A Business Collaboration Registry Model on Top of ebRIM

Birgit Hofreiter, Christian Huemer
Faculty of Computer Science
University of Vienna
Liebigg. 4, 1010 Vienna, Austria
{birgit.hofreiter, christian huemer}@univie.ac.at

Marco Zapletal
Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9-11/188, 1040 Vienna, Austria
marco@ec.tuwien.ac.at

Abstract

UN/CEFACT's Modeling Methodology (UMM) is a well accepted approach to define inter-organizational business processes. UMM models should be managed in a registry for two reasons: Firstly, business partners supporting the process can find it and bind to it. Secondly, a model - or more important parts thereof - may be reused in another model of an inter-organizational process. Accordingly, registering one model as one object in a registry is not appropriate. Those parts of a model that may be reused must become registry objects themselves. Extracting parts of a model results in objects that are logically inter-related. Thus, a registry model taking care of these inter-relationships is needed. In this paper we present a so-called business collaboration registry model that sits on top of the ebRIM (ebXML registry information model) in order to manage UMM business collaboration models in an ebXML registry. Furthermore, we outline the registry management functions for maintaining models in the registry.

1. Introduction

In B2B e-Commerce business partners need to collaborate. The execution of a collaboration depends on commitments established between the participating partners. UN/CEFACT's Modeling Methodology (UMM) [17] is used to model the choreography and data exchange commitments to be agreed between partners. Thus, a UMM business collaboration model becomes a kind of 'contract' that guides a business partnership. Similiarly to a contract a UMM model is defined from a global perspective. A global choreography has the potential to achieve an agreement between the partners. Local choreographies derived from the global UMM model enable the configuration of each partner's system. In order to make UMM models available to potential business partners they should be made public in a registry. This allows business partners to search for and to bind to these processes.

Another reason for maintaining a UMM model in a registry is fostering reuse. It is not always an entire model that is reused. It is even more likely that only a certain part of a model is reused within another model. Thus, it is key to allow registration of these well-defined parts as separate registry objects and to extract the relevant parts if an entire model is submitted. Inasmuch, there exist dependencies between certain registry objects. These dependencies describe that one registry object is executed as part of another registry object, or that one registry object describes the requirements of a choreography, which flow is specified in another registry object. Thus, a registry model supporting the dependencies between certain types of registry objects is needed. The goal of the paper is to define a *business collaboration registry model* on top of the ebXML registry information model (ebRIM) [9] that manages UMM models and parts thereof. It starts from the relevant UMM stereotypes identifying types of registry objects and their inter-dependencies. Furthermore, we identify features that must be realized in a registry implementation ensuring consitent management of UMM models. We are currently implementing these features in a prototype registry implementation connecting our UMM modeling tool [5].

The remainder of this paper is structured as follows: In section 2 we give a brief introduction to UMM [17] - of which we are co-editors - and its artifacts. This helps with identifying those artifacts that are candidate for reuse. In section 3 we elaborate on the relationships between these artifacts which must be handled in a *business collaboration registry model*. In section 4 we outline how UMM models and parts thereof are represented in our *business collaboration registry model*. Section 5 outlines features for managing UMM artifacts consistently within an ebXML registry. Section 6 focuses on related work and a short summary in section 7 concludes the paper.

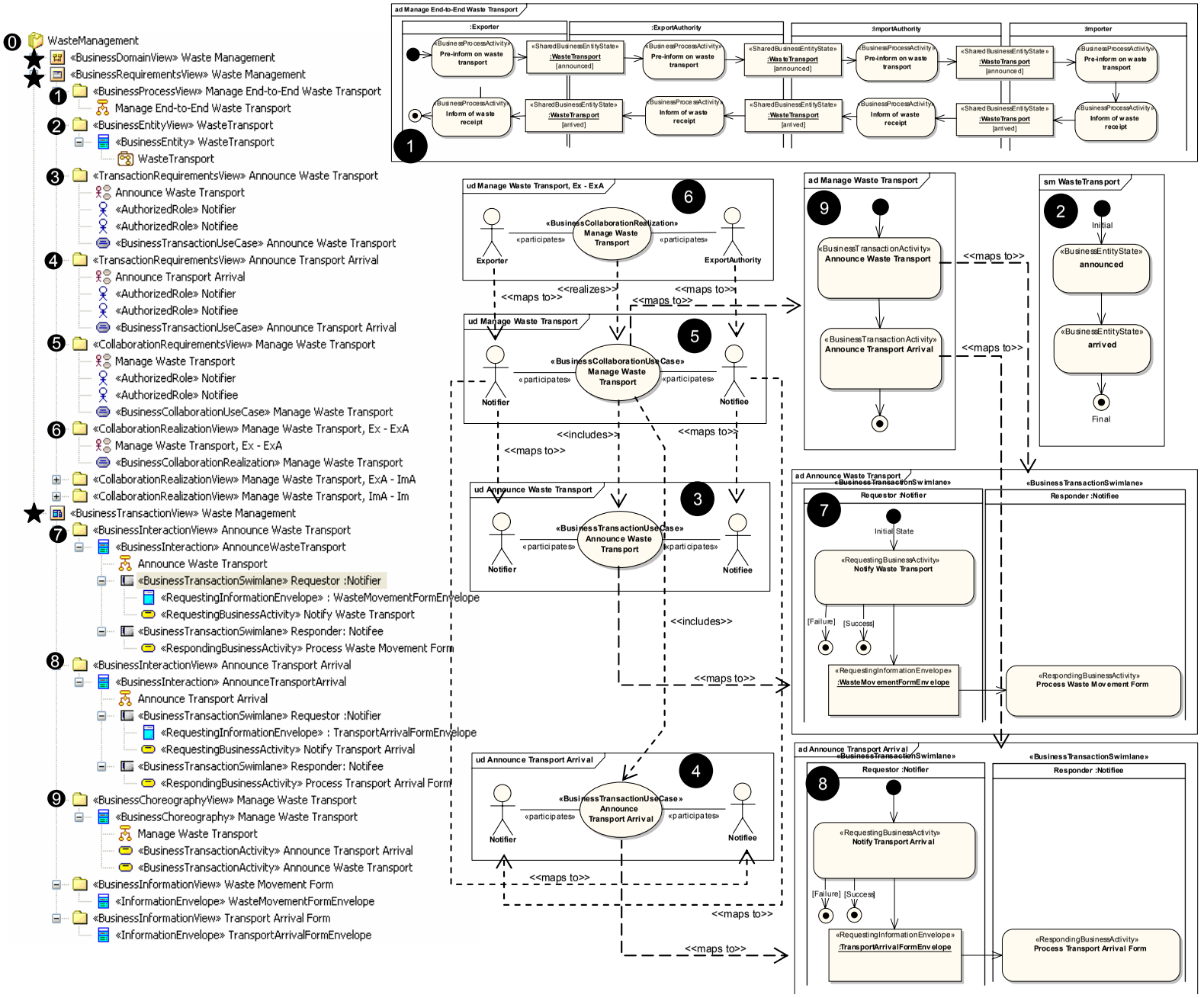

Figure 1. UMM Waste Management Model

2. A Guide to UMM

UN/CEFACTs Modeling Methodology (UMM) is defined as a UML profile - i.e. a set of stereotypes, tagged values and constraints - in order to customize the UML meta model for the special purpose of modeling global choreographies [17]. In this section we briefly describe the steps of UMM and the resulting artifacts. For a better understanding we explain UMM by the means of a very simple, but realistic example in the European `waste management` domain. The cross border transport of waste - even within the EU - is subject to regulations involving the exporter, the importer, and competent authorities in their countries as well as in transit countries. In order to keep the example simple we do not consider the transit countries here.

The relevant artifacts of our example are depicted in figure 1. On the left hand side of this figure we see the structure of our `waste management` model. A UMM *business collaboration model* comprises three main views: the *business domain view* (BDV), the *business requirements view* (BRV), and the *business transaction view* (BTV). The three top level packages of any UMM model - each highlighted with a star in the structure of figure 1 - are always stereotyped in accordance to these views.

The BDV is used to gather existing knowledge. The business process analyst will interview domain experts and stakeholders to get a basic understanding of the business processes relevant in the domain. Due to space limitations we will not elaborate on the *business domain view* of our example. Those business processes from the BDV that provide a chance for collaboration will be further detailed by the business process analyst in the BRV.

The BRV consists of a number of different subviews. The *business process view* (1) and the *business entity view* (2) are both very project specific. The *business process view* gives an overview about the business processes, their activities and resulting effects, and the business partners executing them. The activity graph of a business process may describe a single partners process, but may also detail a multi-party choreography. 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. In our example we detail a multi-party business process for a `waste transport`. The `exporter` pre-informs the `export authority` about a waste transport. The `export authority` then pre-informs the `import authority`, and the `import authority` pre-informs the `importer`. Once, the waste is received the information goes the opposite direction along the chain. The `importer` informs the `import authority`, the `import authority` informs the `export authority`, and the `export authority` informs the `exporter`. In reality, there is also information sent about the waste disposal - which we do not consider here to keep the example as small as possible.

The information exchanged between *business partners* is about the *business entity* `waste transport`. Firstly, a `waste transport` object is created with state `announced`. Later it is set to the state `arrived`. These so-called *shared business entity states* must be in accordance with the *business entity lifecycle* of `waste transport`. This lifecycle is defined in the state chart of the *business entity view* (2).

It is easy to recognize from the requirements captured so far that the same two tasks always take place between a different pair of *business partners*. Thus, it is not appropriate to describe these tasks for each pair again and again. Instead, these tasks are defined between *authorized roles*. A *transaction requirements view* defines the *business transaction use case* for a certain task and binds the two *authorized roles* involved. The *authorized roles* are defined in the exact context of the *business transaction use case*. In our example we have two *transaction requirement views*: `announce waste transport` (3) and `announce transport arrival` (4). The *authorized roles* are in both cases a `notifier` who makes the corresponding announcement and the `notifee`. However, the *authorized role* `notifier` in `announce waste transport` is not the same as the one in `announce transport arrival`. This means, we have two *authorized roles* `notifier`, each defined in the namespace of its *transaction requirements view*. On the left hand side of figure 1 we see that both *transaction requirements views* (3,4) include a `notifier`. Of course the same is valid for the `notifee`.

The *collaboration requirements view* includes a *business collaboration use case*. The *business collaboration use case* aggregates *business transaction use cases* or nested *business collaboration use cases*. This is manifested by *include* associations. In our example the *business collaboration use case* `manage waste transport` (5) includes the *business transaction use cases* `announce waste transport` (3) and `announce transport arrival` (4). Furthermore, the *authorized roles* participating in the *business collaboration use case* must be defined within the context and namespace of the *collaboration requirements view*. Sometimes it is hard to find a good name for an *authorized rol*e, like in our example. We call the roles again `notifier` and `notifiee`. The `notifier` is the one who initiates the management of a waste transport and the `notifiee` is the one who reacts on it. *Maps to* dependencies are used to define which *authorized role* of a *business collaboration use case* plays which role in an included *business transaction use case* (or nested *business collaboration use case*). In our example the `notifier` of `manage waste transport` (5) plays also the `notifier` of `announce waste transport` (3), but plays the `notifiee` in `announce transport arrival`

(4) since the information flows the other way round. For the `notifiee` of `manage waste transport` it is just the opposite.

A *collaboration realization view* covers a *business collaboration realization* - which is a kind of use case that does not elaborate any new requirements. A *business collaboration realization* realizes a *business collaboration use case* between a specific set of *business partners*. This is indicated by a *realize* association. The *business collaboration realization* `manage waste transport` (6) realizes the *business collaboration use case* with the same name (5). The *business partners* participating in the *business collaboration realization* are the ones already defined in the BDV and, thus, are not re-defined in the namespace of the *collaboration realization view*. A *maps to* dependency defines which participant of a *business collaboration realization* plays which role of the *business collaboration use case*. In the first `manage waste transport` realization (6) the `exporter` plays the `notifier` and the `export authority` acts as `notifiee`. The two additional `manage waste transport` realizations between `export authority` and `import authority` and between `import authority` and `importer` are not depicted in figure 1. This concludes all the subpackages of the BRV.

The BTV builds upon the BRV and defines a global choreography of information exchanges and the document structure of these exchanges. In this paper we concentrate on the choreography aspect and do not consider the document structures any further.

The choreography described in the requirements of a *business transaction use case* is represented in exactly one activity graph of a *business transaction*. A *maps to* dependency between them allow traceability between the requirements and the *business transaction*, which is defined in a *business interaction view*. In our example, the `announce waste transport` requirements (3) are mapped to a corresponding choreography (7). The same mapping is made for the `announce transport arrival` requirements (4+8).

A *business transaction* is characterized as follows: If an *authorized role* recognizes an event that changes the state of a *business entity*, it initiates a *business transaction* to synchronize with the collaborating *authorized role*. A *business transaction* is an atomic unit that leads to a synchronized state in both information systems. We distinguish one-way and two-way *business transactions*: In the former case, the initiating *authorized role* reports an already effective and irreversible state change that the reacting *authorized role* has to accept. In the other case, the initiating partner sets the *business entity/ies* into an interim state and the final state is decided by the reacting *authorized role*. It is 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. Irreversible means that returning to an original state requires compensation by another *business transaction*.

Owing to this strict definition, a UMM *business transaction* follows always the same pattern: A *business transaction* is performed between two *authorized roles* that are already known from the *business transaction use case* and that are assigned to exactly one swimlane each. Each *authorized role* performs exactly one activity. An *object flow* between the *requesting* and the *responding business activity* is mandatory. An *object flow* in the reverse direction is optional. Both *business transactions* `announce waste transport` (7) and `announce transport arrival` (8) are one-way transactions which do not return any information.

The requirements described in a *business collaboration use case* are choreographed in the activity graph of a *business collaboration protocol*, which is defined in a *business choreography view*. This one-to-one relationship is denoted by another *maps to* dependency. In our example, the `manage waste transport` requirements (5) are mapped to the homonymous *business collaboration protocol* (9). A *business collaboration protocol* choreographs a set of *business transaction activities* and/or *business collaboration activities*. A *business transaction activity* is refined by the activity graph of a *business transactio*n. In our example, the *business collaboration protocol* of `manage waste transport` (9) is a simple sequence of two *business transaction activities*: `announce waste transport` and `announce transport arrival`. Each of them is refined by its own *business transaction* (7,8). *Maps to* dependencies keep track of this refinement. *Business collaboration activities* - which are not used in our example - are refined by a nested *business collaboration protocol*. We do not further concentrate on the *business information views* which are used to define the structure of *business documents* exchanged in *business transactions*.

So far we have described all UMM artifacts as they would have been created in the scope of a particular project. We already took advantage of reusing a choreography. The `manage waste transport` collaboration is realized between different pairs of *business partners*. This kind of reuse is defined within a model. However, one may want to reuse artifacts that were created in another project. Consider, the `announce transport arrival` transaction was already defined in some project in the logistics domain. Instead of redefining the transaction in each project again, the corresponding *transaction requirements view* (4) and the *business interaction view* (8) are imported into the model. In order to guarantee such a scenario all the stereotypes of packages introduced before inherit the following list of tagged values from an abstract stereotype *business*

library package: baseURN, status, version, description, owner and copyright.

3. Requirements on a Business Collaboration Registry Model

A registry information model must support the registration of UMM models and parts thereof as well as keeping these parts in consistency. In this section we outline the requirements on such a registry information model - which we call *business collaboration registry model*.

First, we have to address the UMM artifacts that may be reused in different models/projects. The BDV as well as the *business process view* (1) and the *business entity view* (2) of the BRV provide project specific information. Candidates for reuse are the elements of the *transaction requirements view* (3,4) and the *collaboration requirements view* (5), because both provide generic concepts. These generic concepts are bound to specific projects by the concept of *business collaboration realizations*. It follows, that *business collaboration realizations* are project specific and hence the *collaboration realization view* (6) is not subject for reuse. All packages of the *business transaction view* - the *business interaction view* (7,8), the *business choreography view* (9) and the *business information view* (not detailed in this paper) - are candidates for reuse.

Taking a closer look on the *business transaction use case* `announce waste transport` we recognize that the artifact is not only built by the use case itself but also by the associated *authorized roles*. Accordingly, a *transaction requirements view* package (3,4) consisting of a *business transaction use case* and the two participating *authorized roles* must not be split during registration. Similarly, the activity graph of a *business transaction* is composed of partitions, activities, object flow states, etc. This information sitting in a *business interaction view* package (7,8) must be kept together during registration. Furthermore it is necessary to keep together the content of a *collaboration requirements view* package (5). Also the content of *business choreography view* package (9) is an indivisible unit. It follows that we always register complete packages.

Another aspect is that the requirements and the resulting choreography build a logical unit. Due to the UMM meta model they are always split into two different packages. Nevertheless, a registration makes only sense if both packages are submitted. This means, a *transaction requirements view* and the corresponding *business interaction view* will be two registry objects intrinsically tied together (3+7,4+8). The same is true for a *collaboration requirements view* and a *business choreography view* (5+9). As outlined in section 2, a business collaboration is built by business transactions and/or nested business collaborations. As a consequence, registration of a business collaboration (5+9) will only be successful if its constituents (3+7 and 4+8) are already registered or contained in the same submission request.

In order to realize the dependencies described in the paragraph above, UMM has well-defined relationships between its stereotypes. As outlined in figure 1 *include* and *maps to* relationships connect model elements crossing different packages. On the one hand side these dependencies exist on model level and must be expressed in a model interchange format. On the other hand side it is necessary to interlink the registry objects representing the packages containing the corresponding model elements for reasons of consistent registry management. As a result we identify the following dependencies between registry objects representing UMM packages:

- Transaction Requirements View : Business Interaction View (1 : 1)
- Collaboration Requirements View : Business Choreography View (1 : 1)
- Collaboration Requirements View : Transaction Requirements View (1..n : 0..m)
- Business Choreography View : Business Interaction View (1..n : 0..m)
- Collaboration Requirements View : Collaboration Requirements View (0..n : 0..m)
- Business Choreography View : Business Choreography View (0..n : 0..m)

4. Business Collaboration Registry Model

4.1. UMM Registry Object Format

Since UMM is defined as a UML profile [17] we propose the XML Metadata Interchange (XMI) [11] format for exchanging and storing artifacts. XMI is defined as a generic interchange format applicable for exchanging any MOF-based meta model [10]. In the past XMI became popular as an exchange format for UML models. In this paper we do not concentrate on the XMI representation of a UMM model. The interested reader is refered to our previous work [6].

Each *business library package* results in a seperate XMI fragment. The machine-readable XMI representation allows navigating easily between the model elements within one package. We learned that UMM requires dependencies between model elements in different packages. This means that the source and the target of a dependency are represented in different XMI fragments. In XMI navigation between model elements of different packages is realized by specifying the ID of the target element within an attribute of

the source element. Since these IDs are unique across package boundaries, navigating the cross package depedencies is possible. However, it is required that the XMI fragments of inter-linked packages are accessible within the registry as well as within a UMM modeling tool after retrival and import. The underlying registry management is described in section 5.1.

4.2. Mapping UMM to ebRIM

In order to foster reuse of UMM artifacts storing the model or parts thereof as opaque units of XML streams is insufficient. Rather, there is a need for a meaningful meta model that facilitates the search for artifacts and the maintenance of connections between dependent artifacts. Thus, we introduce the *business collaboration registry model* on top of ebRIM [9]. It defines a mapping from relevant UMM information to ebRIM concepts. A conceptual overview of the *business collaboration registry model* is shown in figure 2. Figure 3 shows an instance of the *business collaboration registry model* of our `waste management` example.

Each XMI fragment representing a *business library package* is stored as an *extrinsic object* as defined in ebRIM [9]. In case of the `waste management` example this results in seven *extrinsic objects*: the entire `waste management` model (0), the *transaction requirements view* and the *business interaction view* for `announce waste transport` (3,7) as well as for the `announce transport arrival` (4,8), and the *collaboration requirements view* togehter with the *business choreography view* of `manage waste transport` (5,9). The extension mechanism of ebRIM allows us to create subtypes of *extrinsic object* - one for each above mentioned stereotype. Figure 2 shows meta classes for these stereotypes flagged as *extrinsic objects*.

In section 3 we learned that dependencies between these package stereotypes must be realized in the *business collaboration registry model*. These dependencies are summarized in the bullet list at the end of section 3. This list results in ebRIM *associations* between the relevant meta classes in figure 2. Each *association* is further required to specify an *association type*. The ebRIM *classification scheme* for *association types* provides a set of basic types for *associations*. The standard association types *contains* and *implements* are sufficient for our *business collaboration registry model*.

Each *business transaction* is stored as an *extrinsic object* of type *business interaction view* (7,8). The corresponding *business transaction use case* and the participating roles are stored in the *extrinsic object* of type *transaction requirements view* (3,4). The relationship between the *transaction requirements view* and the *business interaction view* is denoted in the registry meta data as an *association* of type *implements*. The requirements captured within the *transaction requirements view* are extracted as meta data using two *slots*. The first *slot* named *actions* is used to describe the activities within the *business transaction*. The second *slot definition* captures the transaction's purpose. The *transaction requirements view* has further two *classifications* associated that specify the participating *authorized roles* defined within a *classification scheme* called *role*.

A business collaboration and its requirements are registered in the *extrinsic objects* of the packages *collaboration requirements view* and *business choreography view* (5,9). A *collaboration requirements view* has the same two types of *slots* - *actions* and *definitons* - associated. Similar to transactions, we denote the participants in a business collaboration using the *classification schema* named *role*. A *classification object* is created for each of its participants. However, a business collaboration might have more than two participants.

Zero or null *business transactions* might be part of a *business collaboration protocol*. Thus, we create *contains* associations leading from the *business choreography view* to each included *business interaction view*. In the `waste management` example the *business collaboration protocol* is composed of two *business transactions*. This is shown by the two *associations* leading from 9 to 7 and from 9 to 8. Similarly, the corresponding *collaboration requirements view* has zero or more *transaction requirements views* associated (5 to 3 and 5 to 4). The *contains* associations between *collaboration requirements views* and *transaction requirements views* correspond to the *include* associations between *business collaboration use cases* and *business transaction use cases*. The concept of nested business collaborations - which is not used in our `waste management` example - is denoted via the self-directed *contains* associations attached to the *business choreography view* as well as to the *collaboration requirements view*.

If an entire model is registered, the registry will create *contains* associations to *business choreography views*, *collaboration requirements views*, *business interaction views*, and *transaction requirements views*. In the `waste management` case six *contains* associations from 0 to 3, 4, 5, 7, 8, and 9 are established. Registering an entire model requires extraction of artifacts as further discussed in section 5.

Having discussed the relationships between the different types of *extrinsic objects*, we have to elaborate on the mapping of tagged values defined for UMM *business library packages* to the *business collaboration registry model*. The UMM stereotype *business libary package* provides a set of tagged values realizing a namespace concept. In ebRIM the base class *registry object* defines attributes for a similar purpose. *Extrinsic object* inherits these attributes from *registry object*. Table 1 gives a brief overview of this mapping. *Name*, *description* and *status* are mapped one-to-one. The

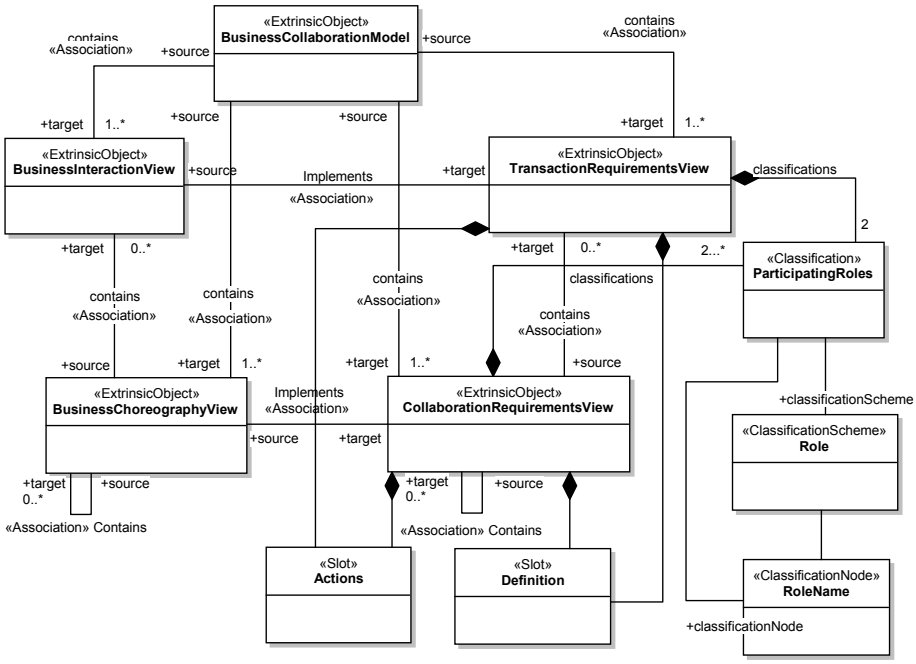

Figure 2. Business Collaboration Registry Model

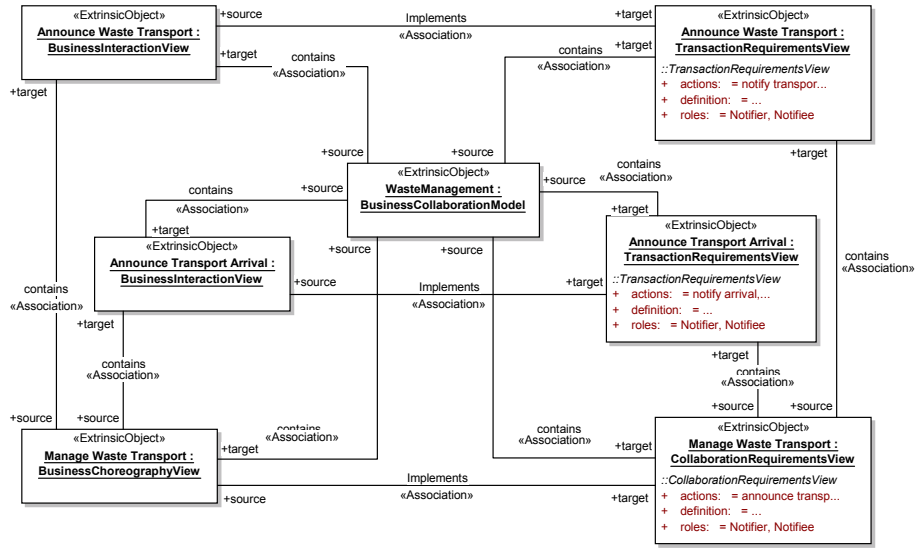

Figure 3. Waste Management Example mapped to the Business Collaboration Information Model

baseURN concatenated with the *local name* of the *business libary package* is mapped to the *logical ID* of the *extrinsic object*. The *logical ID* refers to a logical registry object independent of its version. This construct plus *version* results in the *unique ID* of a registry object. The *version* of a *business library package* corresponds to the *version name* that is part of the *version* info associated with a registry object.

tagged values	attributes
description	description
name	name
BaseURN + name	LogicalID
BaseURN + name + version	ID
version	versionInfo.versionName
status	status

Table 1. Namespaces in UMM and ebRIM

5. UMM Registry Management

5.1. Keeping Consistency

As outlined in section 3 some packages build a logical unit. The package capturing the requirements and the one capturing the flow must always go together (3 and 7, 4 and 8, 5 and 9). Furthermore, a business collaboration requires the existance of the included business transactions and/or business collaborations (5 requires 3 and 4; 9 requires 7 and 8).

The registry has to ensure that these depedencies are satisfied when artifacts are submitted or retrieved. In case of a submission it must check if all required packages are known - either within the same submission or already within the registry. If this is not the case the UMM registry will reject the registration. On retrieval a registry is likewise required to maintain consistency. A response on a retrieval request must contain the requested artifact plus all dependent parts. These requirements avoid abandoned references across XMI fragments on retrieval.

5.2. Managing Meta Data

The mapping of tagged values defined for UMM *business library packages* to the *business collaboration registry model* - as described in the end of the previous section - should not be a user's task. Thus, a UMM registry must set and map these data on the packages. The registry is required to keep the information in the XMI fragments and its registry meta data in consistence. This involves also the version control of the XMI fragments. If a new version is detected the registry is expected to synchronize the tagged values with the meta data. This means that a registry must also manipulate the corresponding information within the XMI fragments and report the results back to the submitter.

Furthermore, a UMM registry has to manage additional meta data describing the *extrinsic objects*: *actions*, *definitions*, and *participating roles* (see figure 3). This meta information must be extracted from the tagged values and/or model elements of the submitted XMI. *Actions* and *definition* correspond to the equally named tagged values of the particular use case. *Participating roles* are known since they are associated with the use case.

5.3. Submission of Models as a Whole

An entire model is submitted in a single XMI file. This XMI file covers - amongst other information - also information about packages that are themselves registrable artifacts. These parts should be extracted and registered in seperate XMI fragments in order to enable the reuse of business transactions and business collaborations. In principle there are two options. The extracted information remains in the XMI fragment of the overall model or it is cutted. For performance reasons, we selected the first option even if information is duplicated within the registry. No matter which option is realized, the registry must create *contains* associations leading from the model to the extracted artifacts. Hence, one can follow in which model a certain transaction or collaboration is used.

6. Related Work

In our approach we use UMM to describe business choreographies. UMM follows the ideas of ISO's Openedi [7] that provides a reference model for defining business processes crossing organizational boundaries. In addition to UMM there exist other approaches that are based on UML to model inter-organizational business processes: The development of RosettaNet Partner Interface Processes (PIPs) [12] is based on global choreographies described by UML. This approach was merged into UMM during the ebXML initiative in 2000. Other approaches use UML in order to describe a choreography specific to Web Services [8][13].

In the world of Web Services a lot of different languages exist to capture business processes like the Business Process Modeling Language (BPML) [1] and the Business Process Execution Language (BPEL) [2]. These languages are limited to orchestrations and local choreographies. UMM models may be transformed to local choreographies described in BPML or BPEL. In [3] we describe such a mapping from UMM to BPEL. The release of the Web Services Choreography Description Language (WS-CDL) draft [18] adds a specification for global choreographies to the family of Web Services which did not exist before. Within the ebXML framework, the Business Process Specification

Schema (BPSS) [15] always describes the choreography of a business collaboration from a global perspective. We describe a mapping from UMM models to BPSS in [4].

In our approach the *extrinsic objects* maintained in the registry are XMI files representing entire models, business transactions, or business collaborations. In [6] we describe in detail how to transform a UMM model into an XMI file keeping consistent relationships beween the UMM elements. Furthermore, we show how to classify the XMI files for various business environments. In this paper we did neither concentrate on the XMI transformation, nor on the business environment-specific classification. In other words, whereas our work in [6] concentrated on the storage and/or interchange format of UMM models, this paper deals with the meta data description of the stored models. Accordingly, this paper adds a *business collaboration registry model* maintaining relationships between different *extrinsic objects* (i.e. XMI fragments) that logically belong together.

Currently, UN/CEFACT develops a registry profile [16] to manage Core Components [14] - the basic building blocks for business documents. An approach for registering business collaborations has not been developed so far. Thus, our *business collaboration registry model* is supposed to fill this gap.

7. Summary

This paper is about managing UMM *business collaboration models* in an ebXML registry. It is evident that an entire UMM model will become a registry object. However, we started from the assumption that parts of a model may be reused in another model. Therefore, these parts of a model must become registry objects as well. We gave a brief introduction into the artifacts that together build a UMM *business collaboration model*. This helped us in identifying those kind of packages and their contents that are subject to reuse. These packages are the *transaction requirements view* together with the *business interaction view* as well as the *collaboration requirements view* together with the *business choreography view*.

We further elaborated on relationships between UMM elements in different packages. If a directed association from a model element in one package refers to a model element in another package, it is essential that the latter package always comes with the first package. It follows that a registry model has to maintain the relationships between the registry objects representing different kinds of packages.

For this purpose we defined our *business collaboration registry model*. This model extends the general ebXML registry information model (ebRIM) in order to maintain different UMM artifacts and to automatically provide a basic classification of these artifacts. Furthermore, we defined a clear registry management process for keeping consistency between dependent artifacts, keeping consistency between an artifacts content and its meta-data maintained in the registry and extracting information from models submitted as a whole. We are currently implementing these features in a prototype connecting our UMM modeling tool [5] and an ebXML registry.

References

[1] A. Arkin. Business Process Modeling Language (BPML). Technical report, June 2002.

[2] BEA, IBM, Microsoft, SAP AG and Siebel Systems. *Business Process Execution Language for Web Services*, May 2003. Version 1.1.

[3] B. Hofreiter and C. Huemer. Transforming UMM Business Collaboration Models to BPEL. In *Proceedings of OTM Workshops 2004*, volume 3292. Springer LNCS, 2004.

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

[5] B. Hofreiter, C. Huemer, P. Liegl, R. Schuster, and M. Zapletal. UMM Add-In, Mar. 2006. http://www.ifs.univie.ac.at/ummaddin/.

[6] B. Hofreiter, C. Huemer, and M. Zapletal. Registering UMM Business Collaboration Models in an ebXML Registry. In *Proceedings of the IEEE Conference on E-Commerce Technology (CEC '06)*. IEEE Computer Society, June 2006.

[7] ISO. *Open-edi Reference Model*, 1997. ISO/IEC JTC 1/SC30 ISO Standard 14662.

[8] G. Kramler, E. Kapsammer, G. Kappel, and W. Retschitzegger. Towards Using UML 2 for Modelling Web Service Collaboration Protocols. In *Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'05)*, Feb. 2005.

[9] OASIS. *ebXML Registry Information Model*, May 2005. Version 3.0.

[10] Object Management Group (OMG). *Meta Object Facility (MOF) Specification*, May 2005. Version 1.4.1.

[11] Object Management Group (OMG). *XML Metadata Interchange Specification*, June 2005. Version 2.0.1.

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

[13] S. Thöne, R. Depke, and G. Engels. Process-oriented, flexible composition of web services with uml. In *Conceptual Modeling - ER 2002, 21st International Conference on Conceptual Modeling, Proceedings*, LNCS. Springer, 2002.

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

[15] UN/CEFACT. *UN/CEFACT - ebXML Business Process Specification Schema*, Nov. 2003. Version 1.11.

[16] UN/CEFACT. *UN/CEFACT Registry Implementation Specification*, Feb. 2006. Version 0.9.

[17] UN/CEFACT. *UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module*, Mar. 2006. Candidate for 1.0, Final Working Draft, http://www.unece.org/cefact/umm/UMM_Foundation_Module.pdf.

[18] World Wide Web Consortium (W3C). *Web Services Choreography Description Language*, Nov. 2005. Version 1.0.