Deriving Business Service Interfaces in Windows Workflow from UMM Transactions

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.

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. UN/CEFACT's Modeling Methodology (UMM) [1] is a UML-based modeling language following this approach. It describes business 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

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 498–504, 2008.

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.

Due to space limitations, we do not cover UMM business transactions in this paper, but refer to its long version that is published as a technical report [3].

2 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. 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.

We detail the mapping by means of figure 1 depicting 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

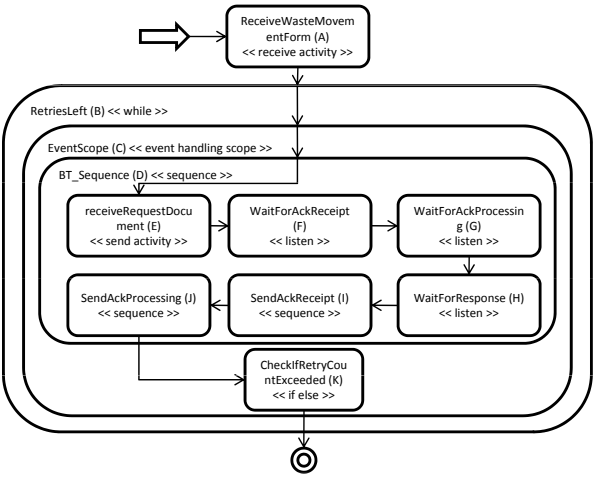

Fig. 1. The initiator's business service interface implemented in WF

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 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.

Step B: Checking the available retries. According to UMM business transaction semantics, 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 UMM. In WF, we use a *while activity* to repeat the execution of the business transaction if required. The *while activity* (B in figure 1) 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. In UMM business transactions, 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). Due to space limitations, we do not discuss event handlers in this paper, but refer to its long version [3]. Within the *event handling scope* the *sequence activity* (D in figure 1) 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 from the initiator's to the responder's BSI. On the initiator's side, the service call is implemented using the *send activity*. Note, that `receive request document` (E in figure 1) is indeed a *send activity*, which refers to the operation offered by the responder. The call is performed asynchronously, 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.

Figure 2 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.

In order to expose a service for receiving the acknowledgment, the first activity in the left branch is a *receive activity* (F1). 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.

In the right branch, the first activity is a *delay activity* (F3). It monitors the agreed time to acknowledge receipt. 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*. Please note that handling faults is not covered in this paper.

Step G: Waiting for the acknowledgment of processing. In this step the initiator expects an *acknowledgment of processing* as shown in figure 3. It

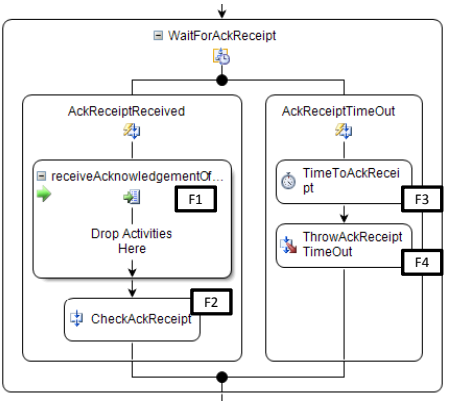

Fig. 2. Step F: Waiting for the acknowledgement of receipt

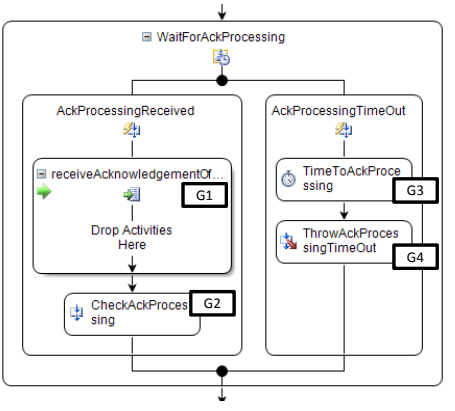

Fig. 3. Step G: Waiting for the acknowledgement of processing

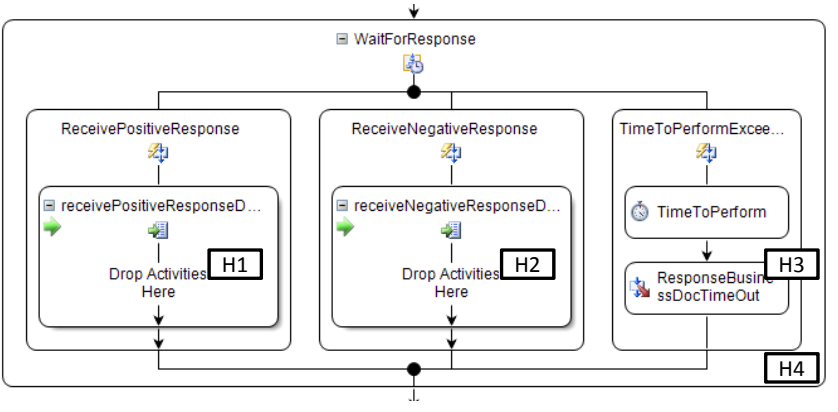

Fig. 4. Step H: Waiting for the response document

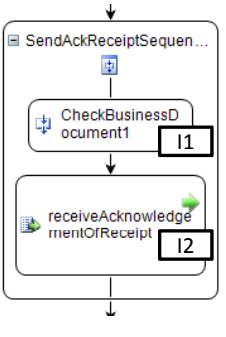

Fig. 5. Step I: Sending the acknowledgment of receipt

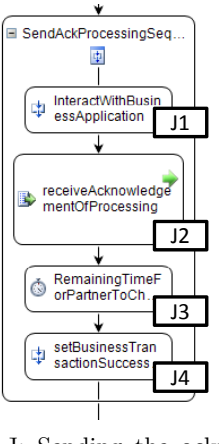

Fig. 6. Step J: Sending the acknowledgment of processing

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.

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 by the UMM business transaction. 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. The cutout in figure 4 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.

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. In our example, we specify two possible response documents - one representing a positive response and the other one a negative response to the request document. Accordingly, the business service interface requires two branches to receive both business document types (see figure 4). 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 the concept of custom activities. If the business document passes the checks in step (I1 in figure 5), the *send activity* (I2) confirms the successful receipt by communicating an acknowledgment of receipt to the responder's BSI.

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. We assume that the business application hosts the business service interface. Therefore, we implement the communication between those systems using a *call external method activity* (J1 in figure 6). Once the business application is delivered, the business application verifies that the document is processable according to predefined business rules.

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 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* has 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.

3 Conclusion

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.

References

1. UN/CEFACT: UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module, Public Draft V2.0 (2008)
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, Los Alamitos (2007)
3. Zapletal, M.: Deriving business service interfaces in Windows Workflow from UMM transactions - long version. Technical report, Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria (2008), `http://publik.tuwien.ac.at/files/PubDat_166624.pdf`