

# An Implementation to transform Business Collaboration Models to executable Process Specifications

Michael Ilger  
ilger@ani.univie.ac.at  
University of Vienna

Marco Zapletal  
marco@cs.univie.ac.at  
University of Vienna

**Abstract:** UN/CEFACT's Modeling Methodology (UMM) is a well accepted and formal notation to analyze and design B2B business processes. In a service oriented architecture (SOA) environment process specification languages like the Business Process Specification Schema (BPSS) are used to configure B2B information systems. However, mappings from UMM models to BPSS process specifications currently exist just on a conceptual level. This results in a gap between defined B2B processes and BPSS configurable e-commerce systems. Thus, a model driven code generation of BPSS descriptions is required. In this paper we present a technical implementation of a transformation engine that generates BPSS process specifications from a UMM model represented in the XML Metadata Interchange (XMI) language. This implementation bridges the gap mentioned above. It has been used in the EU project *GILDAnet* to generate BPSS descriptions from logistic processes modeled in UMM.

## 1 Motivation

Business process modeling has traditionally focused on describing intra-organizational processes. In a business-to-business (B2B) environment two or more organizations take part in an inter-organizational process. Consequently an agreement of all participants on a shared business process, also called collaboration, is required. However, each of the participating partners describes the shared process from his point of view. Therefore the described sights will not match. Thus, in order to specify shared processes it is inevitable to use a method that describes the process from a common point of view. *UN/CEFACT's Modeling Methodology (UMM)* [UN/01] is such a well accepted method in the B2B sector.

Furthermore, it has to be a modeler's goal to use the designed processes for real business instead of leaving them in some unread manuals or strategic papers. It is the intention of UMM to describe processes not just for human understanding but also to create machine-interpretable artifacts. In order to configure e-commerce information systems dynamically in changing environments (e.g. partners, processes, etc.), system-executable process specifications are needed. The *Business Process Specification Schema (BPSS)* [UN/03], known as a part of the *ebXML* framework, is such an XML-based process definition language. However, the mapping of relevant segments of a UMM model to a BPSS instance is only conceptually denoted so far [HH04a] [HHK05]. Thus a tool implementation is required, which generates BPSS instances out of UMM models. In this paper we present

the implementation of a tool which transforms UMM models into BPSS files. In the EU funded *GILDAnet* project this transformation engine was developed to generate system-executable BPSS descriptions from modeled supply chain processes. The remainder of this paper is structured in six sections. In section 2 we present related work before giving an overview about the transformation process in section 3. Section 4 to 6 describe each of the three transformation stages. Section 7 finishes the paper with a conclusion.

## 2 The transformation process: An overview

The transformation engine we implemented transforms UMM models to BPSS process specifications. The overall transformation process covered by our implementation spans over three major stages (denoted by the gray arrows in figure 1). The engine needs a valid UMM model as input to the first stage and outputs a BPSS compliant process specification after completing the third stage.

Considering the input format, it is a definitive goal of this implementation to stay independent of specific UML modeling tools. Thus, an input format is required which is supported by a wide range of different modeling environments. For us the widespread *XML Metadata Interchange (XMI)* [Obj00] standard was the candidate of choice as input format specification. In the first stage of the workflow, the object structure that corresponds to the XMI tree is dissolved. Then equal UML element types are grouped in list data structures in order to ease and speed up further processing. These collections are input to the second stage, where we map these elements to an object structure that conforms to UMM meta model. Furthermore, some basic consistency checks are applied during the mapping to ensure a valid UMM instance. Within the third stage the valid UMM object representation is mapped to a BPSS instance according to the approach described in [HH04a].

Splitting the workflow into the three stages described above enhances the modularization of the transformation engine. The object representation of the UMM meta model acts as the core of the engine. Additional modules that use this core, but implement different input or output formats may easily be implemented. Candidates for other output formats of a UMM transformation might be the *Business Process Execution Language (BPEL)* [HH04b] or the *Web Services Choreography Description Language (WS-CDL)*.

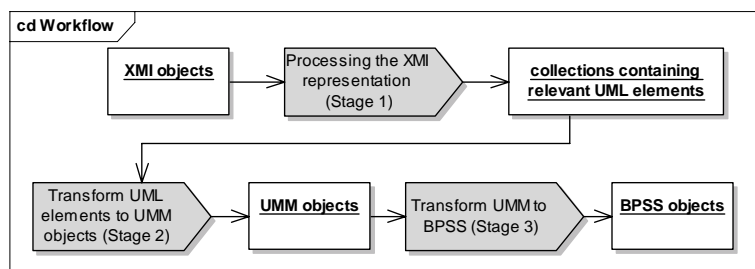


Figure 1: The workflow of the transformation engine consists of three stages (gray arrows)

## The *GILDAnet* UMM example

In this paper we demonstrate the workflow of our implementation on the basis of a simple example. It depicts a real-world process modeled in the EU project *GILDAnet* (*Global Integrated transport Logistic Data NETwork*) using UMM. In the *GILDAnet* project this implementation has been used to generate BPSS process specifications from the described processes.

In UMM a collaborative process is described by a *business collaboration protocol*. Two or more partners participate in the execution of a *business collaboration protocol*. A collaboration is composed of one or more interactions between its participants. Each interaction is performed by exactly two partners and results in an information exchange. UMM denotes an interaction by the concept of a *business transaction activity* that is refined by a *business transaction*. A *business transaction* starts with an activity performed by the initiating party. The execution of this activity outputs a message that is input to an activity of the reacting party. Depending on the business case that is represented by a certain *business transaction* the reacting party sends a response back to the initiating party (compare a request for quote to a shipping notification). The execution of a *business transaction* may succeed or fail, which has impact on the flow in the *business collaboration protocol*.

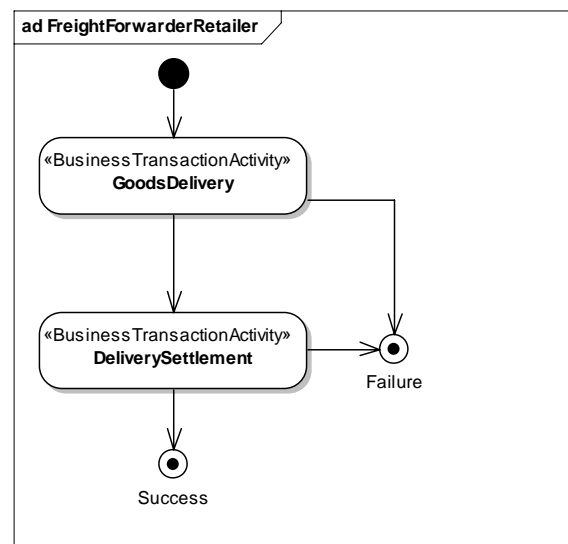


Figure 2: Example *business collaboration protocol* describing the collaboration between freight forwarder and retailer.

Our example consists of one *business collaboration protocol* that describes a supply chain process between a freight forwarder and a retailer dealing with the delivery of perishable goods (figure 2). The *business collaboration protocol* is composed of two *business transaction activities* called *GoodsDelivery* and *DeliverySettlement*. Each *business transaction activity* is refined by an equally named *business transaction*. However, due to space lim-

itation we just describe the goods delivery interaction (figure 3) in detail in this paper. Performing this interaction the freight forwarder has the role of an initiator by sending a CMR <sup>1</sup> concerning the delivery of goods to the retailer. The retailer, playing the role of the reactor, sends then a signed CMR <sup>1</sup> response back to the freight forwarder. In order to ease understanding of our engine, we explain each step of the transformation workflow using the perishable goods example.

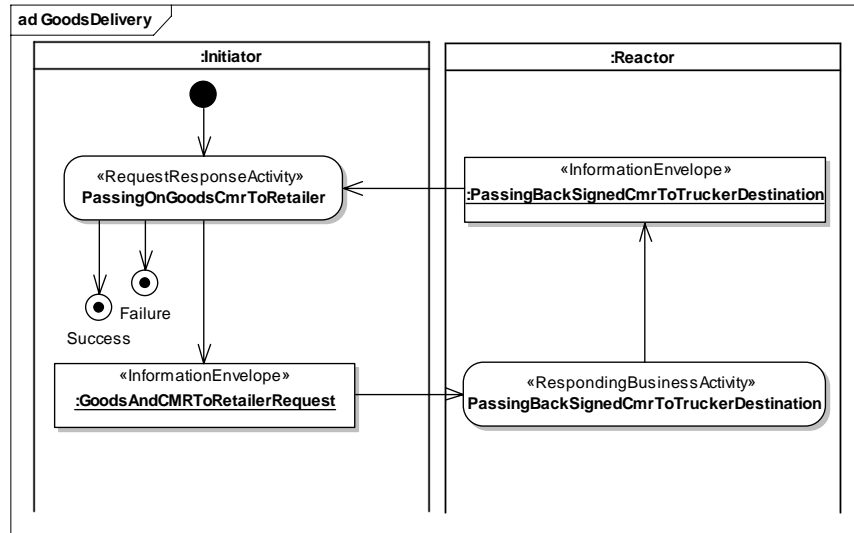


Figure 3: Example *business transaction* describing an information exchange concerning a CMR<sup>1</sup>

### 3 Processing the XMI representation (Stage 1)

As described above the implementation accepts XMI compliant files as input to the transformation process. The XMI source needs to describe a valid UMM model in order to successfully generate a BPSS process description. Due to the huge size of an XMI structure even regarding our small example, we refrain from including the example XMI code here. In order to avoid a manual traversal of the extensive XMI structure we utilized an XML data binding framework. The *Apache XMLBeans*<sup>2</sup> framework was our tool of choice to get an object representation from the XMI file.

<sup>1</sup>CMR is an abbreviation for *Carriage of Goods by Road*

### 3.1 Extracting the relevant XMI objects

The XMI data and hence its object representation is nested in a tree structure analogous to the UML model structure. However, from the large amount of data in the XMI structure we just need one element type as entry point in the further transformation - the activity graphs. All other element types needed for a BPSS generation (e.g. classes, use cases) can be reached by starting from the particular activity graph. Therefore recursive processing is used to traverse the whole tree and to store the activity graphs in a flat collection. However, the *Rational Rose* XMI flavor defines stereotypes and tagged values outside of their elements and links them by a common ID. Hence, the recursive algorithm collects also these elements and puts them into homogeneous dictionary data structures. Using the ID as the key and the element itself as the value enables a fast lookup of required objects in further steps.

In our example the *business collaboration protocol* (figure 2) and the two *business transactions* (figure 3 shows the goods delivery interaction) are stored in the activity graph collection. Stereotypes and tagged values of each model element are stored in the homogeneous dictionaries. The activity graph collection is then input to the second stage.

### 3.2 Obstacles regarding the XMI processing

Choosing XMI as the input format for the implementation raised some serious obstacles. Unfortunately XMI is an ambiguous interchange format. Hence nearly each tool vendor supports a different XMI flavor. Regarding our implementation we concentrated on the XMI flavor produced by the *Unisys XMI Add-In 1.3.6 for Rational Rose 2003*. The *Apache XMLBeans*<sup>2</sup> framework needs a *W3C XML Schema instance (XSD)* as a base to generate an object representation. However, the XMI flavor used by the *Unisys XMI Add-In* is only described by a *Document Type Definition (DTD)*. Consequently, we had to convert the document type definition to a schema representation including some manual adoptions. Furthermore the *Unisys XMI Add-In* uses only a small subset of the XMI standard. Hence, a lot of generated classes are never used. Some problems have also been caused by the *Unisys XMI Add-In* itself. We discovered some serious flaws in this implementation especially regarding the export of UML activity graphs. Transitions that are connected to object flow states are also always exported as leading away from them. Furthermore, object flow states are not exported as being the contents of a partition in the XMI representation. These two problems result in the impossibility to distinguish whether a message is sent by the requesting or responding party. The last problem we faced was that the classifier of a partition is missing in the export. In UMM the classifier of a partition in a *business transaction* corresponds to the role that performs the activity located in the partition. In order to circumvent this handicap the name of the partition has been used to denote the performing role.

## 4 Transforming UML elements to UMM objects (Stage 2)

In the second stage of the transformation we instantiate an object structure corresponding to the UMM meta model using the input from the preceding stage. Thus, we implemented the UMM meta model as a set of *Java Beans* in order to simplify the mapping procedure. At first a UMM model object is instantiated that contains all further UMM artifacts. The UMM model contains in turn a *business transaction view* and a *business requirements view* object. The *business transaction view* is a container for most of the artifacts needed to generate a BPSS. The *business requirements view* object contains just *business collaboration protocol use cases* and *business transaction use cases* that capture the requirements on a corresponding collaboration or transaction.

Regarding the technical implementation of this UML (represented as XMI object structure) to UMM object transformation we focused on keeping the mapping code as simple and clean as possible. Thus, we used reflection mechanisms to dynamically instantiate UMM object types based on the stereotypes occurring in a model instance. Furthermore, tagged values are dynamically looked up and set in the particular object structure based on the values occurring in the corresponding XMI objects.

After the initial UMM model structure is created, we start the actual mapping. The starting point for the mapping procedure is the collection containing the UML activity graphs. *Business collaboration protocols* and *business transactions* are both stereotyped activity graphs and all other relevant artifacts can be traversed by processing them. Thus, a loop iterates over all UML activity graphs and determines whether the current graph is a *business collaboration protocol* or a *business transaction*. However, relying only on the stereotype of the activity graph may cause inconsistent states in further steps if unanticipated elements are found. Hence, based on the stereotypes of the activities contained in the graph the algorithm decides the stereotype of the activity graph. The activity graph is a *business collaboration protocol* if only *business transaction activities* are found or a *business transaction* if a *requesting business activity* and a *responding business activity* are found. If a mix is found, the algorithm quits processing this activity graph by throwing an appropriate exception.

### 4.1 Processing collaborations between partners

Processing a *business collaboration protocol* starts with creating the appropriate object representation. Thus, a *business collaboration protocol* object is instantiated for the *FreightForwarderRetailer* collaboration (figure 2) and added to the *business transaction view*. As next step, the algorithm creates the containing *business transaction activities*. In our example two *business transaction activity* objects called *GoodsDelivery* and *DeliverySettlement* are created. The *business collaboration protocol* object is again a container for the elements contained in the corresponding activity graph. Hence, the two *business transaction activities* are added to the *business collaboration protocol FreightForwarderRetailer*. Then the algorithm processes pseudo states and final states. Regarding the *FreightForwarder-*

*Retailer* collaboration two final state objects are created - one *Success* and one *Failure*. At this point we had to implement a workaround in our UMM meta model representation. Due to the fact that stereotyped elements inherit from their UML base classes, the UMM stereotypes inherit also the relations of their base classes in the UML meta model. This means for example that there are relations defined between activity graphs and final states in the UML meta model, but not between *business collaboration protocols* and final states in the UMM meta model. Thus, we either had to implement the relevant parts of the UML meta model in our object representation (and let the UMM objects inherit from them) or to add some workaround extensions directly to our UMM meta model data structure. In order to keep the object model relatively simple we decided to implement the workaround extensions. Hence, also the final state objects *Success* and *Failure* are added to the *business collaboration protocol* instance by the algorithm. The next step in processing *business collaboration protocols* identifies the transitions including their guard conditions between the model elements of the activity graph. In our example, five transitions as shown in the *business collaboration protocol* are identified. Similar to the pseudo and final states problem, we had to extend the *business collaboration protocol* object again to pass the identified transitions. Finally, the *business collaboration protocol use case* that captures the requirements on a *business collaboration protocol* is identified. In the *GILDAnet* example, the *business collaboration protocol* is modeled as the behavior of the corresponding use case. Thus, the *business collaboration protocol use case* is found by the behavior-context relation in the UML meta model [Obj04]. Again, an object is created that complies with the *business collaboration protocol use case* and added to the *business requirements view* container.

## 4.2 Processing information exchanges between partners

If an activity graph is identified as a *business transaction*, first of all the corresponding object is created and added to the *business transaction view* container object. Considering our example, a *business transaction* object named *GoodsDelivery* is instantiated. The algorithm subsequently iterates over the activities contained in the *business transaction*. A *business transaction* must have exactly two activities, one *requesting business activity* and one *responding business activity*. Unless exactly two activities are found, the algorithm exits with an exception. Otherwise, the adequate object is created and added to the *business transaction* container. In our example a *requesting business activity* *PassingOnGoodsCmrToRetailer* and a *responding business activity* *PassingBackSignedCmrToTruckerDestination* are constructed. Then the algorithm iterates over the object flow states of the *business transaction*. The object flow states represent an instance of a certain *information envelope* that is transmitted. Thus, we need to determine the appropriate classifier of the envelope that is being transmitted. In UML, and hence in the XMI object representation, the relation between a class and an object flow state thereof is denoted via a so called classifier in state [Obj04]. Following this relationship two *information envelopes* are created and added: One representing the *GoodsAndCMRToRetailerRequest* and one for the *Passing-*

*BackSignedCMRToTruckerDestinationResponse*. They are in turn added to the *business transaction*. Finally, the roles participating in the *business transaction* are determined via the classifier of the particular partition. In order to keep the example simple the *requesting business activity* is performed by a role called initiator and analogous the *responding business activity* is performed by a role called reactor. For each role a corresponding object representation is created, added to the *business transaction view* and linked with the activity it performs.

In UMM, a *business transaction* follows one of six pre-defined *business transaction* patterns. The pattern is specified by the stereotype of the requesting activity in a transaction. Two one-way (*notification, information distribution*) and four two-way (*request/response, query/response, commercial transaction, request/confirm*) information exchange patterns are adopted by UN/CEFACT from RosettaNet [Ros02]. According to Open-edi [ISO95], these patterns cover all known legally binding interactions between two e-commerce applications. A further analysis of the transaction patterns is found in [BJJW02]. Except the optional response, the *business transaction* patterns adhere all to the same structure. Neither additional model elements nor another activity flow are allowed. Only additional final states capturing a different semantic are allowed, but not considered in BPSS. Thus, mapping initial and final states as well as transitions of a *business transaction* is not required.

As described in section 3.2 the classifier of a partition is not exported by the *Unisys XMI Add-In*. Hence, as a workaround the classifier of a partition is denoted using the name of the partition. Due to space limitations we left this detail out in our example. Furthermore, there is no possibility to determine which information envelope is sent by which party (see section 3.2). For our implementation we had to assume that the first object flow state exported by the *Unisys XMI Add-In* corresponds to the *information envelope* sent by the requesting party.

### 4.3 Finishing the UMM object model

In a last step *business transaction activities* are linked to their refining *business transactions*. Due to some clashes between the UMM meta model and the modeling environment provided by *Rational Rose 2003*, the refining *business transaction* of a *business transaction activity* can only be identified using pre-defined naming rules. However, a more detailed description of this problem would exceed the scope of this paper and is therefore left out. Anyway, an iteration over all *business transaction activities* is needed to identify their refining *business transaction*. We tried to optimize this step using a dictionary data structure with the name of the *business transaction* as the key and the actual object as value.

The resulting object representation of the UMM model is then input to the last step that generates a BPSS compliant process specification. Figure 4 shows a conceptual overview of the constructed objects and their relationships regarding our example.

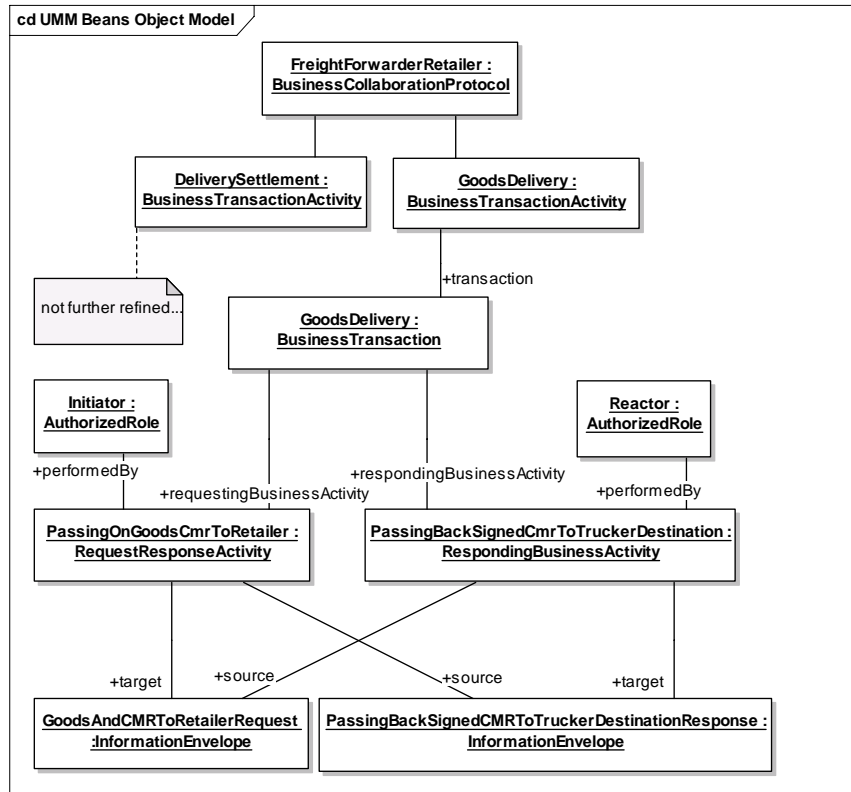


Figure 4: Conceptual overview of the UMM objects created in the example

## 5 Transforming UMM to BPSS (Stage 3)

In the final stage of the transformation workflow we take the UMM object model and map it to a BPSS instance. Since BPSS is designed to be a subset of the UMM specification, the mapping for many elements is intuitive. In the following sub chapters we will describe in detail how our implementation generates BPSS. Thus we use the XML-based BPSS notation in the subsequent illustration of our perishable goods example, simplified to capture just the relevant aspects. Due to the fact that XML is chosen as the notation of BPSS, the documents are still human-readable and relatively easy to understand. The fact that XML is used also creates some additional transformation requirements. For example XML requires a special format for durations as used in *timeToPerform*. In our approach we use *business collaboration protocols* contained in the UMM object model as entry point to start the BPSS generation. Thus, we can trivially create one BPSS instance for each *business collaboration protocol*. This approach is important to us, as we have the goal of creating files which can be uploaded to a registry server. As our registry server can only denote if a company supports a role in a special BPSS this functionality works only in

an environment where one file is available for each *binary collaboration*. Otherwise it is ambiguous to decide which process is supported. The algorithm iterates over these objects and instantiates a *process specification* object in each cycle. The *process specification* is a data binding object corresponding to the root element in a BPSS instance. When the transformation of the complete model is finished a file is created for each BPSS and data is serialized to the file system.

In the description of the BPSS structure we will omit a couple of details which are included in our tool, but not required to understand the function of our mapping. Most importantly each element features a so-called *nameID*, which is a unique id. While it is a real unique ID in generated files we use simplified ones in our example. Every element features also a *documentation* element which may contain further information concerning the respective element. Finally we also focused only on *binary collaborations*, because multi party collaborations do not appear in the *GILDAnet* environment and are also deprecated in the current BPSS version.

## 5.1 Processing collaborations between partners

Due to the one to one relationship between the *business collaboration protocol* and the BPSS *process specification*, the *process specification* inherits the name from the collaboration. Next a *binary collaboration* object is constructed representing the *business collaboration protocol*. Regarding our example the *process specification* as well as the *binary collaboration* object are named *FreightForwarderRetailer*. The *binary collaboration* specifies a set of attributes. The *pattern* attribute has currently no counterpart in UMM and is therefore not specified. It is reserved for future considerations when patterns for standardized collaborations will be available by *UN/CEFACT*. The *isInnerCollaboration* attribute is set true if the particular collaboration is a nested one or otherwise set false. UMM revision 12 [UN/01] used in our implementation does not provide this mechanism, thus this attribute is also left out. The remaining attributes (*beginsWhen*, *endsWhen*, *preCondition* and *postCondition*) are derived from the *business collaboration protocol use case* that captures the requirements on the collaboration.

As next the two participating *roles* of a binary collaboration are processed. The role names are set using the names from the corresponding *authorized roles* taking part in the *business transaction*. Hence, it is a requirement that the same two roles take part in each transaction of one binary collaboration. In our example we extract the two roles *initiator* and *reactor* out of the *business transactions* and create an object representation of them.

Now the algorithm iterates over the *business transaction activities* contained in the *business collaboration protocol* object of the UMM model. For each *business transaction activity* in the UMM *business collaboration protocol* a corresponding *business transaction activity* in the BPSS is instantiated. In our example we create a business transaction activity in the BPSS called *GoodsDelivery* and a second one called *DeliverySettlement*.

Furthermore, we need to specify the initiating role of the *binary collaboration*. Thus, we check for each *business transaction activity* if it is the first one in the collaboration flow. This is determined through the relation to the initial state. In some cases there is no rela-

tionship between the initial state and the first *business transaction activity*, but there are some pseudo states in between. Anyway, a depth first search algorithm is used to determine the *business transaction activity* that is most closely located to the initial state. Once the first *business transaction activity* is located the initiating role of the *binary collaboration* (*initiatingRoleIDREF*) is set using the initiating role of the refining *business transaction*. In the simple example the *initiator* of the *GoodsDelivery business transaction activity* is also the initiator of the collaboration.

Moreover the attributes *fromRole* and *toRole* of a *business transaction activity* specify, which role of the *binary collaboration* plays a certain role in the refining *business transaction*. This relationship can only be constructed by comparing the names of the roles. Similar as with *binary collaborations* the attributes *beginsWhen*, *endsWhen*, *preCondition* and *postCondition* denote some requirements on a *business transaction activity*. These are derived from the associated *business transaction use case* of the *business transaction* (that refines this *business transaction activity*). Regarding our example, the mapping is again trivial (e.g. initiator of the collaboration maps to the initiator of the transaction). Furthermore, two more attributes (*isConcurrent*, *timeToPerform*) are taken from the UMM *business transaction activity* to the BPSS equivalent. At this point our algorithm continues with processing the *business transaction* that is described in section 5.2. In order to facilitate understanding of the generation workflow we continue with the remaining elements of the *binary collaboration* element.

Finally all pseudo states, final states and transitions have to be added to the BPSS. Pseudo states - initial states, decisions, forks and joins - are simply mapped to their corresponding representation in the BPSS. Considering our example there is no pseudo state in the collaboration and hence no such element in the BPSS description. Regarding final states, BPSS distinguishes between successful and failed collaboration executions denoted by *Success* and *Failure* elements. BPSS demands that there exists at least one *Success* and one *Failure* per each *binary collaboration*. Although not mandated by UMM, we postulate this as a requirement to a UMM model that is input to our implementation. In our example *business collaboration protocol* the two existing final states are simply mapped to their BPSS equivalent. In BPSS final states can only be referenced from one single business state. In our example this means that two *Failures* and one *Success* object are constructed. Finally we process the transition collection contained in the *business collaboration protocol* object representation. In a BPSS *transition* elements are just needed if a transition leads from one business state to another. Otherwise, if a transition is connected to an initial, pseudo or final state the transition is denoted using the *fromBusinessStateIDREF* or respective *toBusinessStateIDREF* attribute. Hence, in our example we have one single *transition* element which connects our two *business transaction activity* elements. The remaining transitions which connect the *business transaction activities* to all other states (as in figure 2) are either specified in the *Start* element or in the *Success/Failure* elements. The following listing shows the relevant parts of the *binary collaboration* derived from our example.

```
<bpss:BinaryCollaboration initiatingRoleIDREF="r1"
  nameID="bc1" name="FreightForwarderRetailer">
  <bpss:Role nameID="r1" name="Initiator"/>
  <bpss:Role nameID="r2" name="Reactor"/>
```

```

<bpss:Start toBusinessStateIDREF="bs1"
  toBusinessState="GoodsDelivery" nameID="s1"/>
<bpss:BusinessTransactionActivity businessTransaction="GoodsDelivery"
  businessTransactionIDREF="bt1" toRoleIDREF="r2"
  toRole="Reactor" fromRoleIDREF="r1" fromRole="Initiator"
  name="GoodsDelivery" nameID="bta1"/>
<bpss:BusinessTransactionActivity
  businessTransaction="DeliverySettlement"
  businessTransactionIDREF="bt2" toRoleIDREF="r2"
  toRole="Reactor" fromRoleIDREF="r1" fromRole="Initiator"
  name="DeliverySettlement" nameID="bta2"/>
<bpss:Success conditionGuard="Success" fromBusinessStateIDREF="bt1"
  fromBusinessState="GoodsDelivery" nameID="s1" name="Success"/>
<bpss:Failure conditionGuard="Failure" fromBusinessStateIDREF="bt1"
  fromBusinessState="GoodsDelivery" nameID="s2" name="Failure"/>
<bpss:Failure conditionGuard="Failure" fromBusinessStateIDREF="bt2"
  fromBusinessState="DeliverySettlement" nameID="s3" name="Failure"/>
<bpss:Transition toBusinessStateIDREF="bta2"
  toBusinessState="DeliverySettlement" fromBusinessStateIDREF="bta1"
  fromBusinessState="GoodsDelivery" nameID="t1"/>
</bpss:BinaryCollaboration>

```

## 5.2 Processing information exchanges between partners

A *business transaction* in BPSS is equivalent to a *business transaction* in UMM. Thus, mapping this construct is straightforward. Again, our algorithm creates a *business transaction* object and sets its name according to the UMM transaction. Furthermore, the *pattern* that is used in the particular transaction has to be set. We derive it from the concrete stereotype of the *requesting business activity* in the corresponding UMM transaction. Regarding our example we create a *business transaction* container in the BPSS and assign it the name *GoodsDelivery*. Moreover, we set the *pattern* attribute to *RequestResponseActivity*.

In the next step, we construct the object representations of the *requesting business activity* and the *responding business activity* - in our example the activities *PassingOnGoodsCmrToRetailer* and *PassingOnGoodsCmrToRetailer* - and add them to the BPSS *business transaction* object. Both *business actions* contain a set of tagged values defining parameters regarding security and legal facets. The semantical mapping of these parameters follows the extensive descriptions in [HH04a], whereas the technical implementation is quite simple and hence left out. The last stage in constructing the BPSS representation of a *business transaction* is to denote the information that is exchanged in the particular interaction. An *information envelope* of a UMM *business transaction* corresponds to a *document envelope* in BPSS. Similar to UMM, the envelopes that are described in a BPSS *business transaction* are instances of a certain *business document*. It follows, that in a BPSS structure *business documents* are denoted outside of a certain *business transaction* in order to reuse them. In our example this results in two *business document* objects located directly in the *binary collaboration* container - named *GoodsAndCMRToRetailerRequest* and *Passing-BackSignedCMRToTruckerDestinationResponse*. Then we reference the *GoodsAndCM-*

*RToRetailerRequest* document by adding a *document envelope* object to the *requesting business activity* object and linking them via the *businessDocumentIDREF* attribute of the *document envelope*. The same procedure applies to the *business document* sent back by the requesting party. A *document envelope* is added to the *responding business activity* and linked to the *PassingBackSignedCMRToTruckerDestinationResponse* document. The actual document structure is then referenced by using the *specificationLocation* attribute of the *business document* object. The following listing denotes the second part of our BPSS describing the perishable goods example.

```

<bpss:BusinessTransaction pattern="RequestResponseActivity"
  nameID="bt1" name="GoodsDelivery">
  <bpss:RequestingBusinessActivity nameID="reqBA"
    name="PassingOnGoodsCmrToRetailer">
    <bpss:DocumentEnvelope businessDocumentIDREF="bd1"
      businessDocument="GoodsAndCMRToRetailerRequest" nameID="de1" />
  </bpss:RequestingBusinessActivity>
  <bpss:RespondingBusinessActivity nameID="resBA"
    name="PassingOnGoodsCmrToRetailer">
    <bpss:DocumentEnvelope businessDocumentIDREF="bd2"
      businessDocument="PassingBackSignedCMRToTruckerDestinationResponse"
      nameID="de2" />
  </bpss:RespondingBusinessActivity>
</bpss:BusinessTransaction>

<bpss:BusinessDocument specificationLocation="..."
  nameID="bd1" name="GoodsAndCMRToRetailerRequest"/>
<bpss:BusinessDocument specificationLocation="..."
  nameID="bd2" name="PassingBackSignedCMRToTruckerDestinationResponse"/>

```

## 6 Related work

In the B2B environment there are many different standards, proposals and tools available. The *ebXML* framework is considered to be among the more important ones. The reason might be that it offers a full suite of specifications ranging from messaging to registry servers which allow companies to advertise their services to anybody who is willing to conduct business. Relevant for our implementation is only *ebXML* BPSS, which allows the description of collaborative processes.

Currently no implementation of a mapping between UMM and BPSS is available. This can be considered a result of the limited usage of BPSS in today's e-commerce. Some theoretical approaches on how a mapping could work are already available. In [HHK05] we can find a description of the relation between these two concepts.

Though BPSS is not the only language to describe the choreography of business processes. There exist other well known examples like BPEL, the *Business Process Execution Language* [BEA03], which is supported by major companies or the *Web Services Choreography Description Language (WS-CDL)* [Wor04] developed by the W3C. BPSS has one big advantage though. It is closely connected to UMM, which has become an important factor in modern modeling.

In our JAVA-based tool we decided to use XMI, the XML Metadata Interchange language [Obj00], as the source format. Due to the fact that XMI is used by UML tools as an exchange format this ensures compatibility to as many different tools as possible. Unfortunately, as discussed in section 3.2, this decision caused some unpredictable problems. In order to facilitate the handling of XML structures we incorporated *Apache XMLBeans*<sup>2</sup> as an XML data binding framework. Alternatives would have been *Java Architecture for XML Binding (JAXB)* or *Castor*. The *XMLBeans* framework appeared like the most sophisticated solution at that point.

Due to the fact that source and target format are both XML-based using W3C's *XSL Transformation (XSLT)* [Wor99] technology would have been an obvious solution. However the transformation algorithm requires a lot of logical operations that are difficult to implement using XSLT or even exceeding its functionality. This includes for example the creation of unique IDs, consistency checks or splitting up a UMM model in different BPSS files (see section 5 for further explanations).

## 7 Conclusion

As simple as the transformation from UMM to BPSS seems, it still has some hidden pitfalls, which are not easy to see in advance. One of the biggest problems is right at the beginning of the transformation process. When XMI is chosen as data exchange format because it is based on a common standard you would expect it to look more or less the same for any kind of modeling tool. Unfortunately this is not the case and XMI output for the same model might vary dramatically for different tools. This leads to the need of specialized input converters for each supported dialect. In the special case of *Rational Rose* data output another problem was that some information which was present in the diagrams was not put into the XMI code and therefore had to be reconstructed or forced to be replaced through additional information required by special modeling guidelines.

Generating BPSS code turned out to be the lesser problem. Most information is already provided in a satisfying way by the UMM model. Some problems arise due to the fact that there is some information which is not included in UMM but required by BPSS. Furthermore, some data structures which are supported by UMM cannot be translated into correct BPSS files.

One insight gained during the project was that the approach of a central data structure provides a good and flexibly base for development as it allows a modular design to support different input and output types. This leads to a point where support for different choreography languages and input languages, such as different XMI flavors, can easily be added.

---

<sup>2</sup>Apache XMLBeans framework - <http://xmlbeans.apache.org/>

## References

- [BEA03] BEA, IBM, Microsoft, SAP AG and Siebel Systems. *Business Process Execution Language for Web Services*, May 2003. Version 1.1.
- [BJJW02] Maria Bergholtz, Prasad Jayaweera, Paul Johannesson, and Petia Wohed. Process Models and Business Models - A Unified Framework. In Stefano Spaccapietra, Salvatore T. March, and Yahiko Kambayashi, editors, *ER (Workshops)*, volume 2503 of *Lecture Notes in Computer Science*, pages 364–377. Springer, 2002.
- [HH04a] Birgit Hofreiter and Christian Huemer. ebXML Business Processes - Defined both in UMM and BPSS. In M. Nüttgens and J. Mendling, editors, *XMLABPM 2004, Proceedings of the 1st GI Workshop XMLABPM – XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany, March 2004*, pages 81–102, March 2004.
- [HH04b] Birgit Hofreiter and Christian Huemer. Transforming UMM Business Collaboration Models to BPEL. In Robert Meersman, Zahir Tari, and Angelo Corsaro, editors, *OTM Workshops*, volume 3292 of *Lecture Notes in Computer Science*, pages 507–519. Springer, 2004.
- [HHK05] Birgit Hofreiter, Christian Huemer, and Ja-Hee Kim. Choreography of ebXML business collaborations. *Information Systems and e-Business Management (ISeB)*, 2005.
- [ISO95] ISO. *Open-edi Reference Model*, 1995. ISO/IEC JTC 1/SC30 ISO Standard 14662.
- [Obj00] Object Management Group, Inc. *OMG XML Metadata Interchange (XMI) Specification*, November 2000. Version 1.1 (formal/00-11-02).
- [Obj04] Object Management Group, Inc. *Unified Modeling Language (UML) Specification*, July 2004. Version 1.1 (formal/04-07-02).
- [Ros02] RosettaNet. *RosettaNet Implementation Framework: Core Specification*, December 2002. V02.00.01.
- [UN/01] UN/CEFACT Techniques and Methodologies Group. *UN/CEFACT's Modeling Methodology - Meta Model*, November 2001. Revision 12.
- [UN/03] UN/CEFACT Techniques and Methodologies Group. *UN/CEFACT ebXML Business Process Specification Schema*, October 2003. Version 1.10.
- [Wor99] World Wide Web Consortium (W3C). *XSL Transformations (XSLT)*, November 1999. Version 1.0.
- [Wor04] World Wide Web Consortium (W3C). *Web Services Choreography Description Language*, December 2004. Working Draft Version 1.0.