A model-driven top-down approach to inter-organizational systems: From global choreography models to executable BPEL

Birgit Hofreiter, Christian Huemer

University of Vienna, Austria, birgit.hofreiter@univie.ac.at.at
Vienna University of Technology, Austria, huemer@big.tuwien.ac.at

Abstract

Today, most approaches for inter-organizational business processes start bottom-up from the interfaces and the workflows of each partner described on the IT layer. Alternatively, one may start from the commitments and agreements between business partners to reach their complementary business goals. The latter approach is target of the UN/CEFACT Modeling Methodology (UMM), which models a global choreography. In a model driven approach the UMM artifacts must be further elaborated towards an IT solution for each participating business partner. For this purpose we have developed a UML profile to model a local choreography or an orchestration that respects the agreements made in the global choreography. In order to execute the local choreography / orchestration in the local IT, the processes must be machine-readable. For this purpose we demonstrate a transformation to the business process execution language (WS-BPEL).

1. Motivation

In this paper we concentrate on orchestrations and choreographies [11] in the context of inter-organizational business processes. Both terms are closely related, but must be well distinguished to follow this paper. Orchestration deals with the sequence and conditions in which one business process calls its components to realize a business goal. Choreography describes business processes in a peer-to-peer collaboration. It describes the flow of interactions between the participating business partners that interlink their individual processes. We distinguish local and global choreographies. A local choreography describes the flow from a participating partners point of view. It makes the public parts of its local process visible to others. A global choreography defines the inter-organizational process from a neutral perspective. A global choreography has the potential to achieve an agreement between the partners. Local choreographies enable the configuration of each partners system.

If business partners want to collaborate they must have complementary local choreographies - which is rather unlikely if the local choreographies have been developed in isolation. Thus, we prefer an approach that starts off with an agreement on a global choreography between the business partners. Each business partner may develop its local choreography or orchestration in accordance to the effected global choreographies. The resulting model of a local choreography or orchestration has then to be transformed into a machine-readable workflow language to be executed in the local IT of the business partner.

An approach for modeling global choreographies is delivered by the United Nations Centre of Trade Facilitation and e-Business and their UN/CEFACT Modeling Methodology (UMM). The UMM specification is defined as a UML profile [15], which we have co-edited. Usually, commitments are made on a bi-lateral basis. Accordingly, UMM models always describe business collaborations between two parties. Also similar to a contract, a UMM model describes the commitments and agreements from a neutral perspective. In summary, the UMM provides a UML-based methodology for bi-lateral and global choreographies.

The next step in the top-down approach is the definition of the processes at each partners side that respect the commitments made on the level of the global choreographies. The local process of a partner may include communications with many different other partners - defined in different global choreographies. Consequently, we need a methodology that extends the concepts of UMM in order to model a local multi-party choreography or orchestration. Since UMM is UML-based this extension should also be defined as a UML profile. In our paper [1] we have presented a UML profile for exactly this purpose.

Having defined the local business processes by means of UML, it is necessary to transform them into a workflow language that may be processed by the local IT systems. In an Web Services environment the language of choice is BPEL.

It follows that the automatic generation of BPEL code from the UML models requires a well-defined mapping from the stereotypes of the UML profile for local choreographies /orchestrations to the BPEL constructs. This paper defines the corresponding mapping.

The remainder of this paper is structured as follows: We provide an overview of related work in section 2. In section 3 we introduce UMM by the means of a simple example. Section 4 uses this example to demonstrate the UML profile for local choreographies. In Section 5 we define the transformation to BPEL 5. Section 6 summarizes the paper.

2. Related Work

In order to use UML for modeling business processes different authors have developed either just guidelines or a UML profile that customizes UML for business process modeling. Customizations of UML for modeling business processes internal to a company are described in [12] [8] [6]. Beside UMM, UML customizations for modeling inter-organizational processes are described in the RosettaNet Framework [13] and in Kramler et al. [7].

Already in 2004 we investigated the interdependencies of UMM and BPEL in [2]. The subject of the paper was to scrutinize whether BPEL is appropriate for capturing the choreography modeled in UMM. However, going directly from UMM to BPEL does not allow to derive local choreographies between multiple parties. Furthermore, it cannot result in executable processes, since this approach does not cover all the internal activities in an orchestration. A three step approach as described in this paper is much more advances, since it leads to executable multi-party processes.

Another approach defining a direct mapping from a global choreography to a BPEL process is delivered by Khalaf [5]. He transforms the global choreography of a RosettaNet model into BPEL.

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 [14]. 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.

An approach using global choreographies to derive local choreographies has been presented in [9]. 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.

3 UN/CEFACTs Modeling Methodology

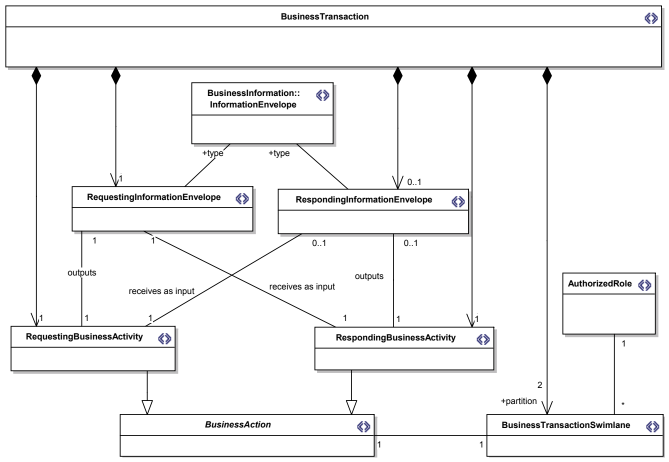
UN/CEFACTs Modeling Methodology (UMM) models the choreography and data exchange commitments independent of the IT. It is a methodology that starts off with analyzing the business environment and the requirements of each partner. In a next step the requirements for a collaboration in an inter-organizational business process are analyzed. These requirements are transformed into a global choreography between the partners. The documents exchanged in the choreography follow the concepts of the core components specification.

By following the methodology a number of well defined UML artifacts are created. These artifacts must follow the UMM meta model that is defined as a UML profile. Accordingly, the UMM meta model specification covers a set of well defined stereotypes including tagged definitions for each of the above mentioned views. Due to page limitations we are not able to elaborate in detail on all artifacts. We limit ourself to the artifacts describing the resulting global choreography: business transaction and business collaboration protocol. The reader interested in more UMM details is referred to the technical specification [15] which we have co-edited and to our publication in [3] detailing many UMM backgrounds.

3.1 UMM Business Transaction

A *business transaction* is an atomic business process between two *authorized roles* synchronizing the *business entity states* between them. It involves sending business information from one *authorized role* to the other and an optional reply. The *business transaction* is built by two partitions - one for each *authorized role*. Hence, a *business transaction* is composed of exactly two *business transaction swimlanes*. Each *business transaction swimlane* relates to one *authorized role*. An *authorized role* is assigned to exactly one *business transaction swimlane*. It follows, that the two swimlanes of a *business transaction* must be assigned to different *authorized roles*.

Within a *business transaction* each *authorized role* performs exactly one *business action* - the *requesting authorized role* performs a *requesting business activity* and the



BusinessTransaction
BusinessInformation::InformationEnvelope
RequestingInformationEnvelope
RespondingInformationEnvelope
RequestingBusinessActivity
RespondingBusinessActivity
AuthorizedRole
BusinessAction
BusinessTransactionSwimlane
+type
outputs
receives as input
+partition

Figure 1. Business transactions: meta model

responding authorized role performs a *responding business activity*. Each *business action* - no matter whether *requesting* or *responding business activity* - is assigned to a *business transaction swimlane*, and each *business transaction swimlane* comprises exactly one *business action*.

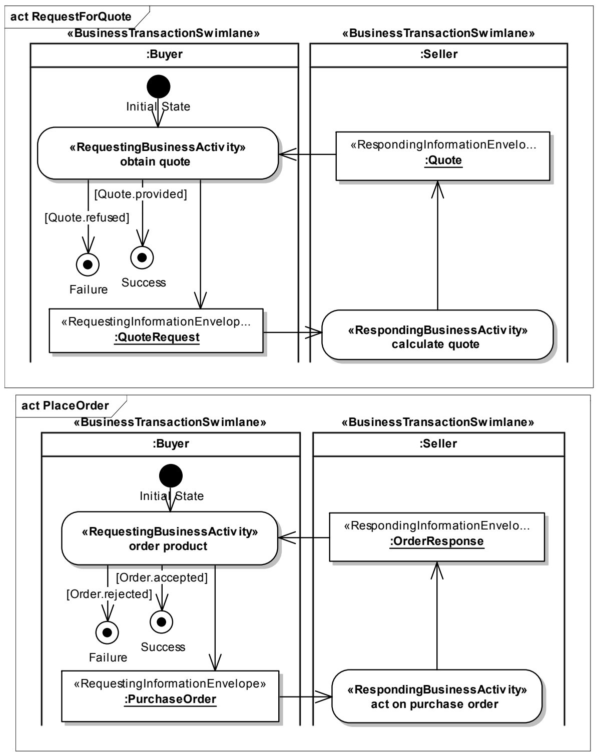
The *requesting business activity* outputs the *requesting information envelope* that is input to the *responding business activity*. The *responding information envelope* created by the *responding business activity* and returned to the *requesting business activity* is optional.

For a better understanding we demonstrate the relevant UMM artifacts by a simple `order from quote` example. This example involves two *business transactions* `request for quote` and `place order` which are depicted in figure 2.

Both *business transactions* follow the meta model described before. For a better understanding we detail the `request for quote` case. It consists of two *business transaction swimlanes*, one for the `buyer`, another one for the `seller`. The buyer performs the *requesting business activity* `obtain quote` which outputs a `quote request` setting the *business entity* `quote` to the interim state `requested`. The `quote request` is input to the sellers *requesting business activity* `calculate quote`. The activity sets the final state of the *business entity* `quote` either to `provided` or `refused`. The final state is communicated by returning the `quote` to the *requesting business activity* `obtain quote`.

3.2 Business collaboration protocol

The flow between *business transactions* is defined in a *business collaboration protocol*. Its meta model is depicted in figure 3. The activities of a *business collaboration protocol* are *business collaboration activities* and/or *business*



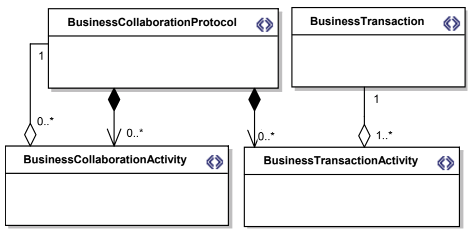
act RequestForQuote
«BusinessTransactionSwimlane»
:Buyer
«BusinessTransactionSwimlane»
:Seller
Initial State
«RequestingBusinessActivity» obtain quote
«RespondingInformationEnvelo... :Quote
[Quote.provided]
[Quote.refused]
Failure
Success
«RequestingInformationEnvelop... :QuoteRequest
«RespondingBusinessActivity» calculate quote
act PlaceOrder
«BusinessTransactionSwimlane»
:Buyer
«BusinessTransactionSwimlane»
:Seller
Initial State
«RequestingBusinessActivity» order product
«RespondingInformationEnvelo... :OrderResponse
[Order.accepted]
[Order.rejected]
Failure
Success
«RequestingInformationEnvelope» :PurchaseOrder
«RespondingBusinessActivity» act on purchase order

Figure 2. Business transactions

transaction activities. Hence, a *business collaboration protocol* is composed of zero to many *business collaboration activities* and of zero to many *business transaction activities*. However, at least one *business collaboration activity* or a *business transaction activity* must be present in a *business collaboration protocol*. Transitions defining the flow among the *business collaboration activities* and/or *business transaction activities* may be guarded by the states of *business entities*.

A *business transaction activity* is characterized by the fact that it is refined by a *business transaction*. Each *business transaction* must be at least once used to refine a *business transaction activity*. A *business transaction* may be nested in different *business transaction activities*.

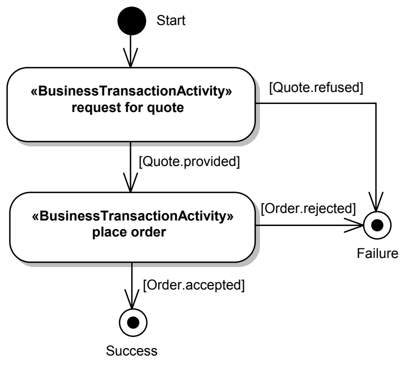
A *business collaboration activity* is characterized by the



BusinessCollaborationProtocol
BusinessTransaction
BusinessCollaborationActivity
BusinessTransactionActivity
1
0..*
0..*
0..*
1
1..*

Figure 3. BCP: meta model

fact that it is refined by another *business collaboration protocol*. Not each *business collaboration* is a refined *business collaboration activity* - only the nested *business collaboration protocols*. A *business collaboration protocol* may be nested in different *business collaboration activities*.



Start
«BusinessTransactionActivity» request for quote
[Quote.refused]
[Quote.provided]
«BusinessTransactionActivity» place order
[Order.rejected]
Failure
[Order.accepted]
Success

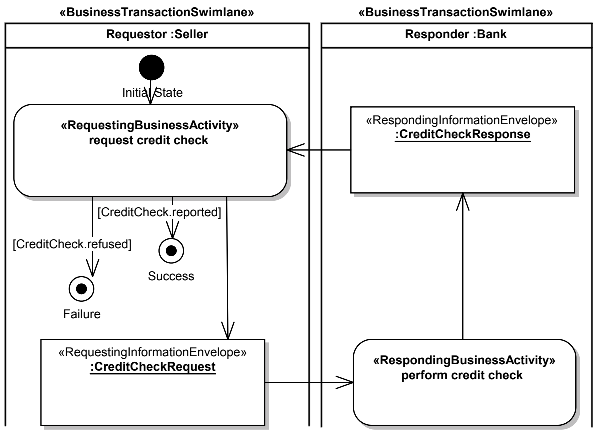
Figure 4. Business Collaboration Protocol

The *business collaboration protocol* `order from quote` - depicted in figure 4 - defines a sequence of the *business transaction activities* `request for quote` and `place order`. Each of the two is refined by the corresponding *business transaction* of figure 2.

3.3 Exkursion: Bi-lateral Nature of UMM

In this subsection we extend our previous example by another authorized role. We assume that the seller contacts the buyers bank to check his credit before giving a quote. From a business point of view the scenario is as follows: First the buyer submits his quote. In order to decide whether to make a firm quote or not, the seller requests the bank to check the buyers credit. After receiving the result of the credit check from the bank, the seller returns a quote document, which either includes the quote or a reason for quote rejection. Evidently, we need another business transaction for check credit between seller and bank. It is depicted in figure 5.

However, the *business transaction* `check credit` cannot be part of the *business collaboration protocol* of figure 4. In this *business collaboration protocol* we define the control flow between the `request for quote` and the `check credit` *business transaction activities*. In general, the transition from one *business transaction activity* to another one is triggered by the fact that the first one is completed. Looking at our example, it is clear that the `request for quote` *business transaction activity* is started first. However, it is not completed before the `check credit` *business transaction activity* starts. In fact, `check credit` is nested within `request for quote`. This means that the nested activity is triggered as part of starting the encompassing activity and the nested one has to complete for the encompassing one to finish. In case of nested business transaction activities, it is not possible to specify a transition between the *business transaction activities* in the *business collaboration protocol*.



«BusinessTransactionSwimlane»
Requestor :Seller
«BusinessTransactionSwimlane»
Responder :Bank
Initial State
«RequestingBusinessActivity» request credit check
«RespondingInformationEnvelope» :CreditCheckResponse
[CreditCheck.reported]
[CreditCheck.refused]
Success
Failure
«RequestingInformationEnvelope» :CreditCheckRequest
«RespondingBusinessActivity» perform credit check

Figure 5. Business Transaction: Check Credit

Thus, the current UMM is only suited for bi-lateral collaborations. From a UN/CEFACT perspective this seems to be ok, because it is the goal to model agreements and commitments between two business partners. Evidently, it is not intended to standardize how enterprises and organizations work internally. Accordingly, mandating a `credit check` as part of a `request for quote` is not the task of UN/CEFACT. It is an internal decision made by a seller, which is part of its internal business process.

4 A UML Profile for Local Processes

4.1 The Local Process by Example

Before explaining the meta model of a UML profile for local processes, i.e. local choreographies or orchestrations, we demonstrate the concepts by means of our `order from quote` example. We present the *local process* of the `seller` who also performs `check credit` with the `bank`.

The flow within this *local process* is depicted in figure 6. Its main building blocks are *initiating activities* and *reacting activities*. An *initiating activity* is used to model the inside of a *requesting business activity*. Similarly an *reacting activity* models the inside of a *responding business activity*. It follows that the sellers *local process* includes equivalent *reacting activities* for its *responding business activities* in the transactions of figure 2: `calculate quote` and `act on purchase order`.

The next step is to specify the flow between these two *reacting activities*. This flow is derived from the *busi-*

ness collaboration protocol `order from quote` in figure 4. The *reacting activities* (or the *responding business activities* respectively) represent the seller's task in the *business transaction activities* that are choreographed in this *business collaboration protocol*. It follows that the sequence of `request for quote` and `place order` in figure 4 is mapped to a sequence between `calculate quote` and `act on purchase order` in the seller's *local process*. Furthermore, the guard conditions are mapped. This means, a refused quote leads to a failure state. The transition from `calculate quote` to `act on purchase order` is guarded by the fact that a quote was provided. A rejected order leads to a failure state after `act on purchase order`, whereas an accepted order leads to a success.

The next step is detailing each of the *reacting activities* in the *local process*. An object node is added for each incoming and outgoing *information envelope*. *Initiating/reacting activities* taken from two-way *business transactions* have both an input and an output node. If they refer to a one-way *business transaction*, an *initiating activity* has only an output node and a *reacting activity* has only an input node. In our example, `calculate quote` and `act on purchase order` both are derived from *two-way business transactions*. Thus, they have an input and an output node. The *information envelopes* assigned to these nodes correspond to the input and output of the corresponding *requesting/responding business activities*. From figure 2 it follows that `calculate quote` has an input of `quote request` and an output of `quote response`. Similarly, `act on purchase order` receives an input of `purchase order` and outputs an `order response`.

For each input node we add a UML *accept event action* that is stereotyped as *receive business information*. It is used to recognize the event of an incoming *information envelope* and to hand it over to the *initiating/reacting activity*. In case of a *reacting activity* this event and the resulting transfer of the *information envelope* is required to start the *reacting activity*. In our example of figure 6 the overall initial state leads immediately to the `calculate quote` activity. However, before the first activity within `calculate quote` is started, the *accept event action* `exchange quote request` must recognize the receipt of a `quote request` and transfer it to `calculate quote`.

In order to demonstrate modeling the flow within an *initiating/reacting activity* we take a look on `calculate quote` in figure 6. As mentioned above it is started with an incoming `quote request`. So the first task of the flow is `file quote request`. The nested task `request credit check` from a bank is the next one. Once this is done the flow continues with `act on the credit check results`. Logically, the next step is either `provide quote` or `refuse quote`. In either case `exchange quote response` is the last task. Note, if a

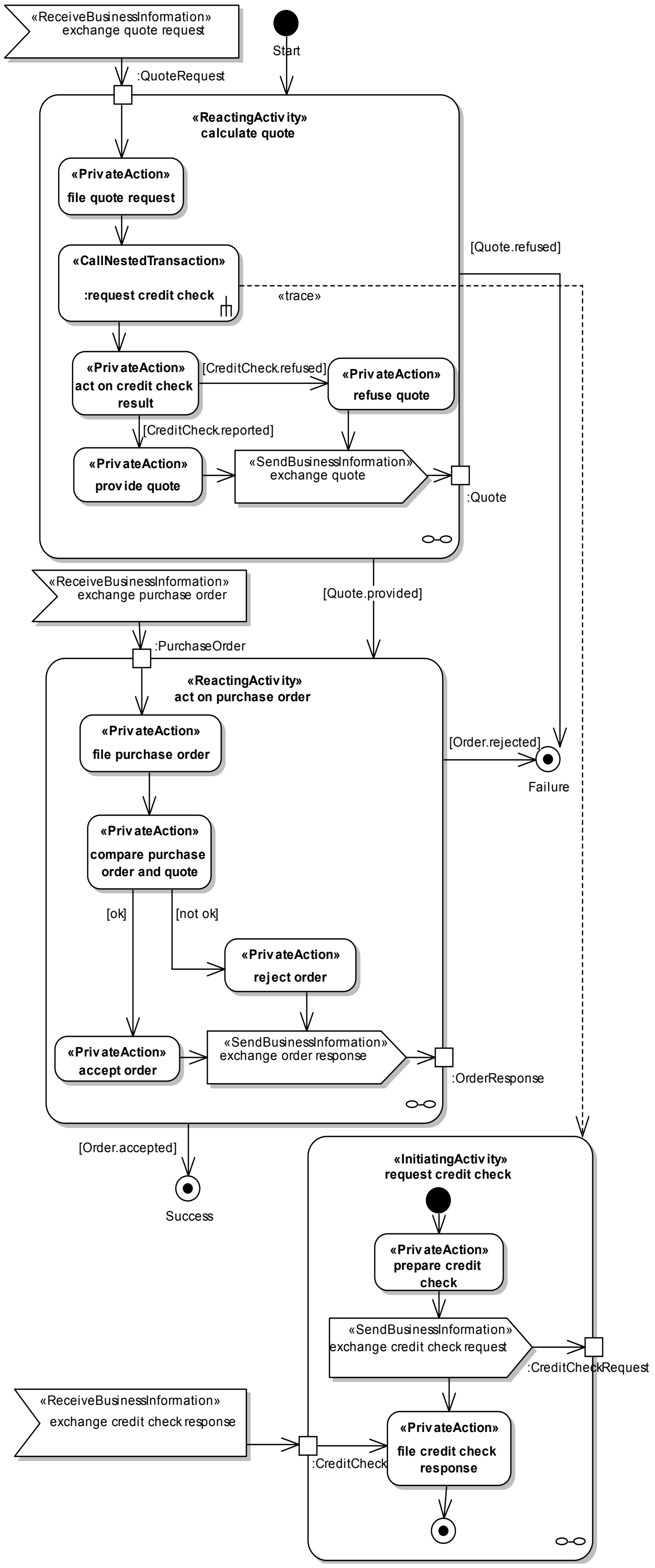

«ReceiveBusinessInformation» exchange quote request
Start
:QuoteRequest
«ReactingActivity» calculate quote
«PrivateAction» file quote request
«CallNestedTransaction» :request credit check
«trace»
[Quote.refused]
«PrivateAction» act on credit check result
[CreditCheck.refused]
«PrivateAction» refuse quote
[CreditCheck.reported]
«PrivateAction» provide quote
«SendBusinessInformation» exchange quote
:Quote
«ReceiveBusinessInformation» exchange purchase order
[Quote.provided]
:PurchaseOrder
«ReactingActivity» act on purchase order
«PrivateAction» file purchase order
[Order.rejected]
Failure
«PrivateAction» compare purchase order and quote
[ok]
[not ok]
«PrivateAction» reject order
«PrivateAction» accept order
«SendBusinessInformation» exchange order response
:OrderResponse
[Order.accepted]
Success
«InitiatingActivity» request credit check
«PrivateAction» prepare credit check
«SendBusinessInformation» exchange credit check request
:CreditCheckRequest
«ReceiveBusinessInformation» exchange credit check response
:CreditCheck
«PrivateAction» file credit check response

Figure 6. Local Process of the Seller

quote is refused the `quote response` will state the reason of rejection.

It is easy to recognize that the activities with an intitating/reacting activity are based on different stereotypes. Most of them are *private actions* which are tasks internal to the organization. Since these are not visible to others, they are not part of a local choreography, but of an orchestration. It follows, when modeling only a local choreography by the concepts of our UML profile, *private actions* must not be used.

The tasks `request credit check` and `exchange quote response` are also relevant for local choreographies. The latter is of stereotype *send business information* which is a special kind of the UML *send signal action*. In our example, `exchange quote response` transfers the `quote` to the output node of `calculate quote`. The fact, that the `quote` is returned to `obtain quote` (executed by the buyer) is not shown in a local process (c.f. figure 6 - it is already defined in the `request for quote` *business transaction* on top of figure 2.

`Check credit` is of stereotype *call nested transaction*. It enables nested transactions with third parties. As explained in subsection 3.3, checking the customer credit has do be done after receiving a `quote request` and before responding to it. This means that the *business transaction* `check credit` is started as part of the seller's `calculate quote` activity.

Accordingly, we define the concept of a *call nested transaction* as a special kind of the UML *call action behavior*, used to call another structured activity - which is an *initiating activity* of the same party. In our example of figure 6, the `calculate quote` activity includes the *call nested transaction* `request credit check`. This one calls the synonymously named *initiating activity request* `credit check` - which is the seller's task in the *business transaction* `check credit` (see figure 5).

We again specify a flow within the *initiating activity* `request credit check`. After finishing this flow, control is given back to `calculate quote`. Since `request credit check` is an *initiating activity*, its internal flow first includes a *send business information* action before receiving a return back. However, its flow is only able to continue with `file credit check`, if a `credit check response` is recognized by the *receive business information action* `exchange credit check response`. Furthermore, it should be noticed that *call nested transaction* is only used for calling a single *initiating activity*. If a whole collaboration is nested, the *call nested collaboration* concept is used. This one calls another *local process*.

4.2 Meta Model for Local Processes

In this subsection we discuss the meta model of our UML profile to model local processes. This meta model is illustrated in figure 7. It serves as a base to model the local business processes of a business partner who collaborates with multiple parties in order to reach his business goal. Per definition, the local business process describes the flow from a particular partner's point of view. The exmple process of the seller's order mangement depicted in figure 6 is compliant to this meta model.

The justification for developing another orchestration/choreography language is the dedicated binding to UMM allowing a straight-through modeling approach. Accordingly, the stereotypes of our local choreography have well-defined relationships to the UMM stereotypes. Thus, we fist elaborate on the relationships between its stereotypes to the ones of UMM:

- A *local process* describes a flow of activities from a participating partys point of view. It extends the concept of a UML activity, that is itself composed of further activities. All its activities are performed by the same party. Consequently, a *local process* is assigned to exactly one *authorized role* from UMM:

- A UMM *business collaboration protocol* always involves two parties. A *local process* - although executed by a single party - involves interactions with multiple parties, i.e. interactions defined in different *business collaboration protocols*. It follows, that a *local process* is related at least to one, but up to many *business collaboration protocols*. A *business collaboration protocol* may be reflected in many different *local choreographies*.

- When a UMM *business collaboration protocol* is part of a *local process*, it is mandatory that each of its *business transaction activities* results in a *local activity*, or in other words in an *initiating activity* or in a *reacting activity*. Since a *business transaction activity* is performed by two parties, it may be the source of up to two *local activities*. However, it should be noted that these two *local activities* will never be used in the same *local choreography*, because they are not executed by the same *authorized role*. Contrariwise, each *local activity* is backed up by exactly one *business transaction activity*.

- In UMM, a *business transaction activity* is refined by a *business transaction*, in which the *authorized role* under consideration in the *local process* performs either a *requesting business activity* or a *responding business activity*- which one is defined by the *authorized role* assigned to the *business transaction swimlane* hosting the *requesting/responding business activity*. If the *authorized role* executes a *requesting business activity*, the *local process* includes an *initiating activity*. In case of a *responding business activity*, the *local process* comprises a *reacting activity*. In other words a *requesting business activity* and an *initiating activity* (as well as a *responding business activity* and a *reacting activity*) represent they same logical concept - however one is used in the flow of inter-organizational systems and the other one in the local flow. In order to avoid mixing up the flows, the logical concepts results in two different stereotypes. However, there is always a one-to-one relationship between *requesting business activity* and *initiating activity* as well as between *responding business activity* and *reacting activity*.

The meta model in figure 7 shows the relationships between the stereotypes of the UML profile for local processes. It is our intension that local processes may be specified in the same model as the related UMM model. However, all artifacts of the local process must reside in their own package structure independent of the UMM.

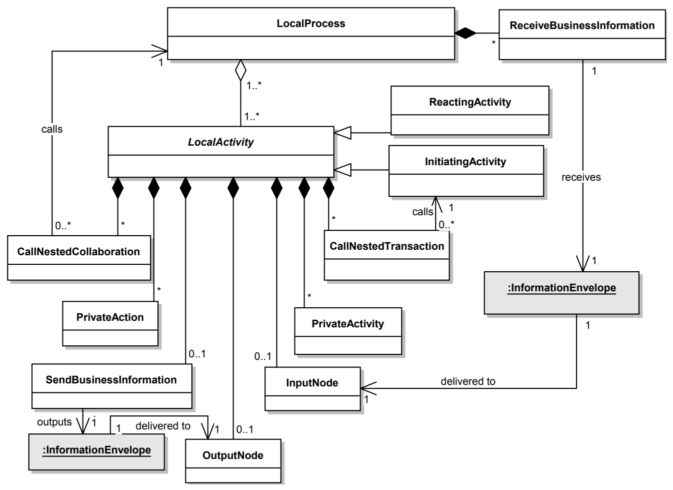


Figure 7. Local Processes: Meta Model

A *local process* includes at least one, but up to many *local activities*. *5* is the abstract super class of *initiating activity* and *reacting activity*. However, a *local process* does not exclusively own a *local activity*. One and the same *local activity* may be re-used in different *local processes* - since the related *business collaboration protocol* may be used in different *local processes*. It is important to note that the transitions between *local activities* must correspond - including their guards - to the transitions in the *business collaboration protocol* from which the *local business activities* are derived.

A *local process* may include *receive business information activities* in addition to *local activities* - however no other children are allowed. The concept of the stereotype *receive business information* is explained below.

In UMM, *requesting/responding business activities* send and/or receive *information envelopes*. This fact must be reflected in *local activities* as well. Thus, an object node is added to a *local activity* for each incoming and outgoing *information envelope*. *Local activities* taken from two-way *business transactions* have both an input node and an output node. If they are part of a one-way *business transaction*, an *initiating activity* has only an output node and a *reacting activity* has only an input node.

For each input node we add a *receive business information* - which is a UML *accept event action* - to the local process. It is used to recognize the event of an incoming *information envelope* and to deliver it to the input node of the corresponding *local activity*. In case of a *reacting activity* this event and the resulting transfer of the *information envelope* is required to start the *initiating activity*.

The flow within a *local activity* comprises the following concepts: *Private actions* and *private activities* are used to model tasks that are internal to the organization and are not visible to other parties. *Private activities* may be decomposed into further activities and actions, *private actions* do not. Private actions and private activities are used to model orchestrations. If the UML profile is used to model a local choreography, these two stereotypes must not be used.

The next concept is *send business information*. It is a special kind of the *send signal action* that is used to deliver an *information envelope* to the output node of the *local activity*. It should be mentioned that output nodes of a *local activity* do not show any further object flow in the *local choreography*. This means the flow of sending the *information envelope* to the other party must be specified in a UMM *business transaction*.

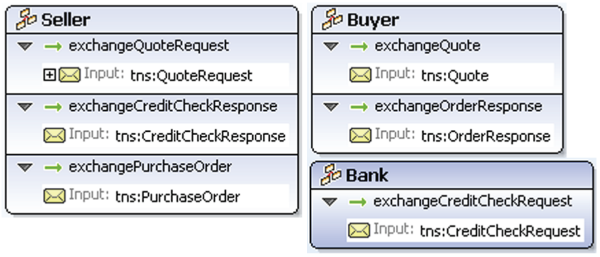
The stereotype *call nested transaction* is another kind of action in the flow of a *local activity*. It is as a special kind of the UML *call action behavior* used to call a flow in another *local activity* of *local choreography*. However, it should be noticed that *call nested transaction* is only used for calling a single *initiating/reacting activity*. If a whole collaboration is nested, the *call nested collaboration* concept is used. This one calls another *local process*.

5 Transformation to BPEL

The Business Process Execution Language (BPEL) has become the most dominant language for specifying workflows in the area of web services. Thus, we prefer to map our UML models of local processes to BPEL. BPEL specifies a business process among web services. To be more precise, it is a flow among web services operations that are provided by one ore more business partners. These operations are defined as part of port types in one or more WSDL files.

Accordingly, we have to identify the port types and the operations that result from our UML models. A port type has to be created not only for the owner of the process, but also for the partners he is communicating with. It follows that we create a port type for each authorized role. In our `order management` example, we create a port type for the process owner `seller` and for the authorized roles `buyer` and `bank`.

In a next step we have to assign appropriate operations to the port types. The operations assigned to the port type of the process owner correspond to the activities stereotyped as *receive business information* in the *local process*. The input message of each operation corresponds to the *information envelope* classifying the object node that is connected to the *receive business information* activity. In our example, `exchange quote request`, `exchange purchase order` and `exchange credit check response` are assigned to the `seller`'s port type. The *send business information* activities result in operations that are located in the port type of one of the other authorized roles. The corresponding *authorized role* is calculated by referring to the other role in the *business transaction* on which the *local activity* (*initiating/reacting activity*) containing the *send business information* activity is based. In our example, the operations `exchange quote response` and `exchange order response` are assigned to the `buyer`'s port type. Whereas, the operation `exchange credit check request` is part of the `bank`'s port type. The resulting port types of the `seller`, the `buyer` and the `bank` are depicted in figure 8. For a better readability we omit to show the XML code of the WSDL files, but present a graphical equivalent.


Seller
exchangeQuoteRequest
Input: tns:QuoteRequest
exchangeCreditCheckResponse
Input: tns:CreditCheckResponse
exchangePurchaseOrder
Input: tns:PurchaseOrder

Buyer
exchangeQuote
Input: tns:Quote
exchangeOrderResponse
Input: tns:OrderResponse

Bank
exchangeCreditCheckRequest
Input: tns:CreditCheckRequest

Figure 8. Port Types: Seller, Buyer, Bank

It is important to note that the calculation of the port types and their operations is based on some conventions on the inter-organizational layer that must be considered in the local processes. A two-way business transaction involves sending a *requesting information envelope* and returning a *responding information envelope*. The exchanges of these two envelopes are transformed into two asynchronous message exchanges to avoid blocking on each partner's side.

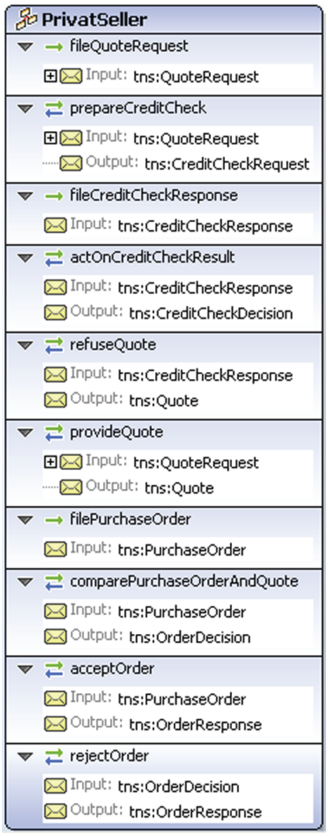

PrivatSeller
fileQuoteRequest
Input: tns:QuoteRequest
prepareCreditCheck
Input: tns:QuoteRequest
Output: tns:CreditCheckRequest
fileCreditCheckResponse
Input: tns:CreditCheckResponse
actOnCreditCheckResult
Input: tns:CreditCheckResponse
Output: tns:CreditCheckDecision
refuseQuote
Input: tns:CreditCheckResponse
Output: tns:Quote
provideQuote
Input: tns:QuoteRequest
Output: tns:Quote
filePurchaseOrder
Input: tns:PurchaseOrder
comparePurchaseOrderAndQuote
Input: tns:PurchaseOrder
Output: tns:OrderDecision
acceptOrder
Input: tns:PurchaseOrder
Output: tns:OrderResponse
rejectOrder
Input: tns:OrderDecision
Output: tns:OrderResponse

Figure 9. Port Type: Private to Seller

Accordingly, an operation is always added to the port type of the *authorized role* that receives the *information envelope*. Furthermore, it is necessary that the exchange of an *information envelope* is named exactly the same in the local processes of the participating *authorized roles*. In other words, the *receive business information* activity in the local process of the receiver must have the same name as the *send business information* activity in the local process of the sender. This requires a naming convention. Therefore, all *receive business information* and *send business information activities* have to be named '`exchange <information envelope>`'. This naming convention was already followed in the local process of the seller in figure 6, e.g. the first *receive business information* activity is called `exchange quote request`.

So far, we have discussed the operations resulting from communications with business partners. If a *local process* models a local choreography there are no other operations involved. However, if a *local process* models an orchestration, there will be operations that are visible to the owner of the local process, but not to the outside world. This is exactly the purpose of the *private actions* in *local processes*. Thus, the *private actions* are transformed to operations that are located in port types that are private to the process owner. The name of the operation corresponds to the name of the *private action*. However, the *private actions* may reflect already existing operations on different port types. In order to ensure a flexible assignment of the operations to different port types, we have decided to add a *tagged value* to the *private action* in the UML profile specifying the port type. Furthermore, a *private action* requires some input and may produce some output. One option would have been to assign object nodes to *private actions* in the UML profile and to model also the object flow in the *local process*. However, experience shows that this leads to overloaded activity diagrams that are hard to grasp. Hence, we opted for the alternative to denote the input and the optional output by further tagged values assigned to the *private action*. We did not show the instantiation of the tagged values in the local process of the `seller` in figure 6, but the input and output of the *private actions* is visible in the operations of the port type that is private to the seller (figure 9). For reasons of simplicity, we have mapped all the *private actions* to the same port type.

Having created all the port types and their operations, it is necessary to orchestrate the flow of operations in a BPEL code according to the UML activity diagram of a local process. In order to ensure a well-defined mapping from UML activity diagrams to BPEL, one must not use all the flow concepts of UML activity diagrams. A mapping is guaranteed, if the business collaboration protocol and, consequently, the flow among initiating/reacting activities, as well as the flow within initiating/reacting activities is limited to the following basic control flow patterns [16]: sequence, parallel split, synchronization, exclusive choice, multiple choice, and simple merge. In this case the BPEL process can be created by basic activities and by the structured activities *sequence*, *switch* and *flow*.

The transformation from the local process of a UML activity diagram is mapped to the BPEL code as follows: The transformation from a sequence in UML to BPEL is rather trivial. Alternative paths in UML are transformed into *switch* activities in BPEL. Parallel UML paths are transformed into *flow* activities in BPEL. A *receive business information* activity of the *local process* is always inserted in BPEL as a predecessor of the activity to which it delivers an input. The subgraph of the behavior of a *call nested transaction* (or a *call nested collaboration*) is always inserted at the point of the *call nested transaction* (or of the *call nested collaboration*, respectively).

The transformation of the `seller`'s local process in figure 6 results in the BPEL structure of figure 10. Due to page limitations and for reasons of better readability we do not show the XML code. Each node of figure 10 corresponds to an operation. The character in the node reflects the port type: `seller` (S), `buyer` (B), `bank` (C) and `private to`

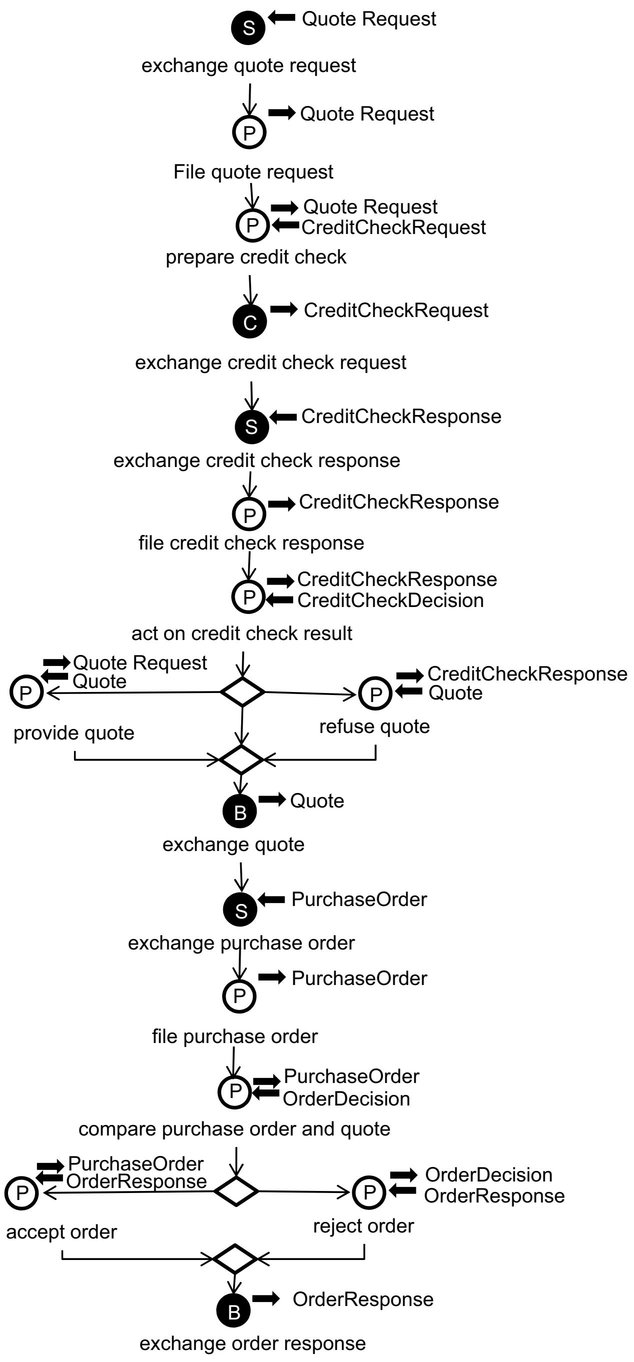


Figure 10. BPEL Process of the Seller

`seller` (P). The operations private to the seller are depicted by white nodes, whereas the operations with business partners are shown by black nodes. The input and output to the operations is marked close to the little arrows above the operations. Operations having only an input represent *receive* activities. In contrary, operations having an output or an output and an input represent *invoke* activities. The UML-like diamond presents a *switch* in the following activities.

6 Summary

In this paper we presented a model driven approach to inter-organizational business processes. We build up-on the UN/CEFACT modeling methodology (UMM). UMM is defined as a UML profile for modeling global choreographies. We used a simple order from quote example to demonstrate modeling a bi-lateral, global choreography by means of UMM. More complex and realistic UMM choreographies are part of all business requirements specifications developed for international trade by UN/CEFACT, as well as by national e-government frameworks, such as XOV in Germany, GovDex in Australia, and the Governments of Canada Strategic Reference Model (GSRM).

However, it is critical for an organization to model its own business processes and at the same time being in-line with business collaboration models the organization agreed up-on. For these reasons we provide a methodology extending the UMM for the purpose of modeling internal processes, i.e. local choreographies or orchestrations. Our approach extends the UMM by another set of stereotypes. We presented the first ideas on this extension in [1]. In this paper we define the exact meta model for the internal processes and the relationship of its stereotypes to the one of UMM. Our profile guarantees that the local choreography respects the commitments made in the global choreography - assuming that the global choreography is correct.

In order to execute the model of internal processes by the local IT systems it must be translated into a workflow language. For this purpose we define a mapping from our UML profile for internal processes to BPEL.

In order to support our approach, we created the UML profile definition for the commercial UML tool Enterprise Architect. Accordingly, all stereotypes are then built into Enterprise Architect and the profile may be used together with our Enterprise Architect Add-in for the UMM foundation module [4]. Nevertheless, it should be noted that our approach sits on top of the UML meta model and, thus, any other UML tool may be used to create compliant models. This fact helped also in the early stages of developing the UML profile: The Australian GovDex project reported the need for modeling UMM-compliant internal processes. We offered a preliminary version of our profile for local choreographies. The feedback loops with GovDex helped a lot in the evaluation and the fine tuning of our approach.

References

[1] B. Hofreiter. Extending UN/CEFACTs modeling methodology by a UML profile for local choreographies. *Journal on Information Systems and E-Business Management*, 0(0), 2008. to appear, online first: 10.1007/s10257-008-0083-3.

[2] B. Hofreiter and C. Huemer. Transforming UMM Business Collaboration Models to BPEL. In *OTM 2004 Workshops*, volume 3292 of *LNCS*, pages 507–519, 2004.

[3] B. Hofreiter, C. Huemer, P. Liegl, R. Schuster, and M. Zapletal. UN/CEFACT'S Modeling Methodology (UMM): A UML Profile for B2B e-Commerce. In *ER 2006 Workshops*, volume 4231 of *LNCS*, pages 19–31. Springer, 2006.

[4] B. Hofreiter, C. Huemer, P. Liegl, R. Schuster, and M. Zapletal. UMM Add-In: A UML Extension for UN/CEFACT's Modeling Methodology. In *Service-Oriented Computing - ICSOC 2007, Fifth International Conference*, volume 4749 of *LNCS*, pages 618–619. Springer, 2007.

[5] R. Khalaf. From RosettaNet PIPs to BPEL processes: A three level approach for business protocols. *Data Knowl. Eng.*, 61(1):23–38, 2007.

[6] B. Korherr and B. List. Extending the UML 2 Activity Diagram with Business Process Goals and Performance Measures and the Mapping to BPEL. In *ER 2006 Workshops*, volume 4231 of *LNCS*, pages 7–18. Springer, 2006.

[7] G. Kramler, E. Kapsammer, G. Kappel, and W. Retschitzegger. Towards Using UML 2 for Modelling Web Service Collaboration Protocols. In *Interoperability of Enterprise Software and Applications (INTEROP-ESA'05)*, pages 227–238. Springer, 2005.

[8] B. List and B. Korherr. A UML 2 Profile for Business Process Modelling. In *ER 2005 Workshops*, volume 3770 of *LNCS*, pages 85–96. Springer, 2005.

[9] J. Mendling and M. Hafner. From Inter-organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *OTM 2005 Workshops*, volume 3762 of *LNCS*, pages 506–515. Springer, 2005.

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

[11] C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003.

[12] M. Penker and H.-E. Eriksson. *Business Modeling With UML: Business Patterns at Work*. Wiley, 2000.

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

[14] D. Skogan, R. Gronmo, and I. Solheim. Web Service Composition in UML. In *8th IEEE Enterprise Distributed Object Computing Conference (EDOC)*, pages 47–57. IEEE, 2004.

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

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