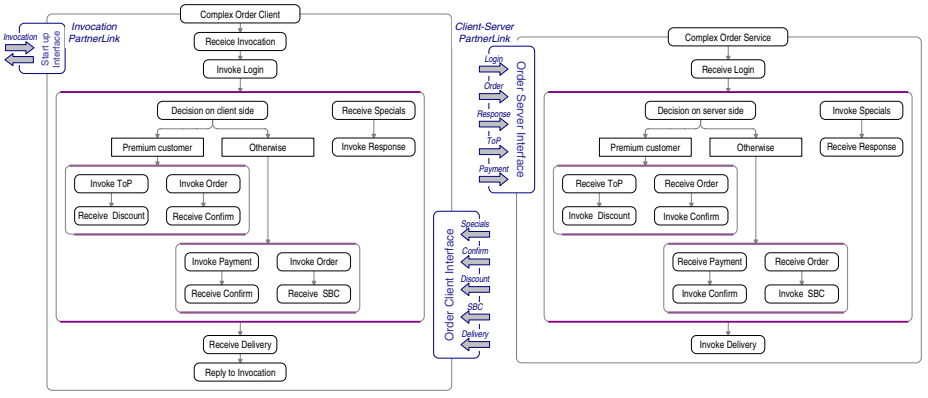


Fig. 1. Online shop example – incompatible BPEL processes of client and server

towards the shop's service. Moreover, the client requires the online shop's Web service to provide the Order Server Interface. Obviously, two BPEL processes can only be composed iff the provided interface of each process equals the required interface of the other. But this *syntactic compatibility* between the interfaces is not sufficient for the successful interaction of two BPEL processes.

An additional requirement – called *behavioral compatibility* [6] – is needed to ensure successful composition. In our example, the client process transmits the login data and makes a decision whether to act as a premium customer or as a regular customer, but its decision is not synchronized with that of the server process, although it is crucial to the further interaction. In case the client acts as a premium customer, it sends its order along with its terms of payment (ToP), and awaits discount information and confirmation. However, the server process might treat him like a regular customer because his last order was too long ago. In that case, the server process will acknowledge the order with the standard business conditions (SBC) and await payment. The concurrent conversation (Invoke Specials) happens regardless of either party's decision. In the end, both processes are waiting and neither can continue on its own - a classic *deadlock* situation. Obviously, the behavior of these two BPEL processes is not compatible.

Handing out an abstract process model to the partner so he can model his process accordingly is one solution to ensure behavioral compatibility, cf. [6]. However, modeling is time consuming and error-prone, and the partner might need many attempts to build a process that is actually compatible. A more elegant solution is to create a template of the partner's process (PP for short) out of the original process (OP) and to hand out this template instead . Thus, the partner only needs to refine the template according to his needs while behavioral compatibility is guaranteed. Similar ideas have been proposed for example in [4]. In this article, we will present a new approach to automatically generating guaranteed compatible BPEL PP in a hybrid approach, which combines the structural and behavioral approaches described in the following.

2 Structural Approach

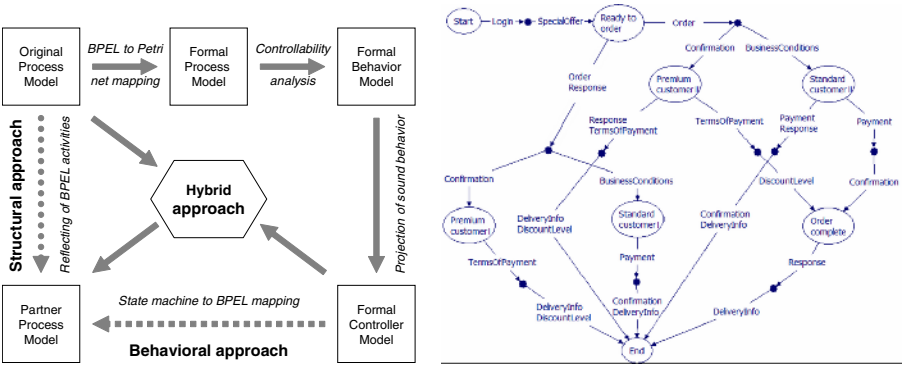
This approach uses the interaction patterns and control structures defined in BPEL processes. It parses the structure of the OP and reflects it using the *duality* between two BPEL activities (Table 1). The partner generation aims to deliver a process template that interacts correctly with the OP, and that has to be further refined by the partner. Hence, the generation focusses on communicating and structuring activities only, while internal activities of the OP (wait, assign, empty, ...) are ignored/mapped to empty. Communication between BPEL processes is either *asynchronous* or *synchronous*, in which case the activity that calls is blocked until a response has been sent. Structured activities define the *control flow* of a BPEL process. Generally, sequential activities in one process can be mapped to sequential execution within the other. Sometimes the blocking in synchronous communication requires multiple parallel threads in the generated PP (cf. [3]). So the most general approach is to map each sequence into a flow and to express the precedence constraints with flow links. Parallel execution in one process can always be mapped to parallel execution within the other.

Mapping *choices* is more complicated. An *externally* determined choice within the OP (pick) can be mapped into an *internal* choice of the PP (switch), where each case branch exchanges messages w. r. t. the communication style – although the mapping of onAlarm branches is not clear. The mapping of a switch activity into pick activity, however, is not possible in general, because there might be no distinguished messages sent in the case branches at all. Mapping instead a switch into a switch might lead to unsynchronized decisions of both BPEL processes, cf. Figure 1. A switch without communicating activities is just an internal activity. If we see a while activity as a kind of switch that loops in one case and does nothing in the otherwise case, the same problems apply to while (cf. [3]).

The client process shown in Figure 1 was generated using this approach. Since the approach is based on static mapping rules, it is very fast, and the generated

Table 1. Conceptual dualities between BPEL activities

Basic process activity	*partner activity*
receive with/without reply	synchronous/asynchronous invoke
synchronous/asynchronous invoke	receive with/without reply
wait, assign, empty, ...	empty or ignored
Structured process activity	*partner activity*
sequence	sequence (flow)
flow	flow (sequence)
pick	switch
onMessage with/without reply	case branch with sync/async invoke
onAlarm	N/A
switch	pick, switch or empty
while	while
scope	scope
handler	N/A



(a) Partner generation approaches (b) Communication graph of server process

Fig. 2. WOMBAT4WS's formal background

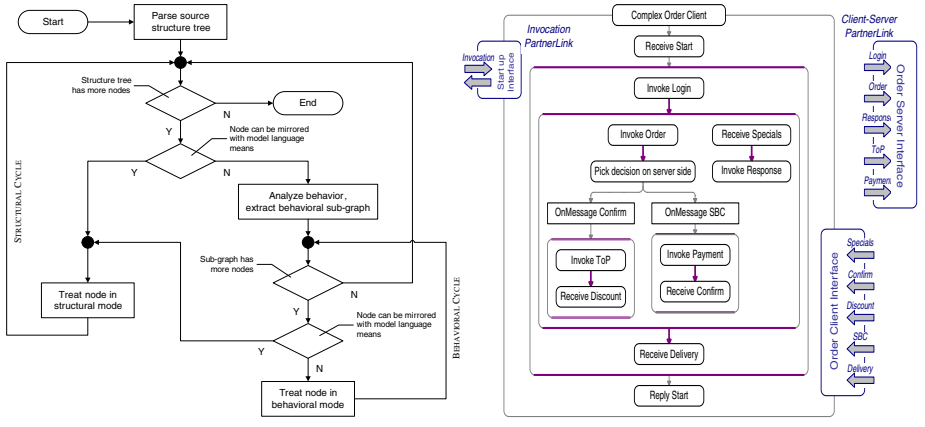
PP's structure is as simple (or complex) as the OP. In general, this makes the PP easily manageable/understandable to humans. These are two major advantages. But a major drawback is that compatibility is not guaranteed if internal decisions within the OP influence the communication between the parties. Moreover, the approach does not cover timers, handlers and complex flow link structures.

3 Behavioral Approach

The behavioral approach uses a formal mathematical analysis of the OP model. It consists of four steps, as shown in Figure 2(a). In the first step, the BPEL OP is transformed into a formal Petri Net representation, see also [5,1,8]. To keep the relation to the BPEL process, all Petri net elements are grouped into *block structures*, each representing exactly one element of the process. The resulting formal process model is called *BPEL-annotated Petri net* (BPN) [6].

However, the BPN is intermediary: In the second step, the communication behavior of the BPN is analyzed. Martens [1] presents an algorithm for analyzing the controllability of a BPEL process that generates a *communication graph* of the BPN model. This graph is the external, i. e. the partners', view on the process. Figure 2(b) shows a subset of the communication graph of the server process from the initial example. Formally, it is a state machine with two different kinds of states. A *visible* state (drawn as a white ellipse) refers to reachable states of the BPN in which input messages are expected. Each outgoing edge of such a state is labeled with a message which the BPN is able to receive in this state. A *hidden* state (drawn as a filled circle) is of intermediary nature. Each outgoing edge is labeled with a message the BPN can send in response to the consumed input.

With all possible sequences of input/output messages of the OP, the third step eliminates those sequences that may produce unwanted situations like deadlocks. The described behavior of the initial client process, for example, is one such communication sequence. The projection yields a sub-graph that contains only sound communication sequences and that represents the controller model for



(a) The Hybrid Algorithm (b) Generated compatible client processes

Fig. 3. Hybrid approach for generating behaviorally compatible partner processes

the BPEL OP. Figure 2(b) shows that sub-graph for the server process of the online shop example. The controller model is transformed into the BPEL PP model, by mapping each edge to a communicating BPEL activity while the control flow is built around them w. r. t. the causal order [3]. In contrast to the structural approach, the behavioral method has been proven to produce a behaviorally compatible PP, if there is one possible at all. Still, the main drawback of this method is that it is computationally expensive and will actually consider all possible sequences of communication activities, e. g. for 12 concurrent communication activities in the OP, there are $12! = 47,900,1600$ possibilities to order them. Hence, the method will often yield a PP that is too complex and rather hard for the partner to refine, as Figure 2(b) shows. The generation of the communication graph alone is exponential w. r. t. the size of the OP (cf. [1]).

4 Hybrid Approach

In isolation, both approach have strengths but also drawbacks. The *hybrid approach* combines their advantages without inheriting their deficits: it uses the structural approach *whenever possible*, to make the transformation fast and the result less complex, and the behavioral approach *whenever necessary*, to guarantee behavioral compatibility. The approach combines the hierarchical, tree-like structure gained by parsing the OP into a *structure tree* (so structured activities form intermediate nodes and basic activities form leaf nodes) with the formal controller model of the OP gained with the behavioral approach.

The hybrid algorithm shown in Figure 3(a) always starts in the structural mode (Structural Cycle): It parses the structure tree as defined by the OP. Then, it enters a loop visiting all nodes of that tree. In the so-called *mirror decision*, it decides for each node in the structure tree whether it can be mirrored with BPEL means. If this is true, then on the basis of the conceptual dualities,

a corresponding BPEL activity for the current node is generated, named and inserted into the result structure tree. If it is false, i. e. if the activity cannot be reflected by structural means, then the node has to be decomposed into sub-structures, i. e. the behavior of the current activity is analyzed by zooming into it and dissecting it into a sub-tree of connected sub-activities. Now, in the Behavioral Cycle, each node of that sub-graph is classified in a decision: If the activity can be reflected with structural means, is handed over to the structural cycle. Else, it is treated in turn according to the formal controller model. Once the sub-graph has been processed completely, the algorithm loops back to the initial *mirror decision* to continue with the next ordinary node. The client process shown in Figure 3(b) was generated using the hybrid approach.

In short, whenever a pattern occurs that cannot be transformed by simply reflecting it, e. g. the internal Decision on Server side (cf. Figure 1), the algorithm switches into behavioral mode to extract information about the pattern's behavior from the communication graph and generates *receive* or *pick*, and *reply* or *invoke* activities for all messages in that sub-graph. Finally, it connects these generated activities by control flow links and/or embeds them into structured activities. Since the structure tree may impose different operation modes, the algorithm will switch back and forth between the two modes depending on the patterns it encounters. For more details see [3].

5 Conclusion

In this paper, a method to automatically generate compatible BPEL partner processes was presented. This method had to satisfy three major requirements: (i) the generated PP had to be a valid BPEL process, (ii) it had to be behaviorally compatible to the OP, and (iii) it had to be sufficiently compact and simple enough to be understood and refined by a human process modeler. Neither a pure structural approach, nor a pure behavioral approach meet those requirements completely. On detailed examination of the two approaches, a third, *hybrid approach*, was developed, which coupled the advantages while overcoming their drawbacks. The hybrid approach connects the structural process elements, as defined by BPEL, with the formal mathematical analysis result (represented by the communication graph). As demonstrated in this paper, the hybrid approach couples behavioral compatibility with human readability and manipulability. A different solution to overcome behavioral incompatibility between two BPEL process models is trying to generate a third one which acts as an *adapter* between those two. This is the topic of a currently conducted research project.

References

1. A.Martens. Analyzing Web Service based Business Processes. In Maura Cerioli, editor, *Proc. of FASE'05*, LNCS 3442, Edinburgh, Scotland, April 2005. Springer.
2. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *WSDL – Web Services Description Language.* W3C, Standard, Version 1.1, March 2001.

3. M. Häbich. Reverse Transformation of Petri Net-Based Communication Graphs to BPEL4WS in Distributed Web Service Environments. Master's thesis, 2005.
4. S. Haddad, T. Melliti, P. Moreaux, and S. Rampacek. Modelling Web Services Interoperability. In *Proc. of ICEIS04*, 2004.
5. R. Hamadi and B. Benatallah. A Petri Net based Model for Web Service Composition. In *Proc. of ADC 2003*. Australian Computer Society, Inc., 2003.
6. A. Martens, S. Moser, A. Gerhardt, and K. Funk. Analyzing Compatibility of BPEL Processes. February 2006.
7. T.Andrews, F.Curbera, H.Dholakia, Y.Goland, J.Klein, F.Leymann, K.Liu, D.Roller, D.Smith, S.Thatte, I.Trickovic, and S.Weerawarana. *BPEL4WS – Business Process Execution Language for Web Services.* Version 1.1, July 2002.
8. W.M.P. van der Aalst. Modeling and Analyzing Interorganizational Workflows. In *Proc. of CSD'98*. IEEE Computer Society Press, 1998.