Deriving business service interfaces in Windows Workflow from UMM transactions - long version - ⋆

Marco Zapletal

Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria
marco@ec.tuwien.ac.at

Abstract. Modeling inter-organizational business processes identifies the services each business partner has to provide and to consume as well as the flow of interactions between them. A model-driven approach to inter-organizational business processes allows abstracting from the underlying IT platform and, thereby, guarantees to survive changes in technology. UN/CEFACT's Modeling Methodology (UMM), which is defined as a UML profile, is currently one of the most promising approaches for modeling platform-independent business collaborations. However, well defined mappings to most of the current state-of-the-art candidate platforms are still missing. A candidate platform of growing interest is the Windows Workflow Foundation (WF). In this paper, we outline a mapping from the basic UMM building blocks, i.e. business transactions, to business service interfaces (BSI) implemented in WF. Our approach shows that UMM business transaction models greatly abstract from the complexity of a BSI.

1 Motivation

Business-to-Business (B2B) electronic commerce presupposes the integration of inter-organizational systems. In recent years, service-oriented computing has become the next evolutionary step in connecting autonomous enterprise systems. Service-orientation is considered as an enabler for aligning services in a business sense with their technical implementation. If each business partner, however, defines the service interactions with other partners in isolation, interoperability is unlikely. Consequently, B2B requires an approach that describes business collaborations from a global perspective. Furthermore, business logic should be abstracted from implementation specifics. This is in line with the Open-edi reference model [1] separating business aspects from technical details for implementing business transactions. UN/CEFACT's Modeling Methodology (UMM) is a UML-based modeling language following this approach. It describes business

⋆ The short version appeared in the Proceedings of the International Conference on Service Oriented Computing (ICSOC'08), Springer LNCS collaborations from a neutral point of view by specifying the services each partner has to provide and to consume as well as the flow between them. A UMM model is not bound to any specific implementation platform. However, in order to realize a business service interface based on UMM, mappings to specific target platforms have to be provided. A typical candidate are Web Services based on the Business Process Execution Language (BPEL), which we already discussed in [2]. In addition to the pure Web Services stack, the Windows Workflow Foundation (WF) is a strong candidate for the implementation of business service interfaces. In this paper we show the transformation of UMM business transactions to business service interfaces (BSI) realized in WF. The flow within the BSI is defined using WF's sequential workflow language. The interface to the workflow, however, is composed of well-defined business services implemented using Web Service specifications. Our model-driven approach yields three major benefits: First, business partners may agree on a global model serving as a kind of contract on the process. Second, the resulting business service interfaces of collaborating roles are complementary to each other ensuring their interoperability. Third, the graphical UMM representation abstracts from the complexity of a business service interface, which becomes evident in the workflow model.

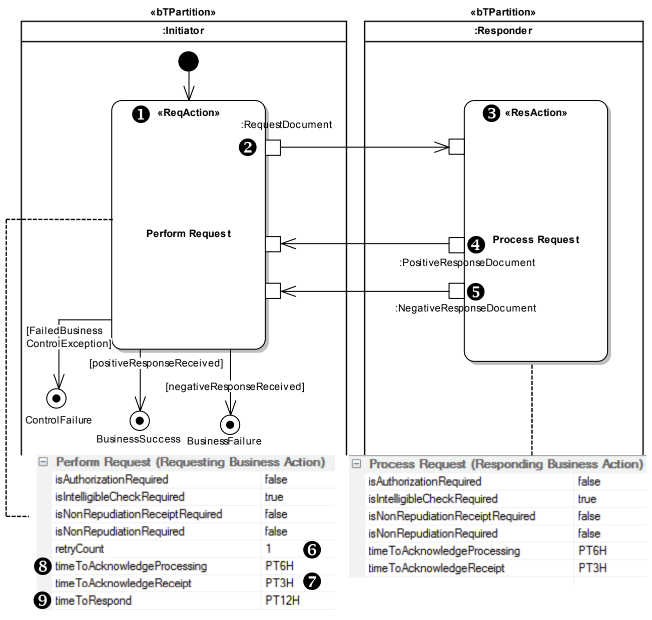
2 UMM Business Transactions at a glance

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. The latest specification is UMM 1.0 (built on UML 1.4.2) [3], which we co-authored. Currently, we are developing UMM 2.0 [4], defined on top of UML 2.1. In this paper, we already use UMM 2.0 as the base for our mapping.

UMM customizes the general purpose language UML to the specific needs of B2B. It is a development process for the definition of business collaborations between two or more business partners. A UML business collaboration model consists of three views - the business requirements view (BRV), the business choreography view (BCV), and the business information view (BIV). The BRV gathers the requirements of the to-be designed business collaborations. In the BCV, the business process analyst designs the flow of the business collaborations based on the requirements collected before. 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 transactions. A business transaction describes the exchange of a business document and an optional response between exactly two participants. Business documents are modeled in their own view - the BIV - in order to be re-usable across multiple business transactions. In this paper, we concentrate on the basic building blocks of UMM - the business transactions.

A business transaction is used to align the information systems of the collaborating business partners. Aligning the 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 generic request document and a response document - either communicating a positive or a negative business intention.



Perform Request (Requesting Business Action)	
isAuthorizationRequired	false
isIntelligibleCheckRequired	true
isNonRepudiationReceiptRequired	false
isNonRepudiationRequired	false
retryCount	1
timeToAcknowledgeProcessing	PT6H
timeToAcknowledgeReceipt	PT3H
timeToRespond	PT12H

Process Request (Responding Business Action)	
isAuthorizationRequired	false
isIntelligibleCheckRequired	true
isNonRepudiationReceiptRequired	false
isNonRepudiationRequired	false
timeToAcknowledgeProcessing	PT6H
timeToAcknowledgeReceipt	PT3H

Fig. 1. UMM Business Transaction example

In UMM, a business transaction is represented as a UML activity graph following always the same pattern. This pattern is built by a 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 partitions* each of them representing an *authorized role* performed by a business partner. The initiating business partner performs exactly one *requesting business action* and the responding business partner performs exactly one *responding business action*. While the object flow from the *requesting* to the *responding business action* is mandatory, object flows in the opposite direction are optional. The exchanged business information is represented by a *requesting* or a *responding information pin*.

In respect to special business needs, UMM distinguishes two types of one-way business transactions and four types of two-way business transactions. The difference between one-way transactions is as follows: if the business information sent is a formal non-repudiable notification, the transaction is called *notification*. Otherwise the business transaction is called *information distribution*. Two-way transactions are distinguished as follows: 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 [1]. 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. The interested reader is referred to the specification [4]. The business transaction types and their tagged values have proven to be useful in RosettaNet [5].

Figure 1 shows a generic example of a two-way business transaction, which serves as the basis for the mapping to Windows Workflow. By generic we mean that we do not use a specific real-world example (e.g., order product, book flight, etc.), but demonstrate the concepts using a simple and abstract example. We use generic terms like initiator and responder for role names or request and response document for the exchanged information. According to figure 1, the `initiator` performs the requesting role and requests a business decision in terms of a response document from the `responder` who takes on the reacting role. The initiator starts a `perform request` and creates a `request document` that triggers the `process request` action of the responder. Having reached this point of the business transaction, the `perform request` action is still alive, because it is waiting for a response document from the responder. Afterwards, the process request action generates either a `positive response document` or a `negative response document` based on an internal decision. Whatever response document is created, is eventually returned to the initiator.

Finally, the `perform request` action on the initiator's side processes the response. In case of a `positive response document`, the business transaction is considered as successful. Otherwise, if a `negative response document` was provided, the business transaction resulted in a failure. Aside from business failures, a business transaction may also involve technical failures: if one of the participants encounters a time-out - i.e., an acknowledgment or a business document is not received in time - he must signal his partner a *time-out exception*. If a message is scrambled during the exchange and, hence, is not processable, a *failed business control exception* must be issued. Similarly, a *failed business control exception* is sent if the *retry count* is elapsed and the execution of the business transaction failed. This implies, that a business service interface supporting a UMM business transaction has to be prepared to receive exceptions at any time during the regular process flow. Furthermore, it is always the task of the initiator of a business transaction to monitor the available retry count and abort the business transaction if necessary.

3 Introduction to Windows Workflow

The Windows Workflow Foundation (WF) is an extensible framework for building workflow-centric applications in .NET. The WF makes workflows first-class citizens of an application by having a clear business process focus. Workflows may either reside within an application or may communicate with other applications by providing or consuming services.

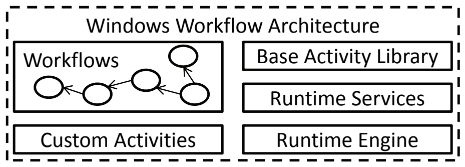


Fig. 2. Windows Workflow (WF) Architecture

The general architecture of WF is shown in figure 2. A more detailed introduction to the various components is found in [6]. In general, workflows are executed within the workflow runtime, which may be hosted in any type of .NET application. The runtime engine's functionality is expendable through runtime services. For example, developers may add persistence or tracking capabilities to their workflows by applying the corresponding services, which come out of the box. The basic building blocks of a workflow are activities. WF provides a set of standard activities for common purposes such as flow control (*parallel*, *if/else*, *sequence*, etc.), event handling (*listen activity*, *event handling scope activity*, etc.), or distributed communication (*send activity*, *receive activity*, etc.). The most primitive activity - the *code activity* - allows to quickly add any custom .NET code. However, in order to re-use custom code across workflows, developers should rather implement specific business logic by the concept of *custom activities.* Custom activities encapsulate a unit of work that serves a specific business purpose.

WF supports two flow paradigms for the definition of workflows - *sequential workflows* and *state-machine workflows*: The former define a prescribed series of execution. A sequential workflow, however, may contain loops, branches with conditions and may listen to external events. The underlying flow model is similar to the one of BPEL. Sequential workflows are typically used in automated processes, where the workflow is in control. On the other hand, state-machine workflows are event-driven. The developer creates a workflow as a set of states and the allowed transitions between them. Events trigger the execution of activities and/or the transition to other states. Thus, the events determine the execution path of the workflow. The target of a transition may be any other state within the workflow. State-machine workflows are often used for business processes that involve much human interaction, where the execution order heavily depends on user input. Furthermore, state-machine workflows are used if the process is mainly driven by external events or if all possible execution paths are hard to describe within a sequential workflow.

4 The transformation process

In this section, we elaborate on the transformation from a global UMM business transaction model to partner-specific BSI's implemented in WF. Since examples facilitate understanding, we detail the mapping by means of our generic business transaction example introduced in figure 1. For describing the mapping, we concentrate on the initiators's part of the process. Evidently, the responder's business service interface is complementary to the one of the initiator. In other words, when the initiator invokes something, the responder has to receive something. Thus, the order of sending and receiving business documents has to be reversed. The same applies for the handling of acknowledgments.

4.1 The regular process flow.

In the following, we detail the mapping by means of figure 1 and figure 3. The latter one depicts the derived business service interface for the initiator implemented in WF. The WF process is defined as a sequential workflow resulting in 11 major steps (A-K), whereby steps F to J contain nested activities. If not explicitly noted else, all used activity types are contained in WF's basic activity library.

Step A: Interacting with the business application. At the very beginning, the initiator's BSI receives the request document from the business

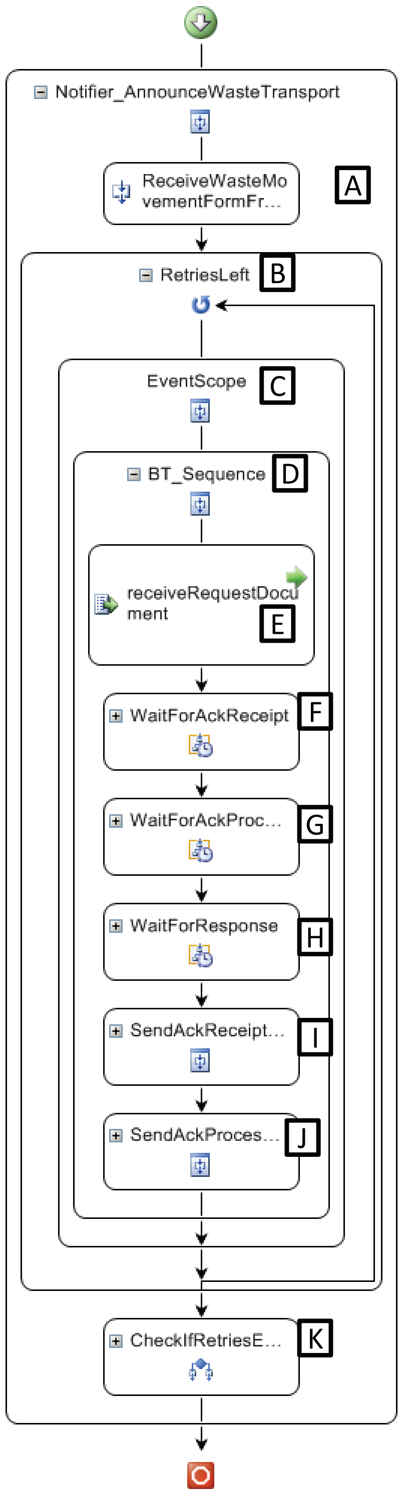


Fig. 3. The initiator's business service interface implemented in WF

application. Receiving the document is implemented by a *handle external event* activity. This presupposes that the business application is implemented in .NET as well. If this is not the case, the *handle external event* activity may be substituted by a *receive activity* for enabling cross-platform communication - for example realized by Web Service calls.

The *receive activity* and the *send activity* integrate the Windows Workflow Foundation with the Windows Communication Foundation (WCF). The WCF is the .NET approach to unify and simplify distributed communication. The WCF follows the 'ABC' model by separating address, binding, and contract. A *receive activity* is bound to a .NET interface known as service contract. The service contract specifies, which operations a service has to expose. WCF services may be deployed using several bindings including the WS-I Basic Profile 1.1, .NET Remoting, and Microsoft Message Queuing. One or more endpoint addresses may be declaratively specified for a WCF service. Moreover, the same WCF service may be exposed at different endpoint addresses using different bindings.

Step B: Checking the available retries. As we know from section 2, the initiator has to re-start a business transaction in case of time-outs. A time-out occurs if a business document or a business signal is not received within an expected time frame. The maximum amount of retries is specified by the tagged value *retry count*. In our example, the retry count is 1 (see (6) in figure 1), which results in a maximum of two attempts. In WF, we use a *while activity* to repeat the execution of the business transaction if required. Consequently, the *while activity* (B in figure 3) has to be executed until either the retry count is exceeded or the business transaction is considered as successful. Thus, we define the loop's condition as `retryCount >= 0 || businessTransactionSuccessful`, whereby both parameters are defined as normal .NET variables within the workflow. In case the business transaction is successful, the last action within the while loop sets the variable `businessTransactionSuccesful` to *true* (step H).

Steps C and D: Listening to business signals during the regular process flow. The only activity within the *while activity* is an *event handling scope activity*. This activity type allows to act upon events concurrently to the execution of the regular process flow. As outlined in section 2, business partner's may receive time-out exceptions or failed business control exceptions from their counterpart at any time during the course of a business transaction. Consequently, we use the *event handling scope activity* for receiving and processing business signals concurrently to the regular process flow. The *event handling scope activity* may have several event handlers attached - one for each event (i.e., business signal). We discuss event handlers and the actions they trigger in sub section 4.2. Within the *event handling scope* the *sequence activity* (D in figure 3) serves as a container for the activities realizing the message exchange with the responder's BSI (steps E to J).

Step E: Sending the request document. Step E communicates the request document ((2) in figure 1) from the initiator's (1) to the responder's BSI (3). On the initiator's side, the service call is implemented using the *send activity*. Note, that `receive request document` (E in figure 3) is indeed a *send activity*, which refers to the operation offered by the responder. The call is performed asynchronously (denoted by the single arrow on the right hand side of the *send activity* (E)), which means that the workflow continues immediately. The semantics of an asynchronous operation call by a *send activity* correspond to a truly fire-and-forget behavior. This entails that the client does not even receive a fault message from the service in case of an exception.

This behavior is in line with the semantics of asynchronous UMM business transactions patterns. Thereby, business document exchanges are completely asynchronous in order to avoid blocking behavior of business service interfaces. Nevertheless, interacting business service interfaces share the same understanding about the state of a business document exchange by communicating business signals as shown in the following steps.

Step F: Waiting for the acknowledgment of receipt. After sending the request document, the initiator waits for a business signal of type *acknowledgment of receipt* from the responder's BSI. According to UMM business transaction semantics, an acknowledgment of receipt is issued after a received business document passes grammar-, schema-, and sequence validation. One may argue that these checks may be performed instantly at the receiving BSI and, hence, the receipt of the request document should be acknowledged synchronously. This may be applicable in a SOA, where service interfaces expect strongly typed messages according to a certain XML schema. In this case, grammar- and schema validation is implicitly performed at the receiving service interface by the underlying implementation technology. However, numerous established electronic data interchange (EDI) standards are not defined in XML Schema, but have their own non XML-based syntax.

Let's consider a business service interface that must be capable of processing the most prevalent EDI standard - EDIFACT. In EDIFACT, the content of a message is separated using plus and colon characters as delimiters: In this respect, the business service interface exposes service operations that take strings as input arguments. Consequently, the service interface is not able to validate grammar and schema of the input message upon receipt. Instead, after receiving the message, some internal functionality of the BSI has to parse the business document and has to perform a variety of checks to ensure its validity. Such checks require processing time - particularly if the business document comprises several EDIFACT messages or some received messages are already enqueued. In this respect, synchronous acknowledgments would lead to blocking behavior of the business service interface, which is evidently not desired in an e-business environment. Thus, implementing e-business transactions mostly requires the decoupling of business documents from their receipt acknowledgments.

Figure 4 shows the required activities of step F in detail. The initiator expects the acknowledgment of receipt from the responder's BSI to confirm that the business document passed the syntactical checks. The *listen activity* in step F has two branches. The left branch is activated when the initiator's business service interface receives the acknowledgment of receipt. If the acknowledgment, however, is not picked up within the agreed time frame, the right branch is activated. The *listen activity* is responsible for activating that branch, whose trigger event occurs first. The remaining branches are canceled.

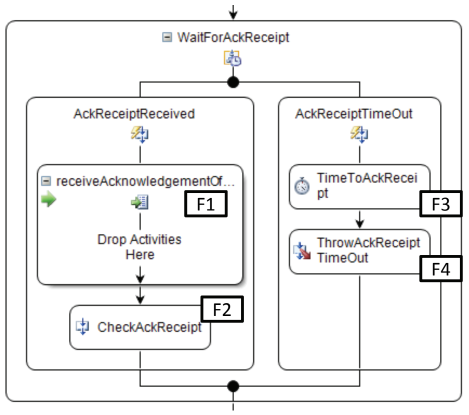


Fig. 4. Step F: Waiting for the acknowledgment of receipt

In order to expose a service for receiving the acknowledgment, the first activity in the left branch is a *receive activity* (F1). In figure 4, the *receive activity* is decorated with an arrow on the left hand side, which visually distinguishes it from the *send activity*. The *receive activity* is bound to an operation called `receive acknowledgment of receipt`. We define this operation in a service contract particularly for business signals. This service contract is not restricted to any business context and may be globally defined for business transactions. In this paper, we assume that at least the initiator and the responder bind their BSI's to this service contract for exchanging business signals.

The *receive activity* is followed by an activity that is responsible for checking the contents of the received acknowledgment (F2). Similar to the service contract for business signals, these checks may be identical for the same type of business signal across different business transactions. Thus, we propose to implement the required checks and constraints in a custom activity type. The *custom activity* `check ack receipt` may then be re-used in different WF business service interfaces. If, by any reason, checking the acknowledgment of receipt fails, the custom activity throws an exception that is further handled as outlined in section 4.2.

In the right branch, the first activity is a *delay activity* (F3). It monitors the agreed time to acknowledge receipt. In our example, this time limit amounts to a maximum of three hours (see (7) in figure 1). If exceeded, the *delay activity* triggers a time event which makes the *listen activity* activating the right branch (and consequently deactivating the left branch). In this case, the business transaction has to be re-started due to a time-out exception. In order to re-start the business transaction the current run has to be canceled and the condition of the *while activity*, which monitors the retries, has to be evaluated again. This behavior is accomplished by the *throw activity* (F4) following the *delay activity*. The *throw activity* actuates a *time-out detected exception* that is caught by a fault handler attached to the *while activity*. Handling faults is further discussed in section 4.2.

Step G: Waiting for the acknowledgment of processing. In this step, as shown in figure 5, the initiator expects an *acknowledgment of processing*. It confirms that the request document was successfully handed over to the responder's business application for further processing. This implies that the business document was delivered to the business application, where it passed additional validation rules. Validation by the business application covers checks on the semantic level - e.g., if an invoice is issued from a foreign company it must not contain VAT.

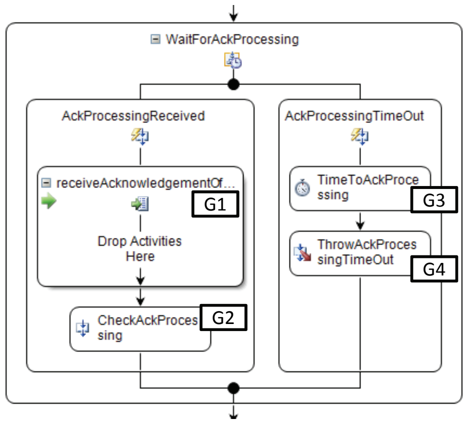


Fig. 5. Step G: Waiting for the acknowledgment of processing

In terms of the activity flow, handling acknowledgments of processing and their contingent time-outs is similar to the tasks processing acknowledgments of receipts. In the left branch, the steps G1 and G2 model the reception and the checks for a received acknowledgment, whereas the right branch (G3 and G4) handles the time-out. The agreed time-out monitored by the *delay activity* (G3) corresponds to the time to acknowledge processing as defined in (8) in figure 1. The acknowledgment of processing affirms the initiator that the responder is able to process the request document and will respond to it.

Step H: Waiting for the response document. Similar to steps F and G, waiting for the response document is implemented by a *listen activity* (H). In case of handling response documents two or more branches are required. Since we may expect time-outs for business documents as well, we define one branch for monitoring the maximum agreed time limit as agreed in the UMM model (see time to perform (9)) in figure 1). The cutout in figure 6 shows that the right branch keeps track of the time limit. If no response document is received within the agreed time to perform, the *delay activity* (H3) triggers a time event and the *throw activity* (H4) terminates the current cycle.

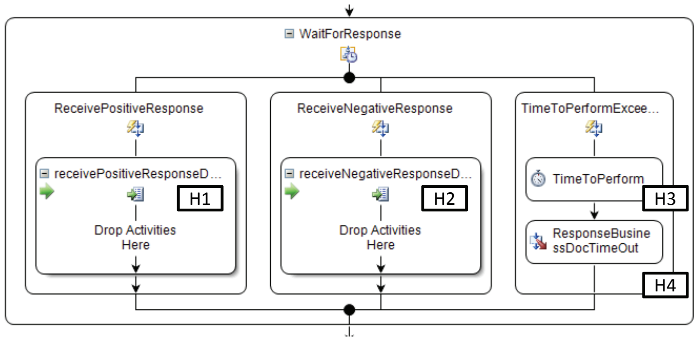


Fig. 6. Step H: Waiting for the response document

Two-way UMM business transactions support one to many response document types. Consequently, a business service interface requires one to many branches for the receiving business documents - one for each business document type. The example business transaction in figure 1 specifies two possible response documents - one representing a positive response (4) and the other one a negative response (5) to the request document. Accordingly, the business service interface requires two branches to receive both business document types (see figure 6). A positive response triggers the execution of the left branch containing the *receive activity* H1. Similarly, the *receive activity* H2 listens to negative response documents.

Step I: Sending the acknowledgment of receipt. Before the receipt of the response document is acknowledged, the business service interface needs to perform grammar-, schema-, and sequence validation. Since these are generic validation routines we employ again the concept of custom activities. If the business document passes the checks in step (I1), the *send activity* (I2) confirms the successful receipt by communicating an acknowledgment of receipt to the responder's BSI. A validation exception would raise a failed business control exception as further outlined in section 4.2.

Step J: Sending the acknowledgment of processing. After the proper receipt of the response document is affirmed, the initiator's BSI hands over the document to the business application for further processing. As outlined in the beginning, our example assumes that the business application hosts the business service interface. Therefore, we implement the communication between those systems using a *call external method activity* (J1). Once the business application is delivered, the business application verifies that the document is processable according to pre-defined business rules.

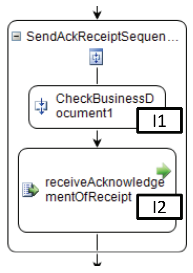


Fig. 7. Step I: Sending the acknowledgment of receipt

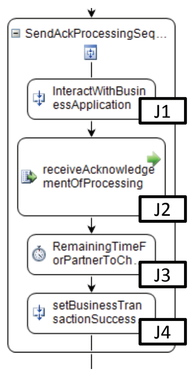

SendAckProcessingSeq...
InteractWithBusinessApplication
J1
receiveAcknowledgementOfProcessing
J2
RemainingTimeForPartnerToCh...
J3
setBusinessTransactionSuccess
J4

Fig. 8. Step J: Sending the acknowledgment of processing

If no exception is thrown by the business application, the BSI sends an acknowledgment of processing (J2) denoting that the verification was successful. The following *delay activity* (J3) keeps the business transaction alive in order to allow the responder to issue a time-out exception or a failed business control exception. The former is communicated by the responder, if the acknowledgment of processing is not received in time by its business service interface. The latter one is thrown, if an acknowledgment is received, which is not processable. How long the business transaction is kept alive is calculated by adding the *time to acknowledge processing* of the response document (10 in figure 1) to the time when the receipt of the response document was acknowledged. Finally, the custom activity J4 sets the variable `businessTransactionSuccessful` to true, so that the condition of the *while activity* (C) is not met any longer.

Step K: Checking the retry count. Before the business transaction is eventually finished, the business service interface must check if the *retry count* is not exceeded. Note, the loop continues until the retry count is equal or greater than zero. If the retry count is decremented to -1 at the end of the last attempt, the condition of the *while activity* is not met any longer and the control flow reaches step K. Therefore, the *if/else activity* in step K queries if the retry count is greater or equal to zero. If true, the business transaction was evidently successful and the execution of the business service interface finishes. Otherwise, a *retry count exceeded exception* is thrown and the business transaction failed. Due to space limitations, we do not include a figure for this step.

4.2 Event and fault handlers

Aside from executing the regular process flow, the business service interface must be capable of handling exceptional behavior. There are two concepts for managing deviations from the regular process flow: event handlers and fault handlers. The former ones deal with business signals communicated by the counterpart in the business transaction, whereas the latter ones handle exceptions that are raised through received business signals or that are detected locally.

Event handlers listen to events that may occur concurrently to the regular process flow. The occurrence of an event activates the respective event handler and executes the nested activities therein, but it does not cancel the regular process flow. In other words, both activity flows are executed concurrently. An event handler guarantees that participants may dispatch a time-out exception or a failed business control exception at any time. In WF, one or more event handlers may be attached to an *event handling scope activity* to listen for events beside executing the nested process flow.

Fault handlers are able to catch any type of .NET exceptions that are raised during the execution of a workflow. Exceptions may either be thrown within code (i.e., within an activity) or declaratively using the *throw activity.* Raising and handling exceptions in WF is similar to common object-oriented programming languages. This means, exceptions may be differentiated by their types and thrown and re-thrown across scopes until a proper fault handler is found. For supporting UMM business transactions, we use combinations of event and fault handlers to manage exceptional behavior in business service interfaces. According to the UMM business transaction semantics, the following faults may occur from the initiator's perspective:

EX-1: The responder sends a time-out exception. If the responder detects a time-out for a business document or a business signal on his side, he communicates a time-out exception to the initiator. For receiving time-out exceptions concurrently to the regular process flow, we attach an event handler to the *event handling scope activity* in step C in figure 1. The first activity of the event handler sets the event to which the handler is listening. In case EX-1, the first activity is a *receive activity* that is bound to an operation for picking up time-out exceptions. Next, a *throw activity* raises a time-out exception that is caught by a fault handler attached to the *event handling scope activity* in step D. Within the fault handler, there is a custom activity that decrements the retry count. The execution of the fault handler cancels the nested activities within the *event handling scope activity* (i.e., steps E to J). Since the *event handling scope activity* is enclosed by the *while activity* (B), the control flow continues with re-evaluating the loop's condition.

EX-2: The responder sends a failed business control exception. A failed business control exception is dispatched by the responder if he is not able to process a received business document or business signal. The initiator's BSI handles the receipt of failed business control exceptions similar to time-out exception. However, a failed business control exception does not decrease the retry count, but terminates the business transaction immediately. Hence, instead of decreasing the retry count within the fault handler, we re-throw the exception beyond the boundaries of the business transaction. This is required in a business collaboration that is composed of multiple business transactions.

EX-3: The responder does not answer in time. The initiator's BSI detects a time-out exception if the responder does not provide a business document or business signal within the agreed time frame. Time-outs may be detected by the initiator in steps F3, G3, and/or H3. If so, the *throw activity* following the respective *delay activity* raises a *time-out detected exception* to be caught by a fault handler attached to the *event handling scope* in step D. Within the fault handler, a *send activity* communicates the time-out exception to the responder and the retry count is decremented. As a consequence of activating the fault handler, the activity flow inside step D is canceled and the condition of the *while activity* (B) gets re-evaluated.

EX-4: Syntactic or semantic document checks fail. If the initiator's BSI is not able to process a business document or a business signal sent by the responder, he has to trigger a failed business control exception. Within the initiator's BSI a failed business control exception may be thrown by the custom activities `check ack receipt` (F2), `check ack processing` (G2), or by validation procedures of the business application invoked by the *call external method* activity in step I2. A failed business control exception is not caught within the *while activity* (B), but by the outermost sequence that encloses steps A to K. This is due to the fact that a failed business control exception terminates all attempts to execute

a business transaction.
EX-5: The retry count is exceeded. The condition of the *while activity* (B) does not evaluate true any longer, if either the business transaction has been successful (indicated by the internal boolean variable `businessTransactionSuccessful`) or if the retry count has been exceeded due to time-out exceptions. In the latter case, the business transaction is evidently not successful, but requires the communication of a failed business control exception to the responder. Consequently, before the business transaction eventually ends, the initiator has to check in step K if the retry count is exceeded or not. If true, the *throw activity* K2 raises a failed business control exception, which is handled similar as in case EX-4.

5 Related Work

In recent years, several business process modeling approaches have been proposed to abstract from implementation specifics. However, only a few of them focus on inter-organizational systems. Among those, some of them customize the general UML the specific needs of B2B. Kim suggests a UML 1.x-based modeling approach in [7]. This approach utilizes activity diagrams for modeling a collaborative process as a flow of transactions in order to create an ebXML compliant business process specification. In [8], the authors propose an approach based on UML 2 for depicting the choreography of Web Services. In their paper, the authors split the models into a layered architecture - collaboration, transaction, and interaction level, in order to compare these levels of granularity with the eCo framework. The work in [9] introduces the efforts of RosettaNet, which has been an early contributor to the development of UMM. The RosettaNet specifications employ UML activity diagrams to describe the flow of interactions between standardized partner interface processes (PIP). Other approaches to inter-organizational business processes specify their own syntax. The Business Process Modeling Notation (BPMN) - standardized by the Object Management Group (OMG) - gained a lot of interest in academia and industry. BPMN provides means to capture intra- as well as inter-organizational processes. BPMN incorporates aspects of already advanced modeling notations (e.g. UML activity diagrams [10], IDEF, [11], ebXML BPSS [12], RosettaNet [5], etc.). Another popoular process modeling technique are Event-driven Process Chains (EPC), which describe a business process as graph of events and functions. EPCs were developed by Scheer as part of the ARIS framework (Architecture of Integrated Information Systems). Traditionally, EPCs are applied for internal business processes. Some work exists, however, to extend EPCs for modeling inter-organizational business processes [13]. Another approach seeks after transforming EPC of internal processes into collaborative BPMN process diagrams.

Model-driven approaches to inter-organizational systems are proposed in [14] and [15]. Both take the concept of OMG's Model-Driven Architecture (MDA) into account. The former approach details a mapping from platform-independent models (PIM) specified in UML to platform-specific models (PSM) implemented in BPEL and Java. The latter approach rather concentrates on a conceptual architecture and design approach involving UML, BPMN, and EPCs. Both approaches, however, lack concepts in PIMs for supporting the business requirements and the business semantics of B2B as UMM does.

A lot of model-driven approaches focus on BPEL as a target platform. The work in [16] shows a derivation of Web Services artifacts in the health care domain. UML activity diagrams and sequence diagrams are semi-automatically mapped to BPEL processes and WSDL artifacts including security and transaction requirements. Due to the fixed business scenario, the approach considers the specific requirements of the health care domain and, hence, is difficult to re-use for e-business transactions. Other UML-based approaches for generating BPEL processes are given in [17] and [18]. Unlike UMM, both approaches do not consider the business context and the specific requirements on e-business transactions. In order to close the gap between the business process design and the business process implementation, OMG's Business Process Management Initiative (BPMI) standardizes the mapping from BPMN to BPEL. However, the mapping imposes some limitations on BPMN in order to map the graph-based BPMN processes to block-oriented BPEL processes. The work in [19] examines the BPMN-BPEL binding, and proposes an approach to overcome the limitations. A unified source model for a derivation of BPEL artifacts, is presented in [20]. The authors outline a mapping technique starting with a core subset of UML activity diagrams and BPMN.

Some mappings of UMM models to XML-based process specifications have already been introduced. The authors of [21] present a mapping from UMM to ebXML BPSS. A BPEL binding has been first introduced in [22]. In [2], we outlined a mapping from UMM business transaction to business service interfaces specified in BPEL. A short comparison between the BPEL and the WF mapping is given in the following section.

6 Conclusion and Future Work

UMM is a platform-independent modeling language for collaborative business processes. In order to deploy UMM models to specific target platforms corresponding mappings have to be defined. This paper contributes a UMM to Windows Workflow binding. The generated code is in fact ready to compile. Before executing the workflow, the following tasks remain: (i) declarative configurations of service endpoints, (ii) hosting the workflow, (iii) binding its internal interfaces to a business application.

As outlined in section 5, we have already proposed a UMM to BPEL binding in [2]. Both, the presented WF approach and the BPEL approach result in business service interfaces following the concepts of SOA. BPEL's evident advantage is being a jointly developed industry standard. Software vendors, however, have already started to implement their own platform-specific extensions to BPEL in their tools. This lowers the portability of BPEL of being a cross-platform standard. A major advantage of WF is the fact that the BSI may directly execute business logic. BPEL, however, is limited to the orchestration of services. Consequently, business logic - even if not intended to be re-used - has to be exposed as service. Furthermore, WF supports WS-* specifications like WS-Security or WS-ReliableMessaging out of the box by simply adapting endpoint configurations.

Future work will concentrate on the transformation of business collaborations to WF. Transforming a business collaboration includes the mapping of the composed business transactions as well as the flow between them. Unlike business transactions, business collaborations do not always follow the same pattern. Consequently, transforming a graph-based business collaboration to block-oriented sequential workflows may become challenging. Thus, we will combine WF state-machine workflows for defining the flow between business transactions with this approach for transforming business transactions.

References

1. ISO: Open-edi Reference Model. (1995) ISO/IEC JTC 1/SC30 Standard 14662.
2. Hofreiter, B., Huemer, C., Liegl, P., Schuster, R., Zapletal, M.: Deriving executable BPEL from UMM Business Transactions. In: Proc. of the IEEE Intl. Conf. on Services Computing (SCC 2007), IEEE CS (2007)
3. UN/CEFACT: UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module. (2006) Technical Specification V1.0.
4. UN/CEFACT: UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module. (2008) Internal Draft V2.0.
5. RosettaNet: RosettaNet Implementation Framework: Core Specification. (2002) V02.00.01.
6. Bukovics, B.: Pro WF: Windows Workflow in. NET 3.0. Apress (2007)
7. Kim, H.: Conceptual Modeling and Specification Generation for B2B Business Processes based on ebXML. SIGMOD Rec. **31**(1) (2002) 137–145
8. Kramler, G., Kapsammer, E., Kappel, G., Retschitzegger, W.: Towards Using UML 2 for Modelling Web Service Collaboration Protocols. In: Proc. of the First Intl. Conf. on Interoperability of Enterprise Software and Applications. (2005)
9. Damodaran, S.: B2B integration over the Internet with XML: RosettaNet successes and challenges. In: Proc. of the 13th Intl. World Wide Web Conference on Alternate track papers & posters, ACM (2004) 188–195
10. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H., Wohed, P.: On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling. In: Third Asia-Pacific Conf. on Conceptual Modelling, Australian Computer Soc., Inc. (2006)
11. Mayer, R., Menzel, C., Painter, M., deWitte, P., Blinn, T., Perakath, B.: Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report. (1995)
12. UN/CEFACT: UN/CEFACT - ebXML Business Process Specification Schema. (2003) Version 1.10.
13. Seel, C., Vanderhaeghen, D.: Meta-Model based Extensions of the EPC for Inter-Organisational Process Modelling. EPK 2005
14. Bzivin, J., Hammoudi, S., Lopes, D., Jouault, F.: Applying MDA approach to B2B applications: A road map. In: Proc. of the Workshop on Model Driven Development at the 18th European Conf. on Object-Oriented Programming. (2004)

15. Bauer, B., Müller, J.P., Roser, S.: A Model-Driven Approach to Designing Cross-Enterprise Business Processes. In: Proc. of OTM Conference Workshops. Volume 3294 of LNCS., Springer (2004)
16. Anzböck, R., Dustdar, S.: Semi-automatic Generation of Web Services and BPEL Processes - A Model-Driven Approach. In: Proc. of the 3rd Intl. Conf. on Business Process Management (BPM 2005). Volume 3649 of LNCS., Springer (2005)
17. Gardner, T.: UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. In: 1st Europ. Workshop on Object Orientation and Web Services. (2003)
18. Korherr, B., List, B.: Extending the UML 2 Activity Diagram with Business Process Goals and Performance Measures and the Mapping to BPEL. In: Proc. of the ER-Conf. on Conceptual Modeling (Workshop BP-UML'06), Springer (2006)
19. Ouyang, C., Dumas, M., ter Hofstede, A.H., van der Aalst, W.M.: From BPMN Process Models to BPEL Web Services. In: Proc. of the Intl. Conf. on Web Services (ICWS2006), IEEE CS (2006) 285–292
20. Ouyang, C., Dumas, M., Breutel, S., ter Hofstede, A.H.M.: Translating standard process models to bpel. In: Proc. of 18th Intl. Conf. on Advanced Information Systems Engineering (CAiSE 2006). Volume 4001 of LNCS., Springer (2006)
21. Hofreiter, B., Huemer, C., Kim, J.H.: Choreography of ebXML business collaborations. Information Systems and e-Business Management (ISeB) (2006)
22. Hofreiter, B., Huemer, C.: Transforming UMM Business Collaboration Models to BPEL. In: Proc. of the Intl. Workshop on Modeling Inter-Organizational Systems (MIOS). Volume 3292 of LNCS., Springer (2004)