An Optimized Way for Mapping BPMN to BPEL

Atefeh Khalili Azimi, Islamic Azad University Shabestar Branch
at_khalili@yahoo.com

Paper Reference Number: 21
Name of the Presenter: Atefeh Khalili Azimi

Abstract
Business Process Management (BPM) is a way for building, maintaining and evolving large enterprise systems on the basis of business process models. A standard for representing such models is the Business Process Modeling Notation (BPMN). BPMN models are mainly intended for communication and decision-making between domain analysts, but often they are also given as input to software development projects. The main purpose of this paper is to present an optimized way for Business Process Modeling Notation to Business Process Execution Language for Web Service (BPEL4WS) mapping. BPEL is essentially an extension of imperative programming languages with constructs to Web service implementations. Because of Business Process complexity and extensibility there is no way except using new method of Business Process Management. Presenting a method for mapping automatically BPMN to BPEL can improve organization agility and flexibility. This method makes the understandable Business Process Models to become executable format. We find structures of BPMN models that can map directly to BPEL such as sequence, flow, while, etc. Then for identifying structure of the process graph and map to BPEL structures we use control flow patterns and replace equivalent BPEL codes. This is an automatic way for BPMN process graphs that mapping to BPEL. One of the main results of this work is to generate readable BPEL process.

Key words: BPMN, BPEL, Mapping, Business Process

1. Introduction

BPM is a notation based on flowcharting for business process modeling. It covers organization's activities to manage and improve their business processes. BPMN is a standard graphic notation for drawing and modeling business processes. These models are understandable by all business users from the business analysts to IT architects and developers and IT users. BPMN diagrams can be mapped to Business Process Execution Language (BPEL) processes to bridge the gap between business process design and implementation [6].

BPMN is a business process modeling notation and BPEL is an execution language. However there is high dependency between the two, but there is not a one-to-one mapping between them. There are some components in BPMN diagrams could not translate to execution language.

Translations from BPMN to BPEL must be to have the following key requirements:

1) Completeness, means that are applicable to any BPMN model
2) Automation, means capable of producing target code without requiring human intervention to identify patterns in the source model

3) Readability, producing target code that is understandable by humans[15]

In order to get these requirements, we represent an automatic way for mapping BPMN process graphs to BPEL processes.

The remainder of the report is structured as follows. Section 2 gives an overview of BPMN and BPEL. Section 3 presents an algorithm for translating BPMN into BPEL. Section 4 represents an algorithm results. Finally, Section 5 concludes and outlines future work.

2. Overview of BPMN and BPEL

2.1. BPMN

Business Process Modeling Notation (BPMN) is a graphical notation that describes the logic of steps in a business process. This notation has been especially designed to coordinate the order of processes and messages that pass between participants in different activities.

BPMN define the notation and semantics of Business Process Diagram (BPD). BPD is a diagram based on the flowchart technique, designed to present a graphical sequence of all the activities that take place during a process. It also includes relative information for making an analysis. In a BPD there are a series of graphical elements that are grouped into categories [2].

BPMN elements are classified into four categories: Flow objects that define the behavior of the processes. Flow objects may be an *event*, an *activity* or a *gateway*. Events affect the flow of the process and usually have a cause and a result. An event may cause the start of a process (*start event*), the end of a process (*end event*), the immediate termination of a process (*end terminate event*), a message that arrives or a specific time-date being reached during a process (*intermediate message/timer event*). Activities represent the work that is run as part of a business process. The activities may be compound or not. Gateways are routing constructs that used to control the divergence and convergence of the business process flow. There are: *parallel fork gateways* for creating concurrent sequence flows, *parallel join gateways* for synchronizing concurrent sequence flow, *data/event-based XOR decision gateways* for selecting one out of a set of mutually exclusive alternative sequence flows, and *XOR merge gateways* for joining a set of mutually exclusive alternative sequence flows into one sequence flow [2][15].Connection objects used to connect two objects in the process flow. These objects consist of three types: *Sequence Lines*, *Association* and *Message Lines*. SwimLanes used to organize flow activities in different visual categories which represent functional areas, roles or responsibilities. *Pools* and *Lanes* are two types of swimlanes. Artifacts used to provide additional information about the process. There are three types: *Data objects*, *Groups* and *Annotations* [2].

In the following figure we show a core subset of BPMN elements.

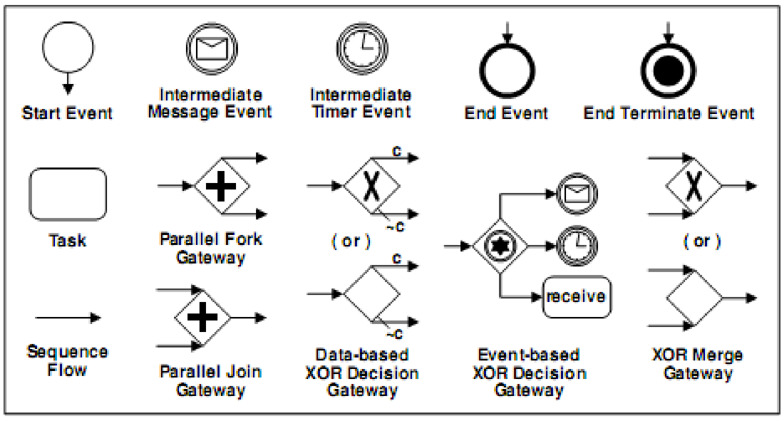


Fig 1: A core subset of BPMN elements.

2.2. BPEL

Business Process Execution Language (BPEL) is a standard executable language for specifying actions within business processes with web services. Processes in BPEL export and import information by using web service interfaces exclusively. The BPEL specification defines a number of activities that are the building blocks of algorithms, which it classifies as *basic* and *structured* types. *Basic* activities used to state management, communication, and exception handling, while *structured* activities define process control-flow [4].

Basic activities include atomic actions such as: *invoke*, for invoking an operation on a web service; *receive*, waiting for a message from a partner, *exit*, terminating the entire service instance, *empty*, doing nothing. To presentation of complex structures the following structured activities are defined: *sequence*, for defining an execution order, *flow*, for parallel routing, *switch*, for conditional routing, *pick*, for race conditions based on timing or external triggers, *while*, for structured looping, and *scope*, for grouping activities into blocks to which event, fault and compensation handlers may be attached.

An event handler is an event-action rule associated with a scope. It's enabled when the associated scope is under execution and may execute concurrently with the main activity of the scope. When an occurrence of the event associated with an enabled event handler is registered (and this may be a message receipt or a timeout), the body of the handler is executed. The completion of the scope as a whole is delayed until all active event handlers have completed [15].

The following code shows the sample code of a BPEL process. There is a very simple example that shows some of the concepts we just talked about:

```
<!—Hello World BPEL Process -->
<process name="HelloWorld"
         targetNamespace="http://samples.otn.com/helloworld"
         suppressJoinFailure="yes"
         xmlns:tns="http://samples.otn.com/helloworld"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/businessprocess/>
  <sequence>
    <! -- Receive input from requestor.
          Note: This map to operation defined in HelloWorld.wsdl -->
    <receive name="receiveInput" partnerLink="client"
                 portType="tns: HelloWorld"
                 operation="initiate" variable="input"
                 createInstance="yes"/>
    <! -- Generate content of output message based on the content of
         The input message. -->
```

```
      <assign>
          <copy>
             <from expression="concat ('Hello', bpws: getVariableData
('input', 'payload','/tns: name'))"/>
             <to variable="output" part="payload" query="/tns: result"/>
          </copy>
      </assign>
      <invoke name="replyOutput"
                partnerLink="client"
                portType="tns: HelloWorldCallback"
                operation="onResult"
                inputVariable="output"/>
   </sequence>
</process>
```

2.3. Differences between the Languages

There are some differences between BPMN and BPEL. First of all, BPMN and BPEL are used in a different stage in the life cycle of BPM. BPMN is used on designing and improving stage of the business process life cycle, while BPEL is used when implementing it. It's clear that different requirements exist in different phases. Second, BPMN is used by business analysts, and BPEL is used by technical analysts and programmers. They use different paradigms and focus on separate issues when modeling a process. BPEL is an execution language, which makes it too detailed and technical to be used by business analysts to design business processes [6].
So when transforming a BPMN model into an executable BPEL process, we have to consider these differences between the Languages.

2.4. Transformation strategies from BPMN models to BPEL processes

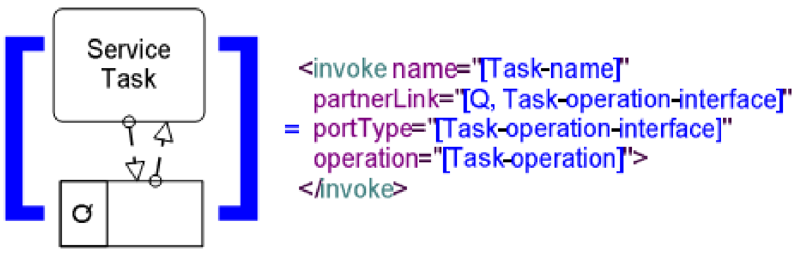
There are five transformation strategies for going from process graphs to executable BPEL processes, which will describe in this section. The first strategy is *Element-Preservation*. In this strategy the process graph needs to be acyclic. Because the BPEL Flow structured activity uses dead path elimination to synchronize parallel paths and dead path elimination works only for acyclic processes. The advantage of this strategy is that it is simple to implement and the resulting BPEL will be very similar to the original process graph since there is a one-to-one correspondence between the nodes. As a drawback, the resulting BPEL control flow includes more elements than actually needed: connectors are explicitly translated to empty activities in BPEL instead of join condition on nodes [13].
The second strategy is *Element-Minimization.* This strategy simplifies the generated BPEL processes of first strategy. The general idea is to remove the empty activities that have been generated from connectors and instead represent splitting behavior by transition conditions of links and joining behavior by join conditions of subsequent activities. The advantage of the resulting BPEL control flow is, at least to a greater extent than strategy 1. As a drawback, it is less intuitive to identify correspondences between the process graph and the generated BPEL control flow [13].
The third transformation strategy is *Structure-Identification.* In this strategy general idea is to identify structured activities in the process graph and apply mappings that are similar to the reduction rules. The advantage of this strategy is that all control flow is translated to structured activities. For understanding the resulting code this is the best strategy since it reveals the structured components of the process graph. As a drawback the relation to the original process graph might not be intuitive to identify [13].
The other strategy is *Structure-Maximization*. The general idea of this strategy is to apply the reduction rules of the Structure-Identification strategy as often as possible to identify a maximum of structure. The remaining process graph is then translated following the

Element-Preservation or Element-Minimization strategy. The advantage of this strategy is that it can be applied for arbitrary unstructured process graphs as long as its loops can be reduced via the reduction rules [13].
The last strategy is *Event-Condition-Action-Rules*. Similar to the Structure-Maximization strategy, the general idea of this strategy to apply the reduction rules of the Structure-Identification strategy as often as possible to identify a maximum of structure. The remaining process graph is then translated according to the Event-Condition-Action (ECA) rules approach. This approach derives a set of BPEL event handlers that a process calls on itself to capture unstructured control flow. The structured part of the process graph is then encapsulated within BPEL event handlers. Unstructured control flow maps to messages sent from some place in the process to itself where it is fetched by an event handler [11].

3. Mapping BPMN process graphs to BPEL processes

In this section we represent mapping BPMN models to BPEL processes. One of the main goals for mapping BPMN to BPEL is to generate readable BPEL code. As mentioned before BPEL is basically based on blocks and we can find in this language a mix of structured blocks and unstructured control-link and events. BPEL is much closer to a standard language such as Java than to a workflow notation such as BPMN which is graph-based.
Business analysts cannot use BPEL because it is not user friendly and it's hard to learn, read and implement and most of all it's hard to hide. In the real world, business analysts design their processes by using unstructured and parallel constructions like BPMN graphs.
Transformation from BPMN to BPEL is very hard to implement and produce correct readable code. There are some structures in BPMN that cannot be mapped to BPEL. So we can say that mapping of processes designed by a real business analyst to BPEL is very hard and difficult.
In this paper we represent an algorithm for mapping BPMN to BPEL. Our algorithm uses mix of transformation strategies that explain in section 2.4. It works like this:

1- In chapter 15 of the BPMN specification [14], a direct mapping from BPMN to BPEL is provided. For first step we try to find components of BPMN graph that can be map directly to BPEL. In this step we replace each component by its BPEL code. The following figure shows the sample of mapping.



```
<invoke name="[Task-name]"
  partnerLink="[Q, Task-operation-interface]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
</invoke>
```

Fig 2: mapping of a Service Task to BPEL

We use BizAgi software to create BPMN models. By use this software we can create XML output of model and then we use this file as an input of our mapping program. The following table represents BizAgi XML and equivalent BPEL code for *Timer Start Event* component.

Element	BizAgi Output	Equivalent BPEL Code

Type		
Timer Start Event	StartEventTrigger="Timer" TriggerTimerTimeCycle= "3 SS" TriggerTimerTimeDate= "03/09/2011" <ActivityName="StartT1">	<wait name="e-name" for="e-timecycle"/> <wait name="StartT1" for="3SS"/> or <wait name="e-name" until ="e-timeDate"/> <wait name="StartT1"until="3DD"/>

Table 1. BizAgi XML and equivalent BPEL code

In this stage, the entire BPMN XML file processed and all BPMN objects such as event, task, gateway and pool identified and then the order of objects in BPMN model specify.

2- We create two sets in this step, perd (o_i) provide the objects that precede o_i, and succ (o_i) provide the objects that follow o_i in BPD. For a given object o_i |perd (o_i)| indicate indegree of object o_i and |succ (o_i)| is outdegree of object o_i.

3- Workflow patterns are classified on different perspective, *control flow perspective* which describes activities and execution order through different constructs, *data perspective* that focus on business and processing data on control perspective, *resource perspective* and *Exception handling perspective*. At this stage we recognize pattern of BPMN models through control flow perspective and then replace each pattern with equivalent BPEL code. In BPMN models we can distinguish different control flow patterns such as *Basic Control-flow Patterns*, *Advanced Branching and Synchronization Patterns*, *Structural Patterns*, *Multiple Instances Patterns, State-based patterns* and *Cancellation Patterns*. The following figure shows Multiple Choice pattern in BPMN models.

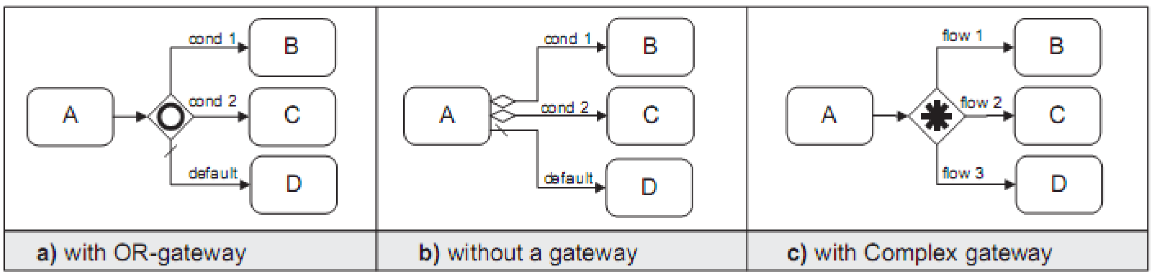


Fig 3: Multiple Choice Patterns in BPMN [22]

For example the following BPEL code is OR-gateway pattern's equivalent BPEL code.

```
 <sequence >
<flow name = "ncname" standard-attributes >
    <switch >
        <case condition="cond 1">
            <sequence name = "ncname">
                Target Activity B
            </sequence>
        </case>
        <case condition="cond 2">
            <sequence name = "ncname">
```

```
                Target Activity C
            </sequence>
        </case>
        <otherwise>
                <empty/>
        </otherwise>
    </switch>
</flow>
<switch >
        <case condition="">
            <sequence name = "ncname">
                Default Activity D
            </sequence>
        </case>
        <otherwise>
                <empty/>
        </otherwise>
    </switch>
</sequence>
```

4. Conclusions

In this paper, we presented an algorithm to translate BPMN models into BPEL. The BPEL process that generated with this translation algorithm is readable. In this algorithm we discover "patterns" in the BPMN models that can be translated onto BPEL processes and furthermore we can translate individual tasks and events to suitable BPEL code.
As mentioned the aim of this work is to generate readable BPEL process from BPMN models that can be used by developers. But this generated BPEL code can be modified and developed during business process implementation. So it would be desirable to have reversible transformations, which the modified BPEL codes can be viewed in BPMN models.

References

[1] Andrews.T, Curbera.F, Dholakia.H, Goland.Y, Klein.J, Leymann.F, Liu.K, Roller.D, Smith.D, Thatte.S, Trickovic.J, Weerawarana.S. (2003). *Business Process Execution Language for Web Services Version 1.1*
[2] BizAgi. (2009). *BizAgi: BizAgi*, available at http://www.bizagi.com
[3] Cardoso.J, Sheth.A, Miller.J, Arnold.J, Kochut.K. (2004). *Quality of service for workflows and web service processes*, Web Semantics: Science, Services and Agents on the World Wide Web 1(2004)281–308
[4] Chatterjee.S, Webber.J. (2003). *Developing Enterprise Web Services: An Architect's Guide*, Prentice Hall PTR
[5] Decker.G, Mendling.J. (2009). *Process instantiation*, University of Potsdam, Germany, Humboldt University, Germany
[6] Dikmans.L. (2008). *Transforming BPMN into BPEL: Why and How*
[7] Erl.T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR
[8] Gao.Y. (2006). *BPMN-BPEL Transformation and Round Trip Engineering*, Chief Architect and Director of Software Engineering, eClarus Software
[9] Khoshafian.S. (2007). *Service-Oriented Enterprises*, Auerbach Publications, Taylor & Francis Group

[10] Krafzig.D, Banke.K, Slama.D. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices*, Prentice Hall.
[11] Mendling.J, Lassen.K.B, Zdun.U. (2006). *On the Transformation of Control Flow between Block-Oriented and Graph-Oriented Process Modeling Languages,* Vienna University of Economics and Business Administration, Austria, University of Aarhus, Denmark
[12] Mendling.J, Lassen.K.B, Zdun.U. (2006). *Experiences in Enhancing Existing BPM Tools with BPEL Import and Export*, Vienna University of Economics and Business Administration, Austria, University of Aarhus, Denmark
[13] Mendling.J, Lassen.K.B, Zdun.U. (2006). *Transformation Strategies between Block-Oriented and Graph-Oriented Process Modeling Languages,* Vienna University of Economics and Business Administration, Austria, University of Aarhus, Denmark
[14] OMG. (2009). *Business Process Model and Notation (BPMN) Specification 2.0*
[15] Ouyang.C, M.P.van der Aalst.W. (2006). *Translating BPMN to BPEL,* Queensland University of Technology, Australia, Eindhoven University of Technology, Netherlands
[16] Ouyang.C, Dumas.M, Breutel.S, Hofstede.A. (2006). *Translating Standard Process Models to BPEL,* Queensland University of Technology, Australia
[17] Ouyang.C, Dumas.M, M.P.van der Aalst.W, H.M.ter Hofstede.A. (2006). *From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way,* Queensland University of Technology, Australia, and Eindhoven University of Technology, Netherlands
[18] Recker.J, Mendling.J. (2005). *On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages,* Queensland University of Technology, Australia, Vienna University of Economics and Business Administration, Austria
[19] Seese.D, Studer.R, Buchwald.H. (2009). *Modeling Workflow Patterns through a Control-flow perspective using BPMN and the BPM Modeler BizAgi,* Institute of Applied Informatics and Formal Description Methods University Karlsruhe (TH)
[20] White.S.A. (2005). *Using BPMN to Model a BPEL Process,* IBM Corp., United States
[21] White.S.A. (2005). *Process Modeling Notations and Workflow Patterns*, IBM Corp., United States
[22] Wohed.P, van der Aalst.W.M.P, Dumas.M, Hofstede.A.H.M, Russell.N (2006). *Pattern-based Analysis of BPMN an extensive evaluation of the Control-flow, the Data and the Resource Perspectives,* The Department of Computer and Systems Sciences, Stockholm University/KTH, Sweden, Faculty of Information Technology, Queensland University of Technology, Australia, Department of Technology Management, Eindhoven University of Technology, The Netherlands