A State Machine executing UMM Business Transactions

Christian Huemer* and Marco Zapletal†
*Business Informatics Group, †Electronic Commerce Group
Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria
huemer@big.tuwien.ac.at, marco@ec.tuwien.ac.at

Abstract— UN/CEFACT's modeling methodology (UMM) is a UML profile for modeling global B2B choreographies. The current UMM version comprises three main views for describing a computation independent model from a neutral perspective. Currently, the UMM version is missing a platform independent model showing how each partner has to realize the message exchanges to support the agreed choreography. In this paper we derive such a platform independent model from a UMM business transaction - a key artifact of the computation independent model. The resulting model is based on a state machine describing the local view of a participating business partner. This state machine unambiguously defines how a business partner has to react on incoming messages and on message expected but not received.

I. INTRODUCTION

The model driven architecture (MDA) approach of the OMG distinguishes three different views of a system that result in different kind of models [1]: (1) The computation independent viewpoint focuses on the environment of the system and its requirements. The details of the structure and processing of the IT system are unspecified. It leads to a computation independent model (CIM) that is familiar to the practitioners of the domain under consideration who do not need to care how to realize the functionality of an IT system. (2) The platform independent viewpoint focuses on the operation of an IT system while hiding the details necessary for a particular platform. It leads to a platform independent model (PIM) exhibiting a certain degree of platform independence in order to be suitable for a number of different platforms of similar type. (3) The platform specific viewpoint extends the platform independent viewpoint with an additional focus on the details of using a specific platform by an IT system. It leads to a platform specific model (PSM) limited to a particular platform.

Traditionally, the MDA approach focuses on the development of application systems and their code. However, a MDA approach may also be used in developing inter-organizational systems describing how autonomous applications of different organizations have to interact. This is also in accordance with the Open-edi reference model, an ISO standard co-ordinating the collaboration between organizations [2]. Open-edi separates the what in the Business Operational View (BOV) from the how in the Functional Service View (FSV). The BOV covers the business aspects such as business information, business conventions, agreements and rules among organizations. The FSV deals with information technology aspects supporting the execution of business transactions. In MDA terms the BOV is a computation independent model and the FSV is a platform specific model.

Candidate platforms on the FSV are traditional electronic data interchange platforms such as UN/EDIFACT and ANSI X12, as well as more recent technologies like Web Services and ebXML. The BOV layer must focus on modeling the business processes of a business collaboration. Traditionally, business process modeling (cf. [3] [4]) concentrates on the orchestration of business processes internal to a company in order to create a customer value [5]. In a B2B environment the focus must be on the processes that happen in-between the organizations. The processes may be described from the point of view of a participating partner (i.e., a local choreography) or from an neutral and global perspective (i.e., a global choreography). The BOV must capture the committments between collaborating business partners. Similarily to a contract these commitments have to be described from a global perspective.

The UN/CEFACT modeling methodology (UMM) is designed to model choreographies of collaborating business partners from a global perspective. It is a UML profile defining a set of stereotypes, tagged values and constraints on the UML meta model for its special purpose. The UMM foundation module [6] - which we have co-edited - specifies three major steps: (1) *The business domain view (BDV)* is used to gather existing knowledge from stakeholders and business domain experts. In interviews the business process analyst tries to get a basic understanding of the business processes in the domain. The resulting use case descriptions of business processes are on a rather high level and are classified according to a predefined business operations map.

(2) Those business processes from the BDV that provide a chance for collaboration will be further detailed by the business process analyst in the *business requirements view (BRV)*. The business process analyst tries to discover interface tasks creating/changing business entities that are shared between business partners and, thus, require communication with a business partner. This enables to detect *business transaction use cases* which describe the requirements for basic interactions between business partners which involve an information exchange and an optional response. Furthermore, *business collaboration use cases* describe more complex processes which usually span over - and thus include - multiple business transaction use cases.

(3) The business transaction view (BTV) builds upon the BRV and defines a global choreography of information exchanges and the document structure of these exchanges. The basic building blocks of a global choreography called *business collaboration protocol* are *business transactions* which are further detailed in this paper.

The global choreography is best suited to define the committments between the partners. Nevertheless, each partner is responsible to implement a system that is compliant to the agreed choreography. This means that each partner must derive its local choreography and implement it on a certain platform. However, the handling of messages to be sent and to be received in the local choreography is independent of the messaging platform of Web Services, ebXML or UN/EDIFACT. Thus, we suggest to add another view to the UMM which defines the local choreography of message exchanges for each partner. In previous versions of UMM, the *business service view (BSV)* was used to depict the message exchanges, but it was removed, since the concepts based on sequence diagrams were inappropriate. In this paper we suggest to extend the UMM by a BSV that is based on state machines executing UMM business transactions.

II. UMM BUSINESS TRANSACTIONS

A business transaction is the basic building block to define a choreography of a business collaboration between collaborating business partners. We provide a more detailed analysis of business transactions and business collaboration protocols in [7]. Communication in a business collaboration is about aligning the information systems of the business partners. Aligning the information systems means that all relevant business objects (e.g. purchase orders, line items, etc.) are in the same state in each information system. If a business partner recognizes an event that changes the state of a business object, it initiates a business transaction to synchronize with the collaborating business partner. It follows that a business transaction is an atomic unit that leads to a synchronized state in both information systems.

We distinguish two very basic types of business transactions: In the first type, the initiating business partner reports an already effective and irreversible state change that the reacting business partner has to accept. Examples are the notification of shipment or the update of a product in a catalog. This is the case of a one-way business transaction, because business information (not including business signals for acknowledgments) flows only from the initiating to the reacting business partner. In the second type, the initiating partner sets the business object(s) into an interim state and the final state is decided by the reacting business partner. Examples include request for registration, search for products, etc. This is the case of a two-way transaction, because business information flows from the initiator to the responder to set the interim state and backwards to set the final and irreversible state change. In a business context, irreversible means that returning to an original state requires compensation by another business transaction [8]. E.g., once a purchase order is agreed upon in a business transaction a rollback is not allowed anymore, but requires the execution of a *cancel order* business transaction.

In UMM, a business collaboration protocol is specified as a flow of business transaction activties. The left hand side of Figure 1 shows a very simple example of a business collaboration protocol specifying a sequence of two business transaction activities. Of course, the number of business transaction activities is usually much higher leading to a much more complex flow including splits, mergers, loops etc. Each business transaction activity is refined by a business transaction which is another activity graph. This graph follows the strict pattern as shown on the right hand side of Figure 1. It is always built by exactly *two business actions*, a *requesting business activity* and a *responding business activity*. Each *business action* is performed by exactly one *authorized role* executed by a business partner. The assignment of the business action to an authorized role is realized by business transaction swimlanes. The requesting business activity is assigned to the initiator and the responding activity is assigned to the responder.

In UMM we distinguish two types of one-way transactions. If the business information sent is a formal non-repudiatable notification, the transaction is *called notification*. Otherwise the transaction is known as *information distribution*. Furthermore, there exist four different types of two-way transactions. If the responder already has the information available beforehand, it is a *query/response* transaction. 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, the next question is whether the transaction results in a residual obligation between the business partners to fulfill terms of a contract. If so, it is a *commercial transaction*. Otherwise it is a *request/response* transaction. These types of business transactions cover all known legally binding interactions between two decision making applications as defined in Open-edi [2]. They have proven to be useful in RosettaNet [9].

The different types of business transaction patterns differ in the defaults for the tagged values that characterize *business actions*: *is authorization required, is non-repudiation required, time to perform, time to acknowledge receipt, and time to acknowledge processing*. The values for *is non-repudiation of receipt required* and for *retry count* are only defined for the *requesting business activity*. Most of these attributes are self-explanatory. An acknowledgment of receipt is usually sent after grammar validation, sequence validation, and schema validation. However, if the *is intelligible check required* flag is set to *false*, the acknowledgment is sent immediately after receipt without any validation. An acknowledgment of processing is sent after validating the content against additional rules to ensure that the content is processable by the target application. It should be noted that both kinds of acknowledgments are business signals and are not acknowledgments on the network level realizing reliable messaging. *Retry count* is the number of retries in case of control failures.
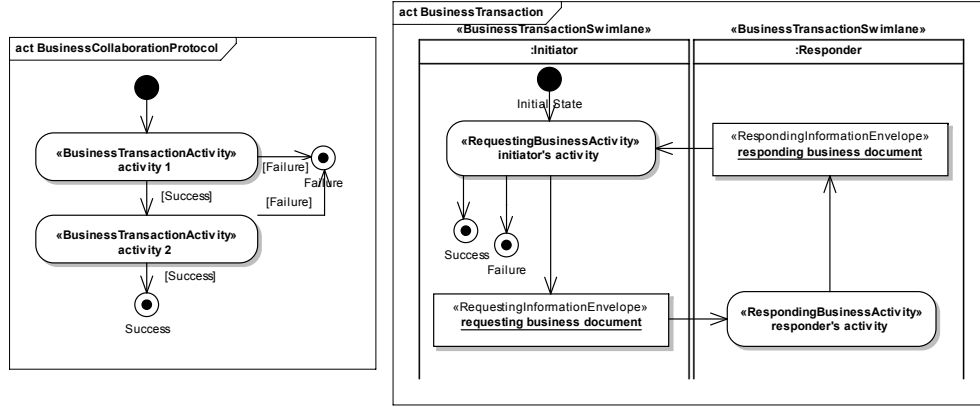
Fig. 1.   UMM Business Collaboration Protocol and Business Transaction

In a one-way transaction business information is exchanged only from the *requesting business activity* to the *responding business activity*. In case of a two-way transaction the *responding business activity* returns business information to the *requesting business activity*. The exchange of business information is shown by an object flow. One *business action* sets an *information object flow state* that is consumed by the other *business action*.

An *information object flow state* refers to an *information envelope* exchanged between the *business actions*. The *information envelope* is characterized by three security parameters: *is confidential, is tamper proof,* and *is authenticated*. The structure of the information envelope's content - which is not detailed in this paper - is modeled in a class diagram that is based on UN/CEFACT core components [10].

III. MAPPING BUSINESS TRANSACTIONS TO STATE MACHINES

In this section we transform the global choreography of UMM business transactions to a local choreography describing the message exchanges of business documents and business signals. UMM business transactions are described by a rather simple pattern. This is also due to the fact that the activity graph shows only the exchange of business documents, whereas the business signals representing acknowledgements are described by tagged values. Thus, the local choreography of a partners interface is not only a sequence of sending and receiving business documents. It requires a more complex choreography that has to reflect a reliable exchange of business documents leading to well-defined business states even in case of failures. In this section we propose to describe this complex choreography by means of a state machine that acts on incoming and outgoing messages. We use a state machine, because it is best suited to describe the valid states of a business partner interface and the events causing state transitions - finally leading to a success or failure of the business transaction.

The actions to be carried out by a business partner interface mainly depend on the instantiation of the tagged values of requesting/responding business activities. This instantion depends on the UMM transaction type. Due to space limitations we do not elaborate state machine representations for each of the six transaction types. Instead, we demonstrate the business partner interface by the most complex pattern - the commercial transaction. This pattern specifies a two-way transaction requiring all kinds of acknowledgements. State machines reflecting other patterns are easily constructed by omitting steps that are not required due to lower security requirements.

A. The initiator's part

A UMM business transaction takes place between two authorized roles. Each role must implement its own business partner interface. It follows, that a UMM business transaction results in two state machines each describing a business partner interface. Figure 2 shows the system specification for the initiator's part of a commercial transaction. One should note, that due to space limitations we shortened identifiers in the figure (e.g., *ResBusinessDoc* instead of responding business document, *AckReceipt* instead of acknowledgement of receipt, etc.).

A business transaction is a unit of work represented by a composite state. It is started when the business partner interface receives the requesting business document from the application. It may be re-initiated due to time-out exceptions. This means that the initiator has to restart the business transaction if an acknowledgement of receipt/processing or a response document is not received in time. The maximum numer of restarts is defined by the retry count. Thus, the first state is *checkRetryCount*. Its activity *checkIfRetryCountLeft* audits the remaining retries to re-initiate the transaction in case of a previous failure. If the retry count is equal or greater than zero the system proceeds with the transmission of the request. When a business transaction is initiated for the first time the check is always passed, because the initial retry count must at least be zero. In case of re-initiating the business transaction with no remaining retries the system transitions to state *sendFailedBusinessControl_Outgoing*. In this state, the
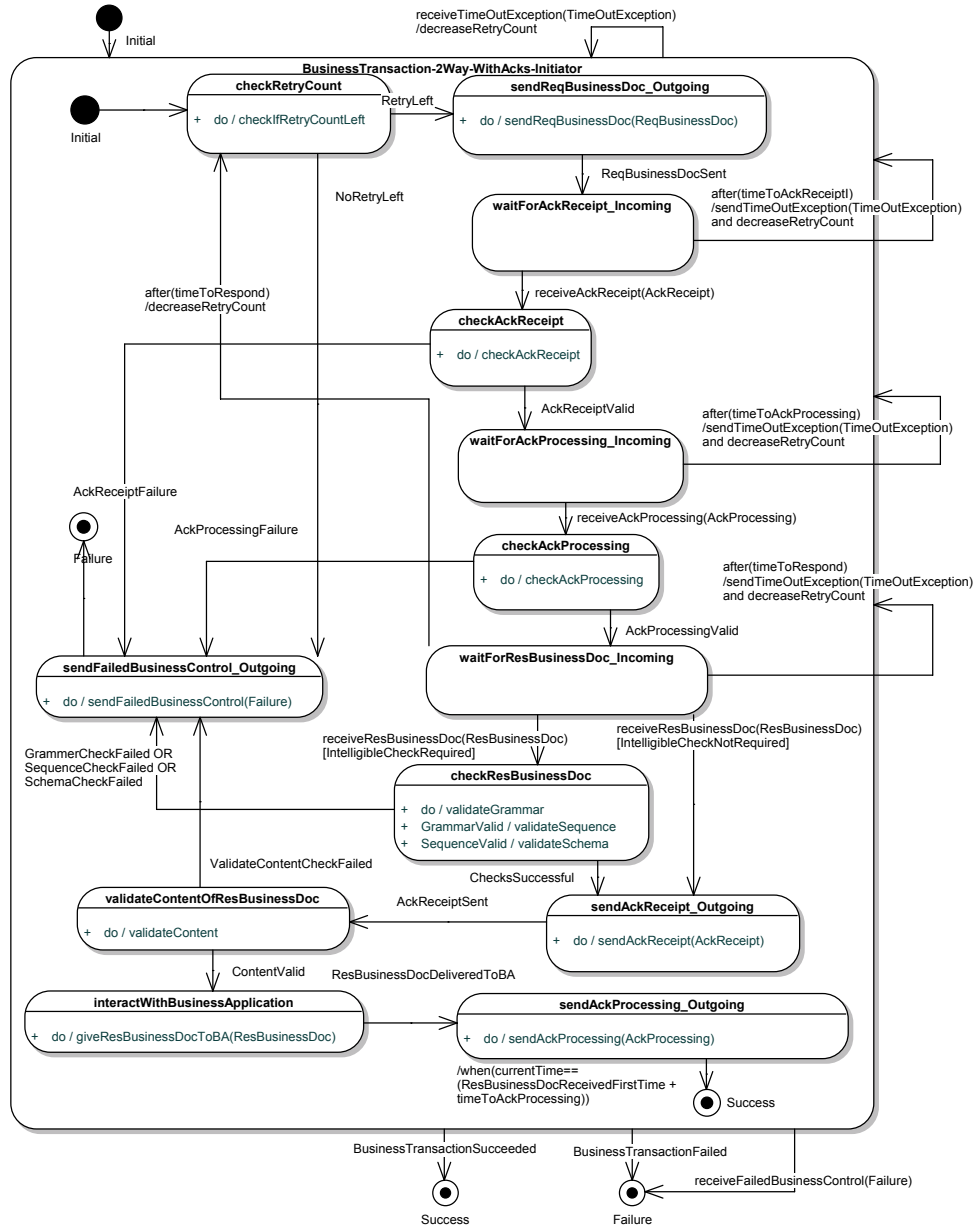
Fig. 2. State machine showing the initiator's part of a commercial transaction

system issues a notification of failed business control to the partner's system and exists the transaction with a failure.

If retries are left, a signal event *RetryLeft* fires the transition to state *sendBusinessDocument_Outgoing*. Within this state the initiator transmits the requesting business document by invoking the operation *sendReqBusinessDoc(ReqBusinessDoc)*.

After sending the document we reach the state *waitForAckReceipt_Incoming*. In this state the initiator expects that the responder acknowledges the receipt of the requesting business document sent before. The system remains there until either of the following two events occurs: (1) If the initiator receives no acknowledgement within the agreed time to acknowledge receipt, a time event specified by *after(timeToAcknowledgeReceipt)* eventuates and executes an

activity that issues a time-out exception and decreases the available retries. The transition activated by the time event re-enters the composite state of the business transaction, which begins again with *checkRetryCount*. (2) The initiator receives the acknowledgement of receipt. This results in the call event *receiveAckReceipt(AckReceipt)*.

If the acknowledgement of receipt is picked up, the initiator enters state *checkAckReceipt*. The activity *checkAckReceipt* checks the acknowledgement's content. A failure yields to an *AckReceiptFailure* event and the system transitions to *sendFailedBusinessControl_Outgoing*. Otherwise, a successful check produces an *AckReceiptValid* event that leads to *waitForAckProcessing_Incoming*.

This state works like the state *waitForAckReceipt_Incoming*

described above, except that the business partner interface waits for an acknowledgement of processing. Again, if the initiator's system does not receive the acknowledgement in the agreed time, a time-out exception is issued, the available retries are decreased, and the composite state is re-entered to check the retries. If the acknowledgement is received and the applied checks are successful the initiator enters the state *waitForRespondingBusinessDocument_Incoming*. Otherwise, if the checks fail an *AckProcessingFailure* is actuated and the system switches to *sendFailedBusinessControl_Outgoing*.

In this step of the business transaction, the initiator waits for the responding business document. Not receiving the responding business document within the agreed time (i.e., time to respond) causes the initiator to issue a time-out exception, to decrease the available retries, and to re-enter the composite state to check the retries. Otherwise, receiving the responding business document actuates the call event *receiveResBusinessDoc(ResBusinessDoc)*.

This call event is specified for two outgoing transitions with mutually exclusive guards concerning an intelligible check of the document. If an intelligible check is required the system enters state *checkResBusinessDoc*. In this state the document has to pass through a grammar, sequence and schema validation prior to issuing a proper acknowledgement of receipt. The grammar validation secures that the document's syntax is processable by the system. The sequence validation assures that the document is received in the proper position in terms of the document order. Finally, the schema validation checks the received document against its associated schema(s). If any of the above mentioned checks fails the initiator's system goes immediately to state *sendFailedBusinessControl_Outgoing*. If all checks are passed or if the intelligible checks were not necessary, the system switches to *sendAckReceipt_Outgoing*. Being there, the initiator confirms the proper pick up of the responding business document by sending an acknowledgement of receipt as indicated by the activity (*sendAckReceipt(AckReceipt)*).

Afterwards, in the state *validateContentOfBusinessDocument* the business partner interface validates the content of the responding business document (*validateContent* activity). If validation of the business document's content fails, the system changes its state to *sendFailedBusinessControl_Outgoing*. Otherwise, in the case of success, we reach the state *interactWithBusinessApplication* in order to pass the responding business document to the application. Once this has been completed, we enter the state *sendAckProcessing_Outgoing* and send an acknowledgement of processing to indicate the successful content validation. After executing this action, the initiator's business partner interface has to keep the business transaction alive until no failure messages may be received anymore. This time ends when the time to acknowledge processing has passed after sending the responding business document. Until this time the responder may still signal a time-out exception or a failed business control.

Both failure messages may not only be received after sending the acknowledgement of processing, but anytime the business partner interface is in the composite state of the business transaction. A time-out exception is a signal by the responder that he has not received a message within the expected time. This requires to re-initiate the business transaction. Accordingly, if the call event *receiveTimeOutException(TimeOutException)* occurs, the business partner interface decreases the retry count and re-enters the composite state of the business transaction. A failed business control is received if the responder is not able to process a message correctly. Thus, the call event *receiveFailedBusinessControl(Failure)* terminates the business transaction with a failure.

B. The responders's part

The responder's state machine representing a business transaction must be complementary to the initiator's one. The resulting state machine is depicted in Figure 3. The concepts used in this state machine are very much similiar to the ones of the initiator's one. However, the order of now receiving and then sending a business document is reversed including the handling of acknowledgements. Due to similarity and space limitations, we do not explain all the states in detail.

The major difference is that the retry count is not controlled by the responder. This means after not receiving an acknowledgement of receipt or processing a time-out exception is issued and the composite state is re-entered, but no retry counter is decreased. By re-entering the composite state, the responder's business partner interface is waiting for a requesting business document. The responder does not need to care about the retry count. If no retry is left, no requesting business document will be received. However, the inintiator must issue a failed business control message, which causes the call event *receiveFailedBusinessControl(Failure)* on the responder's side. This exits the composite state and terminates the transaction by leading to the failure end state.

C. Composing multiple Business Transactions

The state machines described before serve as patterns in order to define state-based descriptions for more concrete business transactions (e.g., place order). Defining such concrete state machines according to the presented patterns includes more or less changing state, event, and function names and substituting time variables with concrete values.

A complex business collaboration consists of multiple business transactions. In UMM, a business collaboration protocol is used to specify the flow of business transactions. In order to derive a business partner interface the business collaboration protocol is transformed to a state machine. Each business transaction activity of the business collaboration protocol is transformed to a state of this state machine. It must be decided whether the business partner is the initiator or responder in the corresponding business transaction, and the state is modeled according to the concrete realization of our patterns. The flow between the business transaction activities must be transformed to corresponding state transitions guarded by the business entity states that are reached in the business transactions.

Fig. 3. State machine showing the responders's part of a commercial transaction

IV. CONCLUSION

The current UMM Foundation Module leads to a global choreography of business document exchanges. Realizing the resulting business collaboration requires to implement a derived local choreography supported by a business partner interface on each side of the collaboration. UMM does neither specify how to derive the local choreography, nor does it guide the message handling within the business partner interface. In this paper, we used UMM business transactions and showed how to derive compliant state machines for implementing the message handling within the business partner interfaces of the initiator and of the responder. It becomes obvious that the rather simple patterns of UMM business transactions require a complex message handling mechanism. Business transactions, which are easy to understand for business persons, are transformed to system specifications that are useful to the software engineer.

REFERENCES

[1] *MDA Guide Version 1.0.1*, OMG, June 2003, omg/2003-06-01. [Online]. Available: http://www.omg.org/docs/omg/03-06-01.pdf

[2] *Open-edi Reference Model*, ISO, 2004, ISO/IEC JTC 1/SC30 ISO Standard 14662, Second Edition.

[3] A.-W. Scheer, *Aris - Business Process Modeling*. Berlin: Springer, 2000.

[4] W. van der Aalst, J. Desel, and A. Oberweis, *Business Process Management*. Berlin: Springer, 2000.

[5] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*. New York (NY): HarperBusiness, 2001.

[6] *UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module*, UN/CEFACT, Oct. 2006, Technical Specification Version 1.0, http://www.unece.org/cefact/umm/UMM_Foundation_Module.pdf.

[7] B. Hofreiter, C. Huemer, and J.-H. Kim, "Choreography of ebXML business collaborations," *Information Systems and e-Business Management (ISeB)*, June 2006.

[8] M. Chessell, C. Ferreira, C. Griffin, P. Henderson, D. Vines, and M. Butler, "Extending the concept of transaction compensation," *IBM SYSTEMS JOURNAL,*, vol. 41, no. 4, June 2006.

[9] *RosettaNet Implementation Framework: Core Specification*, RosettaNet, Dec. 2002, v02.00.01. [Online]. Available: http://www.rosettanet.org/rnif

[10] *Core Components Technical Specification - Part 8 of the ebXML Framework*, UN/CEFACT, Nov. 2003, version 2.01.