Deriving executable BPEL from UMM Business Transactions

Birgit Hofreiter[1], Christian Huemer[2], Philipp Liegl[3], Rainer Schuster[3] and Marco Zapletal[2]

[1]University of Technology Sydney, Australia
[2]Institute of Software Technology, Vienna University of Technology, Austria
[3]Research Studios Austria, Austrian Research Centers GmbH - ARC
birgith@it.uts.edu.au, huemer@big.tuwien.ac.at
marco@ec.tuwien.ac.at, {pliegl, rschuster}@researchstudio.at

Abstract

UN/CEFACT's Modeling Methodology (UMM) is a UML profile for modeling global B2B choreographies. The basic building blocks of UMM are business transactions, which describe the exchange of a business document and an optional response. In addition to these business document exchanges, UMM business transactions mandate business signals that acknowledge the correctness of business documents. It is expected that a business service interface (BSI) on each business partner's side reacts on incoming messages and on messages expected but not received. However the internal orchestration of the BSI is open to interpretations. In this paper we demonstrate an unambiguous mapping from global choreographies described by UMM transactions to a BPEL-based orchestration of the business service interface. It becomes obvious that rather simple looking UMM transactions lead to a more complex message exchange mechanism when implemented on top of Web Services.

1 Motivation

Service-oriented computing is considered as an enabler of inter-organizational systems overcoming the limitations of traditional electronic data interchange (EDI) standards [2]. It results in a shift towards the services exchanging the business document types and towards specifying the flow of these services. The flow of services is described by the concepts of orchestration and choreography [11]. An orchestration defines the sequence and conditions in which one service invokes other services to realize some goal. The interactions between the services are described at message level, including the business logic and the execution order of the interactions. Choreography is used to define the flow of message exchanges in peer-to-peer systems like inter-organizational systems. Thereby we distinguish local choreography from a public choreography. A global choreography describes the interaction between the business partners from a global and neutral perspective. It follows that a single process description is used for all partners. A local choreography describes the interactions between the business partners from the perspective of a certain business partner. It follows, a different process description is created for each business partner. Interoperability requires that these process descriptions are complementary, e.g. if one business partner invokes a service the other business partner must receive the call of the same service.

A global choreography has the potential to achieve an agreement between the partners. Local choreographies derived from the global UMM model enable the configuration of each partners system. Thus, business experts have a greater interest in the global choreography, whereas software engineers are purely interested in the local choreographies. The United Nation's Centre of Trade Facilitation and e-Business (UN/CEFACT) became known for maintaining the UN/EDIFACT standards in first place. However, their mission is rather to develop trade procedures than to develop IT-platform specific solutions. Consequently, UN/CEFACT has started to standardize business scenarios independent of the IT platform. Core Components are business document building blocks specified independent of any transfer syntax realizing the exchange. UN/CEFACT's Modeling Methodology enables capturing business knowledge independent of the underlying implementation technology in order to model the choreography and data exchange commitments to be agreed between partners.

In order to implement a B2B system the UMM models must be transformed into platform specific solutions. A candidate platform are Web Services. However, Web Services have a general infrastructure purpose and are not specifically dedicated to e-business. As a consequence con-

2007 IEEE International Conference on Services Computing (SCC 2007)
0-7695-2925-9/07 $25.00 © 2007 IEEE

cepts implicitly interwoven into UMM must be explicitly revealed when implementing the system by Web Services.

In this paper we concentrate on UMM business transactions which are the basic building blocks in the global UMM choreography. In UMM, a business transaction involves not only the exchange of business documents, but also the exchange of business signals for acknowledging the correctness of documents. These business signals must not be mixed up with acknowledgments in reliable messaging, since business signals confirm/disconfirm the appropriateness of the business content rather than the receipt of a message. On each partner's side a business service interface is responsible for the correct handling of business documents and signals. If this business service interface is implemented by means of Web Services, it must explicitly invoke and provide services for the correct message handling. The correct orchestration of the business service interface is then best described by BPEL. In this paper we present a mapping from the global UMM transactions to BPEL 2.0 [9] orchestrations representing the local business service interfaces on each partner's side.

2 UMM Business Transactions

UN/CEFACT's Modeling Methodology (UMM) is a holistic approach for modeling the collaborative space between enterprises. It provides a graphical modeling language that is defined as a UML profile - i.e., as a set of stereotypes, tagged values and constraints [15] - which we co-authored. It customizes the rather general UML to the specific needs of B2B. The goal of the UMM is to define business collaborations between two or more business partners. A UML business collaboration model consists of three views - the business domain view (BDV), the business requirements view (BRV) and the business transaction view (BTV). In the business domain view existing business processes and business domain knowledge is captured. The BRV gathers the requirements of the to-be designed business collaborations. Finally, in the BTV the business process analyst designs the flow of the business collaboration based on the requirements collected before. In this paper, we concentrate on the business transaction view. A business collaboration is modeled by means of a business collaboration protocol, which is based on a UML activity diagram. A business collaboration protocol choreographs a flow of business transaction. A business transaction describes the exchange of a business document and an optional response between exactly two participants.

A business transaction is responsible for aligning the business information systems of the collaborating business partners. Aligning the business information systems means that all business entities (e.g. purchase order, line items, etc.) are in the same state in each information system. In case of a state change of a business entity, a business transaction is initiated to synchronize with the collaborating business partner. It follows, that a business transaction is an atomic unit responsible for the synchronization between exactly two business partners' information systems.

Basically there are two different types of business transactions. The first one describes the synchronization process in a uni-directional way. In this case the business information only flows from the initiating business partner to the responding business partner. After the initiating business partner reports an already effective and irreversible state change, the reacting business partner has to accept it without giving a response back that could affect the final state. Thus, this type of business transaction is called one-way business transaction (e.g. the notification of shipment, the update of a product in a catalog, etc.). In contrast, the second type provides the possibility for the reacting business partner to change the final state by sending business information back to the initiating business partner. Having a deeper look at this scenario, the initiating business partner sets the business entity to an interim state and the final and irreversible state is decided by the responding business partner. This type of business transaction is called two-way business transaction (e.g., request for quote, search for products, etc.). Figure 1 shows an example of a two-way business transaction dealing with a request for a quote.

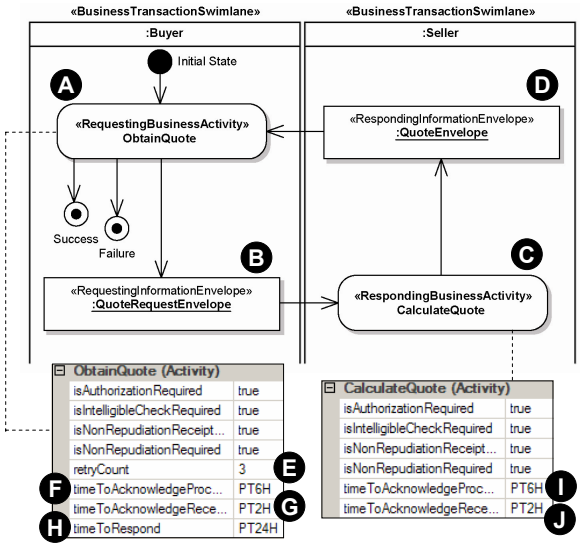

Figure 1. Business Transaction in UMM

In UMM, a business transaction is represented as a UML activity graph following always the same pattern. UN/CEFACT strictly defines this pattern by specifying a certain set of stereotypes a modeler must use in order to create a UMM compliant business transaction. The activity graph of a *business transaction* consists of exactly two *business*

transaction swimlanes each of them representing an *authorized role* performed by a business partner. The initiating business partner performs exactly one *requesting business activity* and the responding business partner performs exactly one *responding business activity*. While the object flow from the *requesting* to the *responding business activity* is mandatory, the object flow vice versa is optional. The exchanged business information is represented by a *requesting* or a *responding information envelope*.

In respect to special business needs, UMM distinguishes two types of one-way business transactions and four types of two-way business transactions. Regarding the distinction of a one-way business transaction: if the business information sent is a formal non-repudiable notification, the transaction is called *notification*. Otherwise the business transaction is called *information distribution*. Regarding the distinction of a two-way business transaction: the transaction is called *query/response*, if the responder already is able to provide the information beforehand. If the responder does not have the information, but no pre-editor context validation is required before processing, the transaction is a *request/confirm* one. If the latter is required and the transaction results in a residual obligation between the business partners to fulfill terms of a contract, it is called a *commercial transaction*. Otherwise the transaction is called *request/response*. The mentioned types of business transactions in this paragraph cover all known legally binding interactions between two decision making applications as defined in Open-edi reference model [6]. They differ in the default values of the tagged values characterizing a *requesting/responding business activity*: *is authorization required*, *is non-repudiation required*, *time to acknowledge receipt*, *time to acknowledge acceptance*, and *is non-repudiation of receipt required*. In addition, only the requesting business activity specifies the *time to respond* and a *retry count*. Because of their nomenclature, these tagged values are considered to be self-explanatory. Furthermore the business transaction types and their tagged values have proven to be useful in RosettaNet [12].

Figure 1 shows an example of a two-way business transaction, which serves as the basis for our proposed mapping to executable BPEL processes. In this example, the `Buyer` performs the requesting role and requests a business document - a `QuoteEnvelope` - from the `Seller` who takes on the reacting role. The buyer starts an `ObtainQuote` activity and creates a requesting business document - the `QuoteRequestEnvelope` - that triggers the `CalculateQuote` activity of the `Seller`. Having reached this point of the business transaction, the `ObtainQuote` activity is still alive, because it is waiting for a response from the `Seller`. Afterwards, the `CalculateQuote` activity generates a *responding information envelope* - the `QuoteEnvelope` - that is returned to the `Buyer`. Finally, the `ObtainQuote` activity on the `Buyer's` side processes the response. If a quote was provided in the `QuoteEnvelope` the business transaction is considered as successful. Otherwise, if no quote was provided, the business transaction resulted in a failure. The modeling of the exchanged business documents plays a major role in our approach and is described in more detail in section 3.1. If one of the participants encounters a time-out - i.e., an acknowledgement or the `QuoteEnvelope` is not received in time - he must signalize his partner a *time-out exception*. If a message is scrambled during the exchange and hence is not processible, a *failed business control exception* must be issued. Similarly, a *failed business control exception* is sent if the *retry count* is elapsed and all attempts to execute the business transaction failed.

3 Mapping Business Transactions to BPEL

For the business transaction depicted in figure 1 we identify four participating entities namely the buyer, the buyer's business application, the seller and the seller's business application. For each of the entities a WSDL is generated. On the buyer's and the seller's side a BPEL process orchestrates the different service invocations. In the following section we will first show how core components are used to derive XML schemas for business documents and then elaborate the generation of WSDL 1.1 [18] and BPEL 2.0 [9] artifacts.

3.1 Generating Business Documents

In UMM the so called *business information view* is used to model the business documents exchanged during a business transaction. Figure 2 shows an example for a business information view depicting the `QuoteEnvelope` exchanged in the business transaction in figure 1. The `QuoteEnvelope` is of type *information envelope* and contains two *information entities* namely `StandardBusinessDocumentHeader` and `QuoteDocument`. The *header* of the *information envelope* contains auxiliary information and the *body* holds the actual workload information. In order to model the actual information in the *body* we use the concept of so-called core components [17]. Core components are reusable building blocks to assemble business documents. If core components are used in a specific context they are referred to as *business information entities* (BIE). In figure 2 the *business information entity* `Quote` is depicted, stereotyped as *ABIE* (*aggregate business information entity*). Underneath the body *information entity* (in our case `QuoteDocument`) the modeler uses *business information entities* to structure the business document.

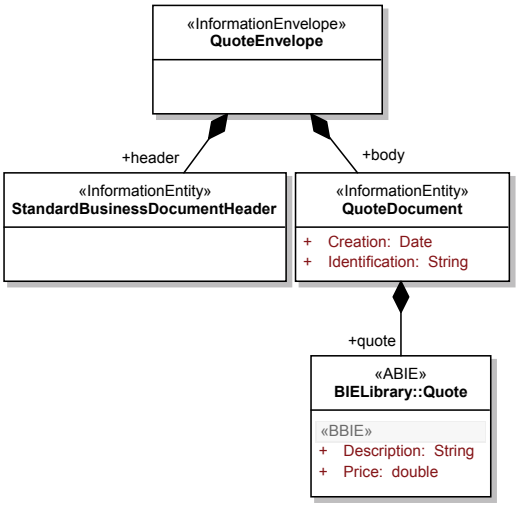

Figure 2. Business Information in UMM

Together with the core components standard UN/CEFACT has released so called Naming and Design Rules for Core Components (NDR) [16]. The NDR are guidelines for generating an XSD schema out of a core components model as depicted in figure 2. In order to support the modeler in establishing valid UMM models we have developed the so called UMM Add-In [7]. The UMM Add-In is an extension of the UML modeling tool Enterprise Architect [14] and supports the modeler with the semi-automatic generation of artifacts or a model validation feature. We have extended our UMM Add-In with a generation feature and the modeler is now given the possibility to automatically generate an XSD schema out of a given core components model [4]. The generated XSD schema is then used to validate the actual information exchanged in the business process.

Listing 1 shows the XML schema, which is generated from the *business information view* in figure 2.

Listing 1. Business information view schema

```
0  <xsd:schema xmlns:inView="urn:ex:infView" xmlns:bie="urn:ex:bInfEnt"
1   xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:ex:infView"
2   elementFormDefault="qualified" attributeFormDefault="unqualified">
3    <xsd:import namespace="urn:ex:bInfEnt" schemaLocation="ABIEs.xsd"/>
4    <xsd:element name="QuoteEnvelope" type="inView:QuoteEnvelopeType"/>
5    <xsd:complexType name="QuoteEnvelopeType">
6     <xsd:sequence>
7      <xsd:element name="StandardBusinessDocumentHeader" type="xsd:string"/>
8      <xsd:element name="QuoteDocument" type="inView:QuoteDocumentType"/>
9     </xsd:sequence>
10   </xsd:complexType>
11   <xsd:complexType name="QuoteDocumentType">
12    <xsd:sequence>
13     <xsd:element name="Creation" type="xsd:date"/>
14     <xsd:element name="Identification" type="xsd:string"/>
15     <xsd:element name="Quote" type="bie:QuoteType"/>
16    </xsd:sequence>
17   </xsd:complexType>
18  </xsd:schema>
```

The *business information entities* are generated separately and imported into the *business information view* schema (Line 3 of listing 1). Listing 2 shows the XML schema for the *business information entity* `Quote`, which is imported in line 3 of listing 1.

Listing 2. Business information entity schema

```
18 <xsd:schema xmlns:bie="urn:ex:bInfEnt" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
19  targetNamespace="urn:ex:bInfEnt" elementFormDefault="qualified"
20  attributeFormDefault="unqualified">
21 <xsd:complexType name="QuoteType">
22  <xsd:sequence>
23   <xsd:element name="Description" type="xsd:string"/>
24   <xsd:element name="Price" type="xsd:double"/>
25  </xsd:sequence>
26 </xsd:complexType>
27 </xsd:schema>
```

The business transaction depicted in figure 1 is successful if a `Quote` is returned within the `QuoteEnvelope`. Hence, if the element `Quote` shown in line 15 of listing 1 is not present, the business transaction fails - if a quote is given the business transaction is successful. The XPath expression is used in our mappings to decide on the business transaction's success or failure. Given the generated XML schema of the exchanged quote information we derive the following XPath expression:

```
/QuoteEnvelope/QuoteDocument/Quote
```

Given the current status the modeler is required to write the XPath expression manually. A future extension of the UMM Add-In will allow the modeler to generate the XPath expression with the support of a graphical user interface.

3.2 Generating WSDL

The business transaction depicted in figure 1 describes how the business process is coordinated between the two business partners. In a Web Service environment, this coordination is done using BPEL specifications where each service interface is described by means of WSDL. In our example the generation of BPEL from the UMM model results in four WSDL files for the following entities: *business application buyer*, *buyer*, *seller* and *business application seller*. Each WSDL file specifies the operations a party offers, the messages and data types which are exchanged as well as the services the operations are bound to. In the WSDL file of the buyer and the seller the partner link types are specified too, representing the interaction between the BPEL process and the involved parties. As an example, listing 3 shows the WSDL file of the buyer. Due to space limitations parts of the WSDL have been left out.

Listing 3. WSDL file of the buyer

```
27 <?xml version="1.0" encoding="UTF-8"?>
28 <wsdl:definitions name="Buyer" targetNamespace="http://buyer.at"
29     [...]
30 <wsdl:types>
31     [...]
32 </wsdl:types>
33 <wsdl:message name="QuoteRequestEnvelope">
34  <wsdl:part name="parameters" element="xsd1:QuoteRequestEnvelope" />
35 </wsdl:message>
36 <wsdl:message name="receiveAckRequest">
37  <wsdl:part name="parameters" element="bas:Acknowledgement"/>
38 </wsdl:message>
39     [...]
```

```
40 <wsdl:portType name="Buyer">
41  <wsdl:operation name="receiveQuoteRequestFromBA">
42   <wsdl:input message="tns:QuoteRequestEnvelope" />
43  </wsdl:operation>
44  <wsdl:operation name="receiveAck">
45   <wsdl:input message="tns:receiveAckRequest" />
46  </wsdl:operation>
47     [...]
48 </wsdl:portType>
49 <wsdl:binding name="BuyerSOAP" type="tns:Buyer">
50     [...]
51 </wsdl:binding>
52 <wsdl:service name="Buyer">
53     [...]
54 </wsdl:service>
55 <plnk:partnerLinkType
56  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
57  name="Buyer-BusinessApplication-PLNKT">
58  <plnk:role name="BusinessApplication"
59  portType="ba:BuyerBusinessApplication" />
60  <plnk:role name="Buyer" portType="tns:Buyer" />
61  </plnk:partnerLinkType>
62 <plnk:partnerLinkType
63  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
64  name="Buyer-Seller-PLNKT">
65  <plnk:role name="Buyer" portType="tns:Buyer" />
66  <plnk:role name="Seller" portType="wsdl1:Seller" />
67 </plnk:partnerLinkType>
68 </wsdl:definitions>
```

In the WSDL file above the data types used and messages exchanged during the process are described between line 28 and 38. The different operations of the buyer are specified between line 40 and 48. The actual web service endpoint through which the buyer is accessible and the binding to specific operations are specified between line 49 and 54. The UMM process per se requires different acknowledgement messages exchanged between the participating business partners - i.e., *acknowledgements of receipt* or *acknowledgements of processing*. The operation `receiveAck` specified in line 44 handles the different acknowledgements received by the buyer. The distinction which type of acknowledgement has currently been received is made via the exchanged acknowledgement type. Listing 4 shows a simplified example XML type of an acknowledgement. Through the element in line 70 a reference to the respective message is made and the element in line 72 specifies the type of acknowledgement - e.g., *acknowledgement of receipt*.

Listing 4. XSD schema of the buyer

```
68 <xsd:complexType name="AcknowledgementType">
69  <xsd:sequence>
70   <xsd:element name="Reference" type="xsd:string" />
71  </xsd:sequence>
72  <xsd:attribute name="type" type="xsd:string" />
73 </xsd:complexType>
```

3.3 Generating Partner Link Types

Partner link types represent the relationship between two services and their roles within. Each role refers to a specific port type of the WSDL file. In our example the buyer interacts with his business application (line 55 of listing 3) and with the seller (line 62 of listing 3). The seller has also two partner link types - one for the interaction with the buyer and one for the interaction with his business application.

The UMM model describes the transaction from an overall point of view. However, BPEL describes a process from the point of view of a specific business partner. Hence, the UMM model results in two different BPEL files - one for the buyer and one for the seller. In order for these BPEL processes to be compliant to each other the partner links define the relationship of the process with its partners. Listing 5 shows a cut-out from the BPEL process of the buyer. In line 77 of listing 5 the partner link to the seller is specified by referencing the partner link type `Buyer-Seller-PLNKT` (specified in line 62 of listing 3). Since the buyer is the owner of the process the attribute `myRole` refers to the buyer role of the partner link type and the attribute `partnerRole` to the seller role. The same constellation is stated in the BPEL file of the seller with the roles specified vice versa.

Listing 5. Partner Link example

```
73 <bpel:partnerLinks>
74  <bpel:partnerLink myRole="Buyer" name="Buyer-BusinessApplication-PLNKT"
75   partnerLinkType="ns1:Buyer-BusinessApplication-PLNKT"
76   partnerRole="BusinessApplication"/>
77  <bpel:partnerLink myRole="Buyer" name="Buyer-Seller-Link"
78   partnerLinkType="ns1:Buyer-Seller-PLNKT" partnerRole="Seller"/>
79  </bpel:partnerLinks>
```

3.4 Generating BPEL

In the following two sub sections we show the derivation of executable BPEL processes specifying the business service interfaces of both partners. At first, we elaborate the buyer's local choreography. Afterwards, we discuss the differences in terms of the seller's process. In order to visualize the processes we use the graphical notation of a tool named ActiveBPEL Designer [1] since it allows a more thorough understanding than showing the BPEL code listings directly.

3.4.1 Generating the Buyer's local Choreography

The buyer's process is shown in figure 3.4.1. The first activity is a *receive* called `ReceiveQuoteRequestFromBA` (1 in figure 3.4.1) awaiting input - the `QuoteRequestEnvelope` (B in figure 1) - from the business application. Since it is the first activity, it creates a new BPEL process instance upon receipt of the document. In a UMM *business transaction*, this action - performed within the buyer's system - is carried out as part of the *requesting business activity* (A).

As we learned, a *business transaction* is a unit of work that may be re-initiated due to time-out exceptions. In case of a time-out - that is actuated if an acknowledgement or a business document is not received within the agreed time - the buyer has to re-start the *business transaction*. The number of allowed retries is specified by the *retry count* (E). In order to allow possible re-starts of the business transaction we use the *repeat until* activity - `CheckRetryCount` (2) - containing the actual process flow. If a `TimeOutException` occurs, the `CheckRetryCount`

activity starts a new iteration - until its condition evaluates true. A new iteration is only started if the *retry count* is greater than zero and a variable `ProcessEnded` is false. For representing the *retry count*, we use a BPEL *variable* that is decremented each time a `TimeOutException` occurs.

According to our example, the `request for quote` transaction may be re-initiated three times - which amounts to four runs in total. Within `CheckRetryCount`, a *scope* (3) is used to structure the business process. As shown in figure 3.4.1 the *scope* consists of a *sequence* carrying out the regular process (4), *fault handlers* (5) and *event handlers* (6). At first, however, we concentrate on the regular process:

Regular process The buyer starts the actual business transaction by an *invoke* (7) calling the seller's operation `CalculateQuote` (C). According to our example, `CalculateQuote` consumes a `QuoteRequestEnvelope` (B). Since the seller needs some time to calculate the quote, the invoke is performed asynchronously and the buyer's process is continued immediately. Next, the *assign* activity (8, `SaveTimestampQuoteRequestSend`) saves the current time to a BPEL *variable*. Based on the time the quote request was sent, we calculate the time limits within acknowledgements and the `QuoteEnvelope` are expected.

In the next step, the buyer's business service interface waits for an acknowledgement from the seller confirming the receipt of the `QuoteRequestEnvelope`. It must be received within two hours - the agreed *time to acknowledgement receipt* (G). Therefore, we use a *pick* activity (9, `WaitForAckReceipt`) that waits for the occurrence of one event from a set of events. According to the semantics of a UMM *business transaction*, there are two possible scenarios in this step:

Firstly, the acknowledgement is received, which is modeled using an *on message* branch (10). In this case, the acknowledgement is handed over to the business application to check the acknowledgement's content. The communication with the buyer's business application is defined with a synchronous *invoke* called `CheckAckReceipt` (11). Its result is evaluated by the following *if* activity (12). In case the checks fail, the *throw* activity (13) raises a `FailedBusinessControlException`. The handling of exceptions is outlined at the end of this sub section.

Secondly, if no *acknowledgement of receipt* is received in time, the *on alarm* event (14) is actuated. *On alarm* corresponds to a timer-based alarm - either specified by a deadline (in terms of a timestamp) or by a duration. In our example, we define the time limit as a deadline that equals to the `TimeQuoteRequestSent` (saved before in the *assign* activity (8)) plus *time to acknowledge receipt*. In case the deadline is met, a `TimeOutException` is thrown (15). As outlined later, handling the `TimeOutException` results in decrementing the *retry count* and terminates the current attempt by going back to `CheckRetryCount`.

If an *acknowledgement of receipt* was properly received the buyer expects an *acknowledgement of processing*. It indicates that the seller's business application is able to process the `QuoteRequestEnvelope`. Handling an *acknowledgement of processing* (16) is similar to handling an *acknowledgement of receipt*, except that the agreed time limit may differ. According to our example, the seller must send the acknowledgement within 6 hours (F) - i.e., the deadline amounts to `TimeQuoteRequestSent` plus *time to acknowledge processing*. Again if the message is not received on time (17), a `TimeOutException` is thrown. Otherwise, upon receipt of the *acknowledgement of processing* it is delivered to the buyer's business application for checking its content (18, `CheckAckProcessing`). If the check was successful, the process expects the `QuoteEnvelope` (D) to be received.

Similar to the wait for acknowledgements a *pick* activity (19, `WaitForQuote`) is used to specify the time constraint. In case the deadline (`TimeQuoteRequestSent` plus *time to respond*) (H) is elapsed a `TimeOutException` is thrown (20). Otherwise, receiving the `QuoteEnvelope` activates on the *on message* branch. In a first step, the current time is stored (21) into a *variable* (`TimeQuoteReceived`) for further processing. In case an intelligible check - i.e., grammar, sequence and schema validation - of the document is required (22), it is sent to the business application via a synchronous *invoke* (23, `CheckQuote`). Given an erroneous `QuoteEnvelope` reported by the business application (24) a `FailedBusinessControlException` is raised.

If the `QuoteEnvelope` is properly received the process proceeds to `SendAckReceipt` (25). This asynchronous *invoke* confirms the successful receipt of the `QuoteEnvelope` to the seller. Subsequently, the `QuoteEnvelope` is handed over to the buyer's business application for further processing (26, `SendQuoteToBA`). The business application synchronously returns whether the `QuoteEnvelope` can be processed or not. In the latter case, a `FailedBusinessControlException` is thrown (27). In the former case, the seller is provided with an *acknowledgement of processing* (28, `SendAckProcessing`).

The business transaction must be kept alive to allow the seller to issue a failure if he does not receive the acknowledgement (`TimeOutException`) or is not able to process it accordingly (`FailedBusinessControlException`). Receiving failures from the seller is detailed in the paragraph about *event handlers*. The time until the buyer has to wait corresponds to `TimeQuoteEnvelopeReceived` plus the *time to acknowledge processing* (I) of the document. Keeping the business transaction alive is specified by a *wait* activity (29) with the given deadline.

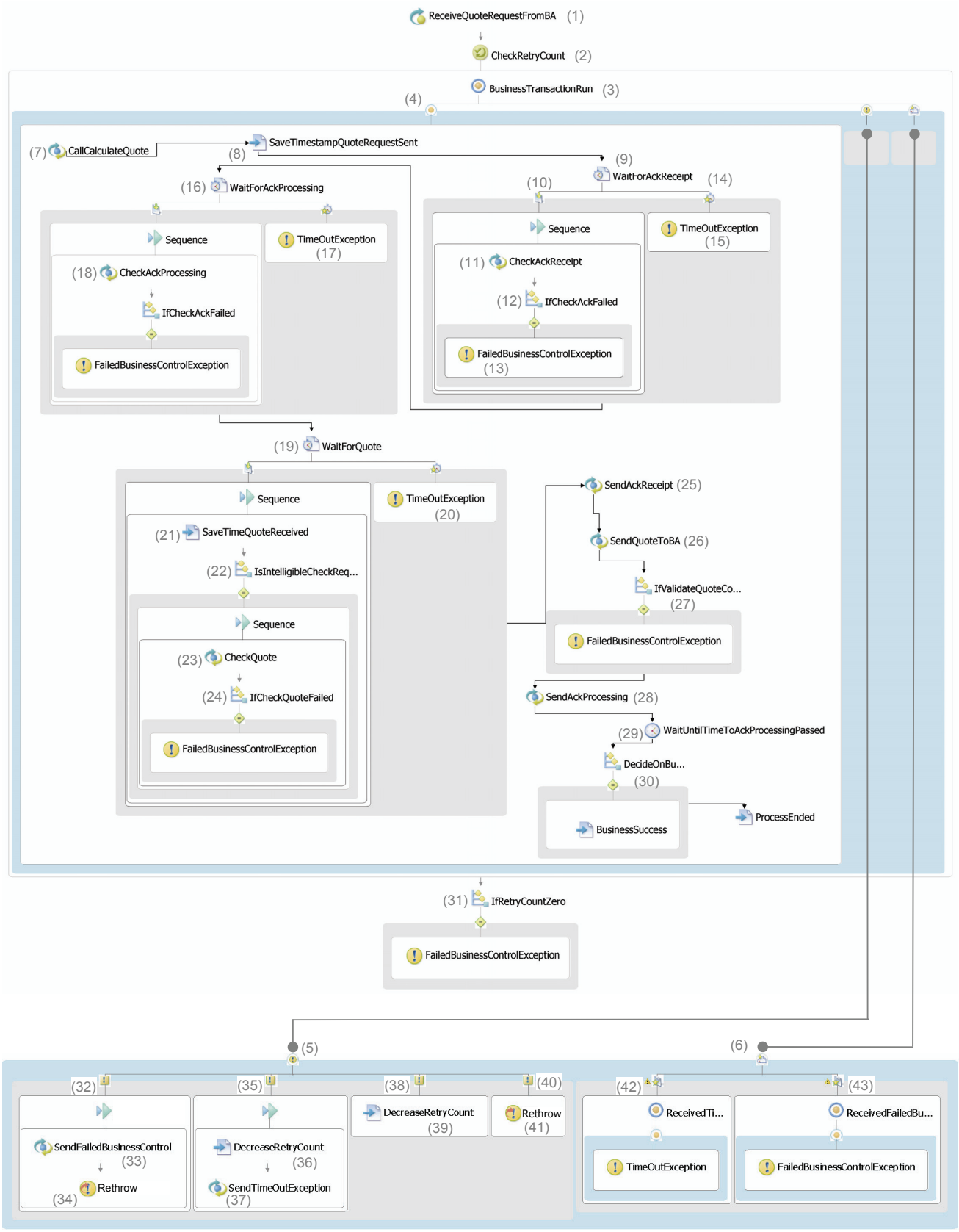

Figure 3. Buyer's BPEL Process

After waiting the process has to check if the business transaction succeeded in a business sense (30) - i.e., if the product was quoted or not. The checking is done by using the XPath expression as discussed above. In other words, we examine the `QuoteEnvlope` if the element `Quote` is present. In case of success, a boolean variable is set true by the assign activity `BusinessSuccess`. On the level of a business collaboration, the flow of the process is governed based on this information. Afterwards, the variable `ProcessEnded` is set true to prevent further runs of the *repeat until* activity `CheckRetryCount` (2). Finally, an *if* activity (31) checks if the *retry count* equals zero. A retry count of zero indicates that all attempts to execute the business transaction failed. In this case, a `FailedBusinessControlException` is thrown.

Fault handlers During the execution of the business process abnormal behavior is represented by exceptions. In UMM business transactions we distinguish between two types of exceptions: *time-out exceptions* and *failed business control exceptions*. The former ones, indicate that a certain message is not received on time. The latter ones signal that a message's integrity is violated or that the maximum amount of attempts to re-initiate the business transactions is reached. Exceptions are either raised internally if abnormal behavior is encountered or are signalized by the business partner.

In BPEL, exceptions are managed by so-called *fault handlers* (5). As figure 3.4.1 shows, the actual process flow is modeled in a *scope* (3), which is the only activity within the *repeat until* container (2). The *fault handlers* are attached to the *scope*. If an exception is encountered, the current execution of the *scope* is terminated and the corresponding *fault handler* is activated. In our example, we installed four different *fault handlers*: The first one (32) is activated if a `FailedBusinessControlException` is thrown internally. In this case, the buyer communicates the seller the failure via the *invoke* `SendFailedBusinessControl` (33) including a reason of failure. Subsequently, the exception is re-thrown (34) to the enclosing owner of the business transaction. The second one (35) handles an internally raised `TimeOutException`. It decreases the available retries by an *assign* activity (36) and communicates the `TimeOutException` (37) referencing the missing message to the seller. Activating a *fault handler* skips the current execution of the *scope*. Since the *scope* is contained in the *repeat until* activity, the loop condition is re-evaluated after fault handling. Thus, it is possible to re-initiate a business transaction. The third *fault handler* (38) treats `TimeOutExceptions` signaled by the seller. The *retry count* is again decremented (39) and the control flow is handed back to the *repeat until* activity. The last *fault handler* (40) catches a `FailedBusinessControlException` that was received from the seller. In this case, the exception is handed over to the process owner (41).

One should note, that the `FailedBusinessControlException` that is raised at the end of the process if the retry count is zero (31) is not handled by the fault handlers (5) attached to the *scope* (3). Instead, we install a *fault handler* - working similar as the first one described above - to the scope of the BPEL process.

Event handlers As we learned, the buyer may receive a `TimeOutException` or a `FailedBusinessControlException` at any time during the process execution. To implement this requirement, we use the concept of *event handlers* provided by BPEL. *Event handlers* listen to events concurrently with the regular process execution. We attach two *event handlers* (6) to the *scope* (3) representing a run of a *business transaction*: The first *event handler* (42) deals with a `TimeOutException` signaled by the seller. Once it is received, an internal `TimeOutException` is raised that is managed by the corresponding fault handler (38). The second one (43) is responsible for `FailedBusinessControlExceptions`. Similarly, it re-throws the exception to be further treated by the fourth *fault handler* (40).

3.4.2 Generating the seller's local choreography

The concepts used in the seller's process are widely analog to those described with respect to the buyer's process. Since, the buyer's and the seller's process must be complementary, the order of receiving and sending the `QuoteRequestEnvelope` and the `QuoteEnvelope`, respectively, is reversed. This also includes the handling of the acknowledgements.

The major difference between the two processes results from the fact that the seller does not control the *retry count*. If the seller does not receive an acknowledgement on time, a `TimeOutException` is issued back to the buyer but no *retry count* is decreased. Hence, no loop activity is required in the seller's process. Due to space limitations we do not depict a diagram visualizing the seller's process.

4 Related Work

A first theoretical examination of the interdependencies of UMM and BPEL has been done by Hofreiter et al. in [3]. The subject of the paper was to scrutinize whether BPEL is appropriate for capturing the process modeled in UMM. However, the BPEL derived in their paper is not executable and the mappings shown are very simplified.

Several approaches have taken up the idea of composing services with the help of UML diagrams. Skogan et

al. propose a method using UML activity diagrams to design web services and OMG's Model Driven Architectures (MDA) to generate executable specifications in BPEL [13]. Their model allows to import existing web service descriptions stored in WSDL files into a UML diagram. Unlike the UMM approach the idea pursued by Skogan et al. does not take the business context into account.

A model driven approach to BPEL specifications has also been proposed in [10] by using BPMN process models. The solution presented uses business process diagrams and splits them up into components. Using a rule based and an activity based translation these components are then transformed into BPEL code. Similar to the first approach presented, this methodology also does not take into account the business context.

Another approach based on the UMM has been shown in [5]. The model presented focuses on how each partner has to realize the message exchanges to support an agreed choreography. The resulting model is based on UML state machines and shows how each business partner has to react on incoming messages and on messages missing.

An approach using global choreographies to derive local choreographies has been presented in [8]. The work uses WS-CDL specifications representing global choreographies to generate local BPEL specifications. Similar to BPEL, WS-CDL is intended to be processed by machines. However, for implementing a successful B2B solution, first a conceptual modeling approach considering the specifics of B2B - like UMM - is required to capture the collaborative space between enterprises.

5 Conclusion and Outlook

In this paper we proposed a mapping from UMM business transactions to executable BPEL processes. The logical flow of the process corresponds to the UML activity graph of a business transaction. As we showed, the rather simple looking UMM business transaction results in a complex BPEL process. With the use of scopes, event handlers and fault handlers the different types of business signals which are exchanged during the process are managed.

This work provides a holistic approach by integrating both, the process flow based on UMM and the exchanged business documents based on core components. For generating the XML schemas out of the core components model we built our solution on top of the work done in [4]. As we outlined, our holistic approach allows to decide if the business transaction resulted in a business success or in a business failure.

In terms of future work we will focus on the mapping of whole UMM business collaborations to BPEL. As we know, a business collaboration choreographs a flow of business transactions. Thus, the work achieved in this paper will serve as a foundation for the future work on business collaborations. Furthermore we will focus on the elaboration of meaningful correlation sets in order to manage the exchange of business documents.

References

[1] Active Endpoints. *ActiveBPEL Designer*.

[2] N. C. Hill and D. M. Ferguson. Electronic Data Interchange: A Definition and Perspective. *EDI Forum: The Journal of Electronic Data Interchange*, 1(1):5–12, 1989.

[3] B. Hofreiter and C. Huemer. Transforming UMM Business Collaboration Models to BPEL. In *Proceedings of the International Workshop on Modeling Inter-Organizational Systems (MIOS)*, 2004.

[4] C. Huemer and P. Liegl. A UML Profile for Core Components and their Transformation to XSD. accepted at the Second International Workshop on Services Engineering, Istanbul, Turkey, 2007.

[5] C. Huemer and M. Zapletal. A State Machine executing UMM Business Transactions. In *Proceedings of the IEEE International Conference on Digital Ecosystems and Technologies*. IEEE, 2007.

[6] ISO. *Open-edi Reference Model*, 1995. ISO/IEC JTC 1/SC30 ISO Standard 14662.

[7] P. Liegl, R. Schuster, and M. Zapletal. *UMM Add-In*. Research Studios Austria, 2007. Version 0.8.2, http://ummaddin.researchstudio.at.

[8] J. Mendling and M. Hafner. From WS-CDL Choreography to BPEL Process Orchestration. In *Journal of Enterprise Information Management (JEIM). Special Issue on MIOS Best Papers*.

[9] OASIS Web Services Business Process Execution Language (WSBPEL) TC. *Web Services Business Process Execution Language Version 2.0*, Jan. 2007. Version 2.0, Commitee Specification.

[10] C. Ouyang, M. Dumas, A. H. ter Hofstede, and W. M. van der Aalst. From BPMN Process Models to BPEL Web Services. In *Proceedings of the International Conference on Web Services*, pages 285–292. IEEE, 2006.

[11] C. Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10):46–52, 2003.

[12] RosettaNet. *RosettaNet Implementation Framework: Core Specification*, Dec. 2002. V02.00.01.

[13] D. Skogan, R. Gronmo, and I. Solheim. Web Service Composition in UML. In *Proceedings of the 8th IEEE Enterprise Distributed Object Computing Conference*, pages 47–57. IEEE, 2004.

[14] Sparx Systems. *Enterprise Architect*.

[15] UN/CEFACT. *UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module*, Sept. 2006. Technical Specification V1.0, http://www.unece.org/cefact/umm/UMM_Foundation_Module.pdf.

[16] UN/CEFACT Applied Technology Group (ATG). *XML Naming and Design Rules*, February 2006. 2.0.

[17] UN/CEFACT TMG. *Core Components Technical Specification*, November 2003. v2.01.

[18] World Wide Web Consortium (W3C). *Web Services Description Language (WSDL)*, Mar. 2001. Version 1.1.