Transforming UMM Business Collaboration Models to BPEL

Birgit Hofreiter and Christian Huemer

Department of Computer Science and Business Informatics
University of Vienna, Liebiggasse 4, 1010 Vienna, Austria
{birgit.hofreiter,christian.huemer}@univie.ac.at

Abstract. UN/CEFACT's Modeling Methodology (UMM) has been developed to analyze and design B2B business processes independent of the underlying exchange technology. It became the methodology of choice for developing ebXML business processes. Another technology for realizing B2B partnerships is Web Services. Currently, the business process execution languages (BPEL) seems to be the winner amongst the Web Services languages for orchestration and choreography. If Web Services is used as underlying exchange technology for B2B, the semantics of UMM business processes must be represented in BPEL. The goal of this paper is to verify whether BPEL is appropriate to capture UMM business collaborations or not. For this purpose we describe a transformation from UMM to BPEL.

1 Introduction

The OMG's Model Driven Architecture (MDA) is an effort to create applications by transforming models to executable software. MDA starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. The MDA Guide [9] identifies 4 very high level steps: (1) specifying a system independently of the platform that supports it, (2) specifying platforms, (3) choosing a particular platform for the system, and (4) transforming the system specification into one for a particular platform. The MDA approach should lead to a re-use of analysis & design artefacts as well as implementations. This should reduce the complexity of software development and result in lower costs. Furthermore, the approach of separating specification and implementation enables the portability of applications to existing and future technologies. Another goal of MDA is interoperability even across platforms. Strict methods guarantee that applications based on different technologies implement the same business logic.

In first place, the MDA approach focuses on the development of enterprise systems. However, we feel that there are no restrictions to follow a similar approach to interconnect systems of different enterprises in B2B e-Commerce. This is also in-line with the Open-edi reference model, which was developed in the electronic data interchange community and which became an ISO standard in 1997 [5]. Open-edi distinguishes a business operational view (BOV) and a functional service view (FSV). BOV related standards provide the tools for formal business description(s) of the external behavior of organizations, as seen by other organizations, in view of achieving a business goal.

These will address the rules for inter-organizational business processes, like operational conventions, agreements, or mutual obligations, as well as the semantics of business data exchanged. The FSV standards address the supporting services meeting the mechanistic needs of Open-edi. Consequently, FSV standards focus on the IT infrastructure for inter-organizational systems.

UN/CEFACT's Modeling Methodology (UMM) defines a development process for BOV standards on top of the UML [13]. It provides a UML profile for modeling B2B. In terms of the MDA, UMM fits into step (1) specifying a system independently of the platform. Candidate B2B platforms on the FSV and for MDA step (2) are typically implemented by the concepts of UN/EDIFACT, ebXML, and Web Services. In step (3) one has to select one of these standards for implementing the B2B system. The MDA requires a well-defined set of rules and techniques for mapping the platform independent model to the target technology in step (4). Thus, the definition of these rules and techniques is a critical task for the success of the MDA.

We are mainly interested in the choreography of business collaborations. Therefore, it is our goal to define the mapping from UMM business collaborations to the different choreography languages for B2B platforms. In other publications we concentrated on the mapping between UMM and ebXML choreography language BPSS [11]. A critical evaluation of these two standards is included in this conference proceedings [6]. Although UMM is not a mandatory part of ebXML, BPSS might be considered as "XML-ification" of parts of the UMM meta model. Owing to the close relationship between UMM and BPSS, the mapping rules are rather simple [4].

Currently, the adoption of Web Services standards by industry is rather quick. It seems that industry support for Web Services is much higher than for ebXML. Therefore, it is important to define a mapping between UMM and Web Services. The winner amongst the different choreography / orchestration languages seems to be the business process execution language (BPEL) [1], which combines WSFL [7] and XLANG [10]. This paper presents a first approach for a mapping from UMM version 12 [13] (that is based on UML 1.4) to BPEL 1.1. Before we go into the details of this transformation in Section 3, Section 2 briefly describes those UMM concepts that are relevant for the run-time systems.

2 UMM Concepts for Run-time Systems

The UMM meta model consists of 4 views in order to describe business collaboration models. Due to space limitations we will not go into the details of each view. We concentrate on the business transaction view (BTV) which captures most of the concepts relevant for a run-time system: business transaction and business collaboration protocol. The interested reader is referred to the UMM Meta Model [12] and the UMM User Guide [13], and our overview article [3].

A *business process* is defined as an organized group of related activities that together create customer value [2]. A *business collaboration* is a special type of a *business process* that involves two or more partners. A *business collaboration* is about aligning the information systems of the business partners, i.e. keeping all relevant business objects (e.g. purchase orders, line items, etc.) in an aligned state. If a business partner rec-

ognizes an event that changes the state of a business object, it initiates a communication to synchronize with the collaborating business partner. It follows that this communication is an atomic unit that leads to a synchronized state in both information systems. Hence, this special type of *business collaboration* is called a *business transaction*. It is the basic building block for more complex *business collaborations*. Although UMM is a top-down process, we start our explanation bottom-up from *business transactions* for educational purposes.

2.1 Business Transaction

The requirements of a business transaction are documented by a business transaction use case. Each business transaction use case results in exactly one business transaction that is represented by an activity graph. The activity graph of a business transaction is always built by exactly two business actions, a requesting business activity and a responding activity. Each business action is performed by exactly one authorized role executed by a business partner. The assignment of the business action to an authorized role is realized by the UML concept of swimlanes. The requesting business activity is assigned to the initiating role and the responding activity is assigned to the reacting role. The exchange of business information is shown by an object flow. One business action sets an information object flow state that is consumed by the other business action. An information object flow state refers to an information envelope exchanged between the business actions. Since we concentrate on the choreography in this paper, we do not further discuss the assembly of the content within the information envelope.

We distinguish two types of one-way transactions - information distribution and notification - as well as four types of two-way transactions - query/response, request/confirm, request/response, and commercial transaction. These six types cover all known legally binding interactions between two decision making applications as defined in Open-edi. The different types of business transaction patterns differ in the default values for the attributes that characterize *business actions*: *is authorization required, is non-repudiation required, time to perform, time to acknowledge receipt, and time to acknowledge acceptance*. The values for *is non-repudiation of receipt required* and for *retry count* are only defined for the *requesting business activity*. Most of these attributes are self-explanatory. An acknowledgment of receipt is usually sent after grammar val-

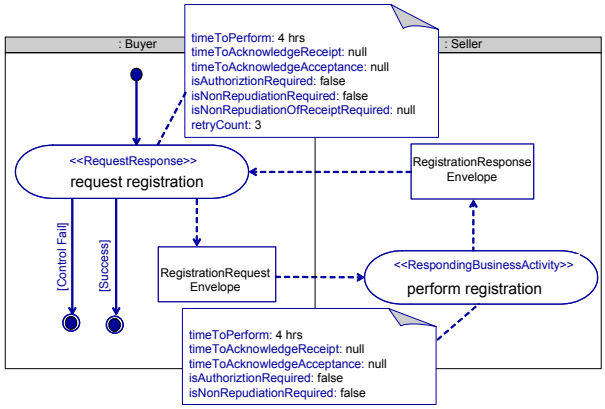

Fig. 1. Business Transaction "Register Customer"

idation, sequence validation, and schema validation. However, if the *is intelligible check required* flag is set to *false*, the acknowledgment is sent immediately after receipt without any validation. An acknowledgment of acceptance is sent after validating the content against additional rules to ensure that the content is processable by the target application. *Retry count* is the number of retries in case of time-outs.

Accordingly, a business transaction follows always a rather strict pattern. The activity graph of register customer depicted in Fig. 1 is a typical example of a two-way transaction. In case of a one-way transaction the responding object flow is omitted. A business expert provides the characteristic of register customer in a business transaction use case. The buyer starts a *request registration* activity. This activity creates a *registration request envelope* that triggers the *perform registration* activity of the seller. According to UMM business transaction semantics, *request registration* does not end after sending the envelope - it is still alive. The *perform registration* activity outputs the *registration response envelope*, which is returned to the *request registration* activity. It becomes obvious that the activity graph of a business transaction shows only the exchange of business information in the corresponding envelopes, but does not show any business signals for acknowledgements. The need for acknowledgements is documented in the tagged values only. Register customer does not need any acknowledgements.

2.2 Business collaboration protocol

The business transaction is the basic building block for more complex business collaborations. Consequently, a business collaboration covers a number of business transactions. It is important that the business collaboration defines an execution order for the business transactions, i.e. a business collaboration choreography. Each *business collaboration protocol use case* specifies the requirements for the choreography that is defined by an activity graph called *business collaboration protocol*. Currently, all activities of the *business collaboration protocol* must be stereotyped as *business transaction activity*. A *business transaction activity* must be refined by the activity graph of a *business transaction*. A *business collaboration protocol* might describe the choreography of multi-party collaborations. Each *business transaction activity* is executed by exactly two parties. Since a *business collaboration protocol* describes the contractual obligations between business partners and since most contracts are bilateral, a *business collaboration protocol* usually defines a binary collaboration.

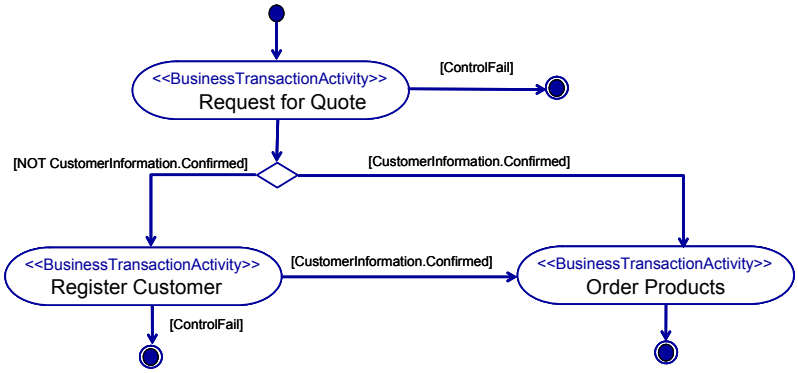

Fig. 2. Business Collaboration Protocol "Order Management"

The transition from one *business transaction activity* to another is triggered by two types of events: the completion of the previous business transaction and (in addition) the availability of a business object in a certain state. In addition to *business transaction activities* a business collaboration protocol includes the pseudo states used in UML activity diagrams: *initial state, final state, decisions* and *merge*, as well as *fork* and *join*.

Fig. 2 shows the business collaboration protocol for a very simple order management between a buyer and a seller. It is an over-simplified example showing only those aspects that are needed to understand the main concepts of a business collaboration protocol and, later, its transformation to BPEL. Thus, we have omitted any complex exceptional cases. The business collaboration starts with a *request for quote*. After the quote is made, i.e. the transaction is completed, a registered buyer can initiate *order products*. If the buyer is not registered, *register customer* is required prior to *order products*. Control failures of *request for quote* and *register customer* terminate the business collaboration. The register customer business transaction activity is refined by the activity graph in Fig. 1. The refinements of the other activities follow later in Fig. 3 and Fig. 4.

3 Transformation to BPEL

UMM defines the business semantics of a (usually binary) business collaboration without considering any specific exchange technology. In this section we concentrate on mapping the choreography of the business collaboration from UMM to BPEL. As the acronym indicates, BPEL defines executable business processes within a company. However, BPEL can be used for specifying so-called business protocols. A business protocol specifies the potential sequencing of messages exchanged by one particular partner with its other partners to achieve a business goal [8]. According to this definition, a business protocol is able to capture the B2B choreography defined in UMM. BPEL uses the concept of an abstract process to realize a business protocol. Abstract processes use all the concepts of BPEL but data handling might be more abstract handling only protocol-relevant data[1].

3.1 Transformation to WSDL

A business process defines how to coordinate the interactions with a business partner. In a Web Service environment this means a BPEL process definition coordinates activities representing the execution of Web Services. An activity is either receiving a message from a business partner via one's own Web Service or invoking a partner's Web Service. The interface of each Web Service is described by the means of WSDL. It follows that a BPEL process references Web Services interfaces defined in a WSDL file.

A UMM model defines which type of services are expected from which type of business partner. Therefore, a WSDL file—including port types and operations as well as messages used as input/output to the operations—can be derived from a UMM model. Each business partner of the collaboration results in its own port type. In our order management example a buyer collaborates with a seller. Hence, we create a *buyer port type* (lines 1 to 17) and a *seller port type* (lines 18 to 39).

In a UMM business transaction each partner performs exactly one activity. This activity sends and/or receives message envelopes. In case of asynchronous messaging (c.f.

```
[1]<portType name="BuyerPortType">
[2]    <operation name="ReceiveResponseForRequestRegistration">
[3]       <input message="RegistrationResponseEnvelope"/>
[4]    </operation>
[5]    <operation name="ReceiveResponseForPlaceOrder">
[6]       <input message="PurchaseOrderResponseEnvelope"/>
[7]    </operation>
[8]    <operation name="AckReceipt">
[9]       <input message="BusinessSignalAckReceipt"/>
[10]   </operation>
[11]   <operation name="AckAcceptance">
[12]      <input message="BusinessSignalAckAcceptance"/>
[13]   </operation>
[14]   <operation name="ControlFailure">
[15]      <input message="BusinessSignalControlFailure"/>
[16]   </operation>
[17]</portType>
```

```
[18]<portType name="SellerPortType">
[19]   <operation name="calculateQuote">
[20]      <input message="RequestForQuoteEnvelope"/>
[21]      <output message="QuoteEnvelope"/>
[22]   </operation>
[23]   <operation name="performRegistration">
[24]      <input message="RegistrationRequestEnvelope"/>
[25]   </operation>
[26]   <operation name="processOrder">
[27]      <input message="PurchaseOrderEnvelope"/>
[28]      <output message="PurchaseOrderResponseEnvelope"/>
[29]   </operation>
[30]   <operation name="AckReceipt">
[31]      <input message="BusinessSignalAckReceipt"/>
[32]   </operation>
[33]   <operation name="AckAcceptance">
[34]      <input message="BusinessSignalAckAcceptance"/>
[35]   </operation>
[36]   <operation name="ControlFailure">
[37]      <input message="BusinessSignalControlFailure"/>
[38]   </operation>
[39]</portType>
```

subsections 3.4 register customer and 3.5 order product) each activity that receives a message becomes an operation of the corresponding partner's port type. The operation is called exactly as the acitivity. In case of an requesting activity we add *receive response for...* at the beginning of the name. In case of synchronous messaging (c.f. subsection 3.3 request quote) only the responding activity becomes an operation. Additionally, each port type needs operations to receive acknowledgment of receipt, acknowledgement of acceptance and control failure messages (c.f. subsection 3.5).

3.2 Transformation of Partner Links

A BPEL file defines the exchanges with different partners - in case of a binary collaboration it is exactly one partner. The partner is connected to the process by a partner link type. A partner link type defines a relationship between two services and their roles within. Each role refers to a port type of the WSDL file. A binary collaboration requires exactly one partner link type (lines 40 to 43). Our example defines the partner link type *buyer seller link type* (line 40). The *buyer* role (line 41) refers to the *buyer port type* (line 1) and the *seller* role (line 42) to the *seller port type* (line 18).

```
[40]  <partnerLinkType name="BuyerSellerLinkType">
[41]     <role name="Buyer">   <portType name="BuyerPortType"/>    </role>
[42]     <role name="Seller">   <portType name="SellerPortType"/>    </role>
[43]  </partnerLinkType>
```

A UMM model describes a business collaboration between business partners from an overall point of view. BPEL describes a business process always from the point of view of a particular partner. Consequently, a UMM model results in multiple BPEL processes, one for each business partner involved. Of course these BPEL processes must be compliant to each other. *Partner links* define the relationships of the process owner with its partners. In the process definition for the buyer the *partner link* is defined in lines 44 to 46. The partner link (line 45) refers to the *buyer seller link type* (line 40). Since the buyer is the owner of the process, the *myRole* attribute refers to the *buyer* role (line 41) of the partner link type and the *partnerRole* attribute to the *seller* role (line 42). The partner link in lines 47 to 49 defines the pendant for the process owned by the seller.

```
[44]  <partnerLinks>
[45]     <partnerLink name="LinkToSeller"
                partnerLinkType="BuyerSellerLinkType"
                myRole="Buyer" partnerRole="Seller" />
[46]  </partnerLinks>
```

```
[47]  <partnerLinks>
[48]     <partnerLink name="LinkToBuyer"
                partnerLinkType="BuyerSellerLinkType"
                myRole="Seller" partnerRole="Buyer" />
[49]     </partnerLinks>
```

3.3 Transformation of Request For Quote (Synchronous)

In subsection 3.1 we already explained that the Web Services operations are derived from the requesting and responding activities of UMM business transactions. The basic activities in a BPEL process refer to these operations. The activity <*invoke*> is used to call an operation of a partner's port type and the activity <*receive*> for receiving messages from a partner via an operation of one's own port type. In addition, <*reply*> is used to specify a synchronous response following a <*receive*> activity.

In UMM, a business transaction is the basic building block for interactions between the business partners. Hence, we transfer each business transaction into a set of BPEL activities, that will be a building block for assembling a BPEL process (c.f. subsection 3.6). We demonstrate a mapping for the three business transactions of our example, each representing a different type of complexity. Fig. 3 depicts the *request for quote* transaction which is the one of the lowest complexity. It is stereotyped as *query/ response* business transaction. This means the responding business partner has the information available prior to the request. No acknowledgments are needed. Due to this characteristics the request for quote is realized by a synchronous interaction in BPEL.

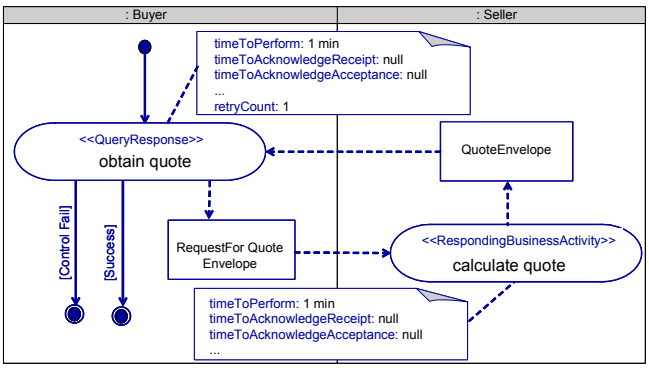

Fig. 3. Business Transaction "Request for Quote"

The synchronous interaction is realized by a single operation *calculate quote* on the *seller port type*. This operation expects a *request for quote envelope* as input and outputs a *quote envelope*. In the BPEL file for the buyer process the interaction is defined by the simple activity <*invoke*> (lines 50 to 53). It references the partnership with the seller defined in the *partner link* named *link to seller* (line 45). Furthermore, <*invoke*> references the *calculate quote* operation (line 19) of the *seller port type* (lines 18 to 39). The *inputVariable* and the *outputVariable* are used for data handling of sent and received messages in executable processes. They are optional in abstract processes. Nevertheless, we include them to hint on the type of input/output message. Additionally, the presence of an *outputVariable* signifies a synchronous invocation, because an asynchronous invocation needs an input but no output. The <*source*> statements in line 51 and 52 are explained later in subsection 3.6.

```
[50] <invoke partnerLink="LinkToSeller" portType="SellerPortType" operation="calculateQuote"
            inputVariable="SentRequestForQuote" outputVariable="ReceivedQuote">
[51]   <source linkName="RequestForQuote2RegisterCustomer"
          transitionCondition="bpws:getVariableData('CustomerInformationConfimed') = 0"/> <!--ref line 118 -->
[52]   <source linkName="RequestForQuote2OrderProduct"
          transitionCondition="bpws:getVariableData('CustomerInformationConfimed') = 1"/> <!--ref line 119 -->
[53] </invoke>
```

The BPEL file for the seller's process includes the activity <sequence> (line 54) that is built by a <*receive*> activity (line 55) and a <*reply*> activity (line 56). Both included activities must reference the same operation: *calculate quote* (line 19). Consequently, the values for *partnerLink*, *portType* and *operation* must be the same in both cases (c.f. line 55 and 56).

```
[54] <sequence>
[55]   <receive partnerLink="LinkToBuyer" portType="SellerPortType" operation="calculateQuote"
                variable="ReceivedRequestForQuote"/>
[56]   <reply partnerLink="LinkToBuyer" portType="SellerPortType" operation="calculateQuote"
                variable="SentQuote"/>
[57] </sequence>
```

3.4 Transformation of Register Customer (Asynchronous)

Register customer is the next business transaction we consider. We already used *register customer* (Fig. 1) to illustrate UMM business transactions in section 2.1. It is stereotyped as *request/response* transaction. This means a decision process at seller side is stimulated by the request and must be finished before a response. Therefore, the request and the response are defined as two asynchronous interchanges. *Register customer* does not need any acknowledgements. According to the retry count, the buyer has to restart the transaction three times if he does not receive an response. If the buyer does not succeed after three trials, UMM requires him to send a control failure message.

Owing to space limitations we only illustrate the buyer's process in lines 58 to 85. Note, in this code we do not show partner links and message variables anymore. An <*invoke*> activity (line 69) is used to call the *perform registration* operation (line 23). Next the buyer expects to receive a message via its *receive response for request registration* operation. We use a <*pick*> activity that awaits the occurrence of one of a set of events and then performs the activity associated with the event that occurred. The first

```
[58]   <sequence>
[59]     <target linkName="RequestForQuote2RegisterCustomer"/> <!-- references line 118 -->
[60]     <source linkName="RegisterCustomer2OrderProduct"/> <!-- references line 120 -->
[61]     <assign>
[62]       <copy>
[63]         <from expression="3"/>
[64]         <to variable="PerformRegistrationRetryCount"/>
[65]       </copy>
[66]     </assign>
[67]     <while condition="bpws:getVariableData('PerformRegistrationRetryCount') &gt; 0 AND
                           bpws:getVariableData('PerformRegistrationRetryCount') = NULL">
[68]       <sequence>
[69]         <invoke ... portType="SellerPortType"operation="performRegistration" ... />
[70]         <pick>
[71]           <onMessage ... portType="BuyerPortType" operation="ReceiveResponseForRequestRegistration"
                           variable="ReceivedRegistrationResponse">
[72]             <empty/>
[73]           </onMessage>
[74]           <onAlarm for="PT4H">
[75]             <assign> <!-- decrement Perform RegistrationRetryCount --> </assign>
[76]           </onAlarm>
[77]         </pick>
[78]       </sequence>
[79]     </while>
[80]     <switch>
[81]       <case condition="bpws:getVariableData('PerformRegistrationRetryCount') = 0">
[82]         <throw faultName="RequestForQuoteControlFail"/>
[83]       </case>
[84]     </switch>
[85]   </sequence>
```

event is given in the <*onMessage*> element (line 71) receiving a message from the *receive response for request registration* (line 2). The other event is stated in the <*onAlarm*> element (line 74) which is effective if a time frame of 4 hours is exceeded. In this case one retry is consumed. Hence, the *perform registration retry count* variable - which is initiated at the beginning of the transaction in lines 61 to 66 - must be decremented (line 75). The <*while*> loop (lines 67 to 79) continues if no message was received and the maimum number of retrys is not reached. If the while loop is stopped because no retrys are left, the condition of the <*case*> statement (line 81) in the <*switch*> (lines 80 to 84) holds, and a control failure exception is thrown (line 82).

3.5 Transformation of Order Product (with Acks)

Finally, we take a look at the *order product* business transaction, which is the most complex one. It is stereotyped as *commercial transaction*. In UMM this means that the business transaction results in a residual obligation between the business partners to fulfill terms of a contract. A commercial transaction requires acknowledgments to be sent. These acknowledgements are not the ones on the network level. An acknowledgment of receipt is sent after grammar validation, sequence validation, and schema validation. An acknowledgment of acceptance is sent after validating the content against additional rules to ensure that the content is processable by the target application.

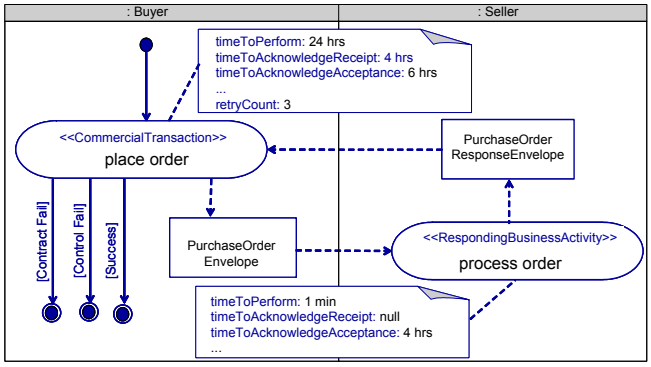

Fig. 4. Business Transaction "Order Product"

The code in lines to 112 represent the order product business transaction in the buyer's process. We do not detail the transaction for the seller's process. The idea of this code pattern is similar to the one used in the previous subsection. This time the request results in three return messages, two acknowledgements and the response. A control failure must be raised, if any one of these is not received in the permitted time frame. As a consequence, we recursively nest <*pick*> elements (lines 95 and 97) within the <*onMessage*> elements of the previously "picked" message.

```
[86] <sequence joinCondition="bpws:getLinkStatus('RequestForQuote2OrderProduct')
                              OR bpws:getLinkStatus('RegisterCustomer2OrderProduct')">
[87]     <target linkName="RequestForQuote2OrderProduct"/> <!-- references line 119 -->
[88]     <target linkName="RegisterCustomer2OrderProduct"/> <!-- references line 120 -->
[89]     <assign> <!-- initiate OrderProductRetryCount --> </assign>
[90]     <while condition="bpws:getVariableData('OrderProductRetryCount') &gt; 0 AND
                           bpws:getVariableData('ReceivedPurchaseOrderResponse') = NULL">
[91]       <sequence>
[92]         <invoke partnerLink="LinkToSeller" portType="SellerPortType"
                     operation="processOrder" inputVariable="SentPurchaseOrder" />
```

```
[93]              <pick>
[94]                <onMessage ... portType="BuyerPortType" operation="AckReceipt" ... >
[95]                  <pick>
[96]                    <onMessage ... portType="BuyerPortType" operation="AckAcceptance" ...>
[97]                      <pick>
[98]                        <onMessage ... portType="BuyerPortType" operation="ReceiveResponseForPlaceOrder">
[99]                          <invoke ...  portType="SellerPortType"  operation="AckAcceptance" ... />
[100]                       </onMessage>
[101]                       <onAlarm for="PT18H"> <!-- decrement OrderProductRetryCount --> </onAlarm>
[102]                     </pick>
[103]                   </onMessage>
[104]                   <onAlarm for="PT2H"> <!-- decrement OrderProductRetryCount --></onAlarm>
[105]                 </pick>
[106]               </onMessage>
[107]               <onAlarm for="PT4H"> <!-- decrement OrderProductRetryCount --> </onAlarm>
[108]           </pick>
[109]         </sequence>
[110]     </while>
[111]     <switch> <!-- Throw exception if OrderProductRetryCount = 0 --> </switch>
[112]   </sequence>
```

3.6 Transformation of Business Collaboration Protocol

A UMM business collaboration protocol choreographs the business transactions of the business collaboration. So far we detailed the choreography within a business transaction. Now we define the flow between the business transactions in order to finalize the BPEL process. For this purpose BPEL provides the structured activity called flow. A flow is a group of activities. It completes when all of the activities in the flow have completed. Completion of an activity in a flow includes the possibility that it will be skipped if its enabling condition turns out to be false [1]. Additionally, the flow specifies synchronization dependencies between activities. A link element is used to express a transition from one activity to the other. The link has a unique name within the process. The activity that is the source for the link includes a *source* element that references the link by name. Similarly, the target activity includes a target element that references the corresponding link by name. Furthermore, the *source* element may specify a transition condition. If an activity includes more than one *target* element, it is useful to specify a join condition for its activation. By default the status of at least one incoming link has to be positive to start an activity.

Code lines 113 to 133 represent the buyer's process of the *order management* collaboration shown in Fig. 2. After a starting <*process*> element the partner link to the seller (copy of lines 44 to 46) and a definition of all variables follow. Next, the main part groups all activities of the business collaboration into a <*flow*> element (lines 116 to 125). All sub-processes representing the business transactions *request for quote* (lines 50 to 53), *register customer* (lines 58 to 85) and *order product* (lines to 112) are defined as part of the flow.

Prior to these sub-processes all the transitions between the transactions are defined. According to Fig. 2, three synchronization dependencies exist: (1) from *request for quote* to *register customer* (line 118), (2) from *request for quote* to *order product* (line 119), and (3) from *register customer* to *order product* (line 120). The first two transitions start from the request for quote sub-process (lines 50 to 53). Consequently, the <*invoke*> activity covering the whole sub-process includes two <*source*> elements (lines 51 and 52) referencing the two links. Since the two links are mutually exclusive depending on the status of the customer information, corresponding transition condi-

tions are included in the <*source*> elements. The third transition starts from the *register customer* sub-process that includes a related <*source*> element (line 60). Since this activity is also the destination of the first transition, it includes a corresponding <*target*> element in line 59. Both the second and the third transition end at the *order product* sub-process, which is marked by the <*target*> elements in lines 87 and 88. If any of the two transitions becomes active, order product should start. Therefore, the join condition of the <*sequence*> element (line) is a Boolean *OR*.

```
[113]<process>
[114]   <!-- insert partner links; lines 44 - 46 -->
[115]   <!-- insert varaible definitions, not shown in this paper -->
[116]   <flow>
[117]      <links>
[118]         <link name="RequestForQuote2RegisterCustomer"/>
[119]         <link name="RequestForQuote2OrderProduct"/>
[120]         <link name="RegisterCustomer2OrderProduct"/>
[121]      </links>
[122]      <!-- insert invoke statement of Transaction Request For Quote lines 50 - 53 -->
[123]      <!-- insert sequence statement of Transaction Register Customer lines 58 - 85 -->
[124]      <!-- insert sequence statement of Transaction Order Product lines 84 - 108 -->
[125]   </flow>
[126]   <faultHandlers>
[127]      <catch faultName="RequestForQuoteControlFail">
[128]         <invoke ... portType="SellerPortType" operation="ControlFailure" ... />
[129]         <terminate/>
[130]      </catch>
[131]      <catch faultName="OrderProductControlFail"> ... </catch>
[132]   </faultHandlers>
[133]</process>
```

The flow automatically starts with the *request for quote* sub-process, because it does not include an incoming synchronization dependency. The process automatically ends after the *order product* sub-process, because all other activities were active before. Note, *register customer* might be skipped according to the transition condition. The control failures thrown in lines 82 and 111 also terminate the process. These control failures are subject to the fault handlers in lines 126 to 132.

4 Summary

UN/CEFACT's modeling methodology (UMM) is a UML-based methodology for capturing the business semantics according to the BOV of Open-edi. ebXML and Web Services are to popular technologies for implementing B2B processes on the FSV of Open-edi. The Open-edi concept is very similar to the MDA - it separates the specification from the implementation. This approach guarantees reusability, portability, and interoperability. A key success factor is a well-defined set of rules and techniques to map from the UMM model to the different B2B implementation technologies. Our current research is directed towards the B2B choreography. In our previous work we concentrated on the mapping from UMM to ebXML's BPSS. Since BPSS is based on the UMM meta model the mapping is comparatively easy. Owing to the growing popularity of Web Services, BPEL is gaining more and more acceptance by industry. Hence, there is a desire to transform UMM models to BPEL processes. This paper presents a first evaluation to what extent UMM models can be transformed to BPEL. Future work will include a detailed analysis on the strengths and limitations of both BPSS and BPEL to express business collaborations.

In this paper we show how the UMM artefacts business transaction and business collaboration protocol are represented in BPEL. Whereas a transformation from UMM to BPSS results in a single process definition for all partners, a transformation to BPEL results in a different process definitions for each partner. BPSS is able to capture UMM's security requirements on the activities and exchanged messages. BPEL is not. Instead other Web Services standards must capture the security semantics. Nevertheless, we are able to represent all the UMM choreography information in BPEL. We demonstrated the transformation by means of a simple example. Since UMM business transactions have a very strict structure, the resulting transformations provide useful patterns. Furthermore, the transformation of the business process protocol follows a clear approach based on grouping all business transaction activities to BPEL flow structure and transforming the UMM transition to BPEL link elements.

References

1. Andrews, T., Curbera, F., Dholakia, H., Goland Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.; Business Process Execution Language for Web Services, Version 1.1, May 2003 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbizspec/html/bpel1-1.asp
2. Hammer, M., Champy, J.; Reengineering the Corporation: Manifesto for Business Revolution; Harper Business; 1993
3. Hofreiter, B., Huemer, C.; Modeling Business Collaborations in Context; Proceedings of On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops; Springer LNCS; November 2003
4. Hofreiter, B., Huemer, C.; ebXML Business Processes -Defined both in UMM and BPSS; Proceedings of the 1st GI Workshop on XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany, March 2004 http://wi.wu-wien.ac.at/~mendling/XML4BPM/xml4bpm-2004-proceedings-bpss.pdf
5. ISO; Open-edi Reference Model. ISO/IEC JTC 1/SC30 ISO Standard 14662. ISO; 1997
6. Kim, J.-H.., Huemer, C.; Analysis, Transformation and Improvements of ebXML Choreographies based on Workflow Patterns; Proceedings of On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE; Springer LNCS; October 2004
7. Leymann, F.; Web Services Flow Language (WSFL 1.0); May 2001 http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
8. Leymann, F., Roller, D.; Modeling Business Processes with BPEL4WS; Proceedings of the 1st GI Workshop on XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany, March 2004 http://wi.wu-wien.ac.at/~mendling/XML4BPM/xml4bpm-2004-proceedings-bpel4ws.pdf
9. Miller, J., Mukerji, J.; MDA Guide Version 1.0.1; OMG omg/2003-06-01, http://www.omg.org/docs/omg/03-06-01.pdf
10. Thatte, S.; XLANG - Web Services for Business Process Design; June 2001; http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
11. UN/CEFACT; ebXML - Business Process Specification Schema v1.10; October 2003; http://www.untmg.org/downloads/General/approved/ebBPSS-v1pt10.zip
12. UN/CEFACT; UMM Meta Model, Revision 12; January 2003; http://www.untmg.org/downloads/General/approved/UMM-MM-V20030117.zip
13. UN/CEFACT; UMM User Guide, Revision 12; September 2003; http://www.untmg.org/downloads/General/approved/UMM-UG-V20030922.zip