Extending BPEL4WS for Multiple Instantiation

Jan Mendling, Mark Strembeck, Gustaf Neumann
Department of Information Systems, New Media Lab
Vienna University of Economics and BA
{firstname.lastname}@wu-wien.ac.at

Abstract: A means to define multiple activity instantiation is an important feature of a business process modelling language. In this paper we suggest to extend BPEL4WS with structured activities for multiple instantiation. In particular, we propose to extend BPEL4WS with a `collect` and a `broadcast` activity to model multiple instance behavior as well as `array` data structures to handle messages of multiple parties that act according to the same role.

1 Introduction

The Business Process Execution Language for Web Services (BPEL4WS) [ACD+03] is the de facto standard for XML-based business process modelling. Although it provides a rich set of primitives to specify Web Service compositions, it does not support multiple instantiation (see e.g. [WvdADtH03]). However, often activities need to be modelled which are executed multiple times within the same process instance without knowing the number of parallel executions a priori. This is especially the case for interorganizational business processes that often include one-to-many interactions. Typically, they can be divided into two parts (see e.g. [Ba98]) as the example of an online auction process illustrates:

1. One-to-many phase: A set of potential partners is created . In an auction process each bidder can be regarded as a potential business partner. The bidder with the best offer is chosen as the partner for further interaction.
2. Bilateral phase: The offerer and the auction winner continue the process in a bilateral way. The winner receives a bill, and the offerer initiates the shipment.

Online auctions are only one example of such one-to-many situations in business processes. The interaction between a teacher and multiple students or a public invitation to tender are further examples. The Business Process Modeling Notation (BPMN) [Wh04] provides dedicated control flow elements to model multiple instantiation. However, such language constructs are not included in BPEL4WS. Although corresponding work-arounds exist (see e.g. [WvdADtH03]), they are complicated. Nevertheless, the bilateral interaction in the second phase of an interorganizational process can be modelled with BPEL4WS in a straight forward manner.

2 BPEL4WS and Multiple Instantiation

BPEL4WS is an XML-based language for the definition of business processes based on Web Services. In this paper, we focus on activities, data flow, and partners in BPEL4WS processes. For a complete discussion of BPEL4WS see [ACD+03].

In essence, BPEL4WS defines conversational relationships between two parties via a so-called `partnerLink` that links one internal party to one corresponding external party. Process related data, e.g. message content, is stored in `variables`. Message exchange and logical operations are modelled via *basic activities* including the `receive` activity to receive a message; `invoke` for invocation of remote Web Service activities; or `assign` for data operations on `variables`. The control flow logic of a BPEL4WS process is defined through *structured activities*: `flow` for among others parallel execution; `sequence` for sequential execution; `switch` for branching related to a calculated value; `while` for loops; `compensate` for compensation actions; and `pick` for branching in response to an event or receipt of different messages.

Listing 1

```
<process A>
   <while condition="C1">
      <invoke "process B" ...
      </invoke>
   </while>
</process>
```

Listing 2

```
<process B>
   <receive "process A" ...
      createInstance="yes"
   </receive>
</process>
```

Listing 3

```
moreInstances:=true
i:=0
<while condition="moreInstances OR i>0">
   <pick>
      <onMessage "StartNewActivity A">
         <invoke "A" ... />
         <assign "i:=i+1"/>
      </onMessage>
      <onMessage "ActivityFinished A">
         <assign "i:=i-1"/>
      </onMessage>
      <onMessage "NoMoreInstances">
         <assign "moreInstance:=false"/>
      </onMessage>
   </pick>
</while>
```

Figure 1: Work-arounds for multiple instance modeling in BPEL4WS (abbreviated syntax).

Figure 1 illustrates two work-arounds for modeling multiple instances in BPEL4WS as presented in [WvdADtH03]. In listings 1 and 2, each of multiple instances in process A is executed via a spawned child process B. If the spawned instances must be synchronized, an approach as shown in listing 3 is needed. Here, the `pick` activity creates new instances and keeps track of the number of active instances. The loop exits if no more active instances exist ($i = 0$) and no new instances need to be created. In [Wh04] other work-arounds for multiple instance modelling in BPEL4WS are presented. Nevertheless, the lack of a native means to define structured activities for multiple instantiation in BPEL4WS makes modelling of processes like an online auction complicated. This missing feature might even prove to be a roadblock for BPEL4WS adaption.

In order to allow for multiple instantiation, the following issues have to be addressed:

- *Array of External Parties:* In a multiple instantiation activity different partners may act in the same role. The respective `partnerLink` should include an attribute to indicate such capabilities. This implies some correlation mechanism to identify messages of individual parties acting in the same role.

- *Array of Messages:* For data handling of multiple parties, we propose to extend BPEL4WS with `arrays` and array related operations. An `array` declaration should be similar to a `variable` declaration including an additional `partnerLink` reference in order to correlate messages stored in the array to its sender.

- *Structured Activities for Multiple Instantiation:* From our experiences two kinds of structured multiple instance activities are needed to extend BPEL4WS:

  1. An activity to model the receipt of multiple messages of different parties acting as the same `partnerRole`, as for example in an online auction where multiple parties act as bidders and send bid messages. We propose a new BPEL4WS activity called `collect` to address this need. It should support different synchronization conditions like time related conditions as duration, deadline, or maximum time-span between new instantiations; message content related conditions; or maximum amount of messages to be received.
  2. Often similar messages must be sent to a set of external parties who were identified via a previous `collect` activity. In the auction example, each bidder receives a notification after the auction. For this purpose, we propose to define a new BPEL4WS element called `broadcast`. It should rely on an `array` and it should be able to sort the listed messages similar to a `sort` statement in XSLT [Cl99]. Then, similar to a `switch` activity in BPEL4WS it should be able to specify different paths of processing.

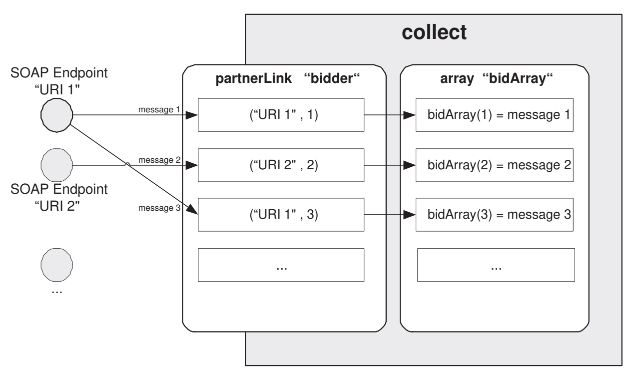

Figure 2: Data handling of collect activities.

Figure 2 illustrates how external parties and related messages can be administrated via `collect`. Each receipt of a message creates a new instance of activities nested in the corresponding `collect`. Moreover, the process engine has to store the following information:

- The message is stored in an associative array `bidArray` where n is the identifier of the current activity instance (`bidArray(n) = currentMessage`).
- The pair `(currentEndpoint,n)` is stored in the partnerLink `bidder` array.

Each individual activity instance can only access its "own" messages which are stored in an `array` at a certain position. For example an individual instance a works with `bidArray` like it was a `variable`. Actually, the variable represents the `array` at position a. This behavior is similar to data handling in FLOWer which is described in Workflow Data Pattern 4 in [RtHEvdA04]. The respective partnerLink manages an array of `(endpoint,n)` pairs that correlates the position of messages in an array to SOAP endpoints. Thus, each individual instance perceives only one endpoint associated to the partnerLink and one variable. As soon as a `collect` is completed, the whole array can be accessed by subsequent activities and from each array position, the corresponding endpoint can be identified. A downstream `broadcast` can then use the relationship between array positions and endpoints to send multiple messages to multiple parties. In a `compensate` activity or a fault handler, this mechanism can be exploited to undo the effects of a complete `collect` activity.

```
 1 <process auction>
 2    <partnerLink name="bidder" multiple="yes" .../>
 3    <array name="bidArray" messageType="bid" partnerLink="bidder"/>
 4    <sequence>
 5       <collect for="P7D"> <!-- XML Schema Simple Type Duration -->
 6          <receive partnerLink="bidder" variable="bidArray" .../>
 7          <invoke partnerLink="website" operation="newBid" .../>
 8       </collect>
 9       <broadcast array="bidArray">
10          <sort select="bid/price" order="descending">
11          <case condition="sort-position()=1">
12             <invoke partnerLink="bidder" operation="sendInvoice" .../>
13          </case>
14          <otherwise>
15             <invoke partnerLink="bidder" operation="sendReport" .../>
16          </otherwise>
17       </broadcast>
18    </sequence>
19 </process>
```

Figure 3: Listing of an auction process expressed with collect and broadcast.

Figure 3 shows a potential application of `collect` and `broadcast` structured activities in an auction process. The `partnerLink` is declared to be *multiple*. That means it must be able to handle multiple external partners playing the same role in parallel. The `array` includes a link to a `partnerLink` in order to correlate messages to its sender. When a process instance is created, `collect` gets activated for seven days (expressed as *P7D*, in conformance with XML Schema simple type *duration* [BLM$^+$01, BM01], see Line 5). It is able to receive multiple bids and for each of those it updates the auction web-site via an `invoke`. The bids are stored in an `array` named *bidArray*. After seven days the auction is closed and the `broadcast` activity is activated. It sorts the messages of the *bidArray* according to the XPath sort statement [CD99]. Different execution paths for the messages of the *bidArray* are specified with `case` and `otherwise` elements: the invoice is sent to the bidder who offered the highest price, other bidders receive a report of the auction.

3 Related Work

In [vdAtHKB03] four different control flow patterns addressing multiple instantiation are distinguished. The *Multiple Instantiation without Synchronization* pattern (Pattern 12) describes the spawning of activities that do not need to be synchronized later in the process. Patterns 13 to 15 describe multiple instantiation with synchronization. The general case (Pattern 15) captures multiple instantiation without a priori runtime knowledge about how many instances may be active. An analysis of BPEL4WS with respect to these workflow patterns is reported in [WvdADtH03]. In [vdAtH03] a Petri net based workflow language called YAWL is defined to capture the semantics of the above mentioned workflow patterns (excluding the implicit termination pattern). A different approach is presented in [MR03]: multiple instance patterns are modelled as so-called Reference nets which can be used to control arbitrary java applications. The analogy between multiple instances and single instance split and join control flow patterns is illustrated in [ZSY02] where Zhou et al. define five multiple instance patterns that correspond to branching patterns of single instance control flows reported in [vdAtHKB03]. The Business Process Modeling Notation (BPMN) [Wh04] also stresses the importance of multiple instantiation patterns by providing a dedicated syntax element. Furthermore, the `broadcast` activity presented in this paper can be related to communication Pattern 6 introduced in [RMB01]. Recent research into Workflow Data Patterns identifies three different ways to handle data of multiple instances [RtHEvdA04]. Yet, our approach addresses the handling of multiple parties in multiple activity instantiation.

4 Conclusion and Future Work

In this paper, we presented a concept to extend BPEL4WS with multiple instantiation. For this purpose, we proposed two additional structured activities, the `collect` and the `broadcast` activity, with related `partnerLink` and `array` elements. In our opinion, such modelling constructs are a prerequisite to provide for a comprehensive modelling of Web Service based business processes via BPEL4WS. One important goal in our future work is to implement a BPEL4WS process engine including the multiple instantiation extensions outlined in this paper.

References

[ACD+03] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., und Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1. Specification. BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems. 2003.

[Ba98] Bakos, Y.: The Emerging Role of Electronic Marketplaces on the Internet. *Communications of the ACM*. 41(8):35–42. 1998.

[BLM+01] Beech, D., Lawrence, S., Moloney, M., Mendelsohn, N., und Thompson, H. S.: XML Schema Part 1: Structures. W3C Recommendation 02 May. World Wide Web Consortium. 2001.

[BM01] Biron, P. V. und Malhorta, A.: XML Schema Part 2: Datatypes. W3C Recommendation 02 May. World Wide Web Consortium. 2001.

[CD99] Clark, J. und DeRose, S.: XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November. World Wide Web Consortium. 1999.

[Cl99] Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 November. World Wide Web Consortium. 1999.

[MR03] Moldt, D. und Rölke, H.: Pattern Based Workflow Design Using Reference Nets. In: *Proceedings of Business Process Management, International Conference, BPM 2003, LNCS 2678*. S. 246–260. 2003.

[RMB01] Ruh, W. A., Maginnis, F. X., und Brown, W. J.: *Enterprise Application Integration: A Wiley Tech Brief*. John Wiley and Sons, Inc. 2001.

[RtHEvdA04] Russella, N., ter Hofstede, A. H. M., Edmond, D., und van der Aalst, W. M. P.: Workflow data patterns. Technical report. Queensland University of Technology. 2004.

[vdAtH03] van der Aalst, W. M. P. und ter Hofstede, A. H. M.: YAWL: Yet Another Workflow Language (Revised Version). QUT Technical report, FIT-TR-2003-04. Queensland University of Technology. 2003.

[vdAtHKB03] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., und Barros, A. P.: Workflow Patterns. *Distributed and Parallel Databases*. 14(1):5–51. July 2003.

[Wh04] White, S. A.: Business Process Modeling Notation. BPMN 1.0, http://www.bpmn.org. Business Process Modeling Initiative. 2004.

[WvdADtH03] Wohed, P., van der Aalst, W. M. P., Dumas, M., und ter Hofstede, A. H. M.: Analysis of Web Service Composition Languages: The Case of BPEL4WS. In: *Proceedings of Conceptual Modeling - ER 2003, LNCS 2813*. S. 200–215. 2003.

[ZSY02] Zhou, J., Shi, M., und Ye, X.: On Pattern-based Modeling for Multiple Instances of Activities in Workflows. In: *Proc of the 1st International Workshop on Grid and Cooperative Computing, Hainan*. S. 723–736. 2002.