Towards Using UML 2 for Modelling Web Service Collaboration Protocols

Gerhard Kramler[1], Elisabeth Kapsammer[2], Werner Retschitzegger[2], and Gerti Kappel[1]

[1] Business Informatics Group, Vienna University of Technology, Austria {kramler, gerti}@big.tuwien.ac.at

[2] Department of Information Systems, Johannes Kepler University of Linz, Austria {ek, werner}@ifs.uni-linz.ac.at

Summary. In a web environment, graphical specifications of service collaborations which focus on the protocols of collaborating services are especially important, in order to attain the desired properties of interoperability and loose coupling. Different to modelling of generic software component collaborations, additional requirements must be considered, including security and transaction aspects, and the characteristics of specific target technologies such as ebXML and BPEL. This paper describes a UML-based approach for platform independent modelling web service collaboration protocols, which takes into account the specific requirements, and supports mappings to relevant target technologies.

1 Introduction

Web services are being used for the coordination of communicating processes, e.g., in the automation of business collaborations such as the standard airline ticketing example. Specification of such collaborations in a manner that facilitates interoperability and loose coupling is provided by a so-called *collaboration protocol*, which provides a global public view on multiple cooperating web services. Collaboration protocols, also called choreographies [15] or conversation policies [8], can be specified using languages like ebXML Business Process Specification Schema (BPSS) [13] and Web Services Choreography Description Language (WS-CDL) [15].

Related to collaboration protocols are interface specification and implementation of a web service. An *interface* describes the public aspects of a web service, including both its provided and required operations as well as its observable behavior. The behavioral aspect of an interface is also called choreography [5, 14], orchestration [5], and abstract process [1]. Web service interfaces are specified using languages like WSDL, BPEL, and WSCI. An *implementation* (also called executable process [1]) is the private aspect of a web service, specified by a language such as BPEL, Java, etc.

Using current Web Services languages, i.e., BPEL and WSDL for the specification of web service collaborations brings up three problems: (1) the languages are XML-based, lacking a standardized graphical representation, which would ease modelling and understanding of collaboration protocols, (2) there is no support for collaboration protocols but only for individual interfaces, leading to potential consistency problems [4], and (3) the level of abstraction is too low for conveniently expressing transactions, as specifically useful in business collaborations [3].

Although there are approaches addressing these problems, no complete solution has been found yet. There is a UML profile for BPEL [7] addressing problem (1). WS-CDL complements BPEL by addressing problem (2) but does not provide a graphical representation. BPSS addresses (3) and there is also a UML representation for BPSS [11] addressing (2), however, these approaches does not support Web Service specification languages and lack flexible specification of intra-transactional interactions. One of the origins of BPSS, the UN/CEFACT Modeling Methodology (UMM) addresses (1-3) but it is limited to the domain of business collaborations and not supporting the specifics of Web Service technology.

Our approach to cope with the identified problems is a UML-based modelling technique that supports platform independent modelling of web service collaboration protocols and that is closely aligned with BPEL and BPSS concepts. In particular, the semantics of UML 2 are refined for applying it to collaboration protocol modelling. This way, no new language and notation need to be invented and problems (1-2) are addressed. Furthermore, different levels of abstraction are supported, thereby supporting both top-down and bottom-up development and addressing problem (3).

In the next section, an overview of our proposed modelling technique is presented. The used UML diagrams and specific semantics are elaborated in sections 3–5. A brief comparison of our approach to related work is given in Section 6. The paper concludes with the open issues that need to be resolved to make the whole approach operational.

2 The Big Picture

The main idea of our approach is to explore UML's existing modelling concepts for collaboration protocol modelling. Therefore, we attempt to identify modelling concepts in UML that are similar to those of the target technologies. Since in many cases no direct equivalence can be found, we first define a platform independent modelling technique, and in a second step define a mapping to specific platforms.

We identify the main specification concepts of the target technologies based on a previously conducted comparison [3]. In that comparison, the layered architecture of the eCo framework [6] was used as basis for comparison, including the layers information items, documents, interactions, and services.

- The concepts dealt with in the *information items* and *documents layer* are the data structures of documents being exchanged among the participants of a collaboration. Re-usable data structures are of particular interest here.
- In the *interaction layer*, the main concept is the message sent from one participant to another, conveying documents. A meaningful interaction is formed by one or more message exchanges.
- The *service layer* considers transactions among participants, and the composition of transactions to build up complex services.
- Furthermore, the *business and market layers*, which were not used in [3] but will be in this paper, consider different types of participants based on the set of services they provide to and require from related participants.

The realization of these layers and concepts in the target technologies, i.e., ebXML and WSDL/BPEL, has been discussed in [3]. For the creation of an UML-based PIM, we need to find modelling concepts that are suitable to both target technologies and that are supported by UML.

UML provides a range of diagrams that cover the concepts in the above named layers. We propose the use of five kinds of UML models as shown in Fig. 1. The models are arranged in a layered architecture which - due to the idiosyncracies of UML - is different from the eCo one. The different levels are interrelated by refinement and usage relationships, meaning that elements specified in one level are refined at lower levels, and conversely, specification elements can be re-used at upper levels. Thus both top-down and bottom-up development is supported.

The *collaboration level* roughly corresponds to eCo's business and market layers. It is concerned with participants and their collaboration and communication relationships, thus providing an overview of a collaboration. This is expressed in the collaboration model, a UML collaboration diagram.

The *transaction level* corresponds to the services layer of eCo and considers transactions and transactional processes, each transaction being performed by a set of participants in collaboration. This level abstracts from the distribution of state and control in a collaboration to provide a convenient high-level model. Two kinds of models are proposed in this level. The *activity model*, a UML activity diagram, defines transactional processes, which refine collaborations as defined in the collaboration level. The *object model* uses class diagrams and protocol state machines to define the attributes and states of objects that are used as pre- and post-conditions of transactions.

The *interaction level* covers the eCo interactions, documents, and information items layers. It specifies the messages actually exchanged among participants, thus being at a low level of abstraction. Again two kinds of models are proposed. The *interaction model*, a UML interaction diagram, defines the details of a transaction in terms of message exchanges among the participants, thus refining the individual transactions of the activity model. The *message content model* is a class diagram defining the content of messages.

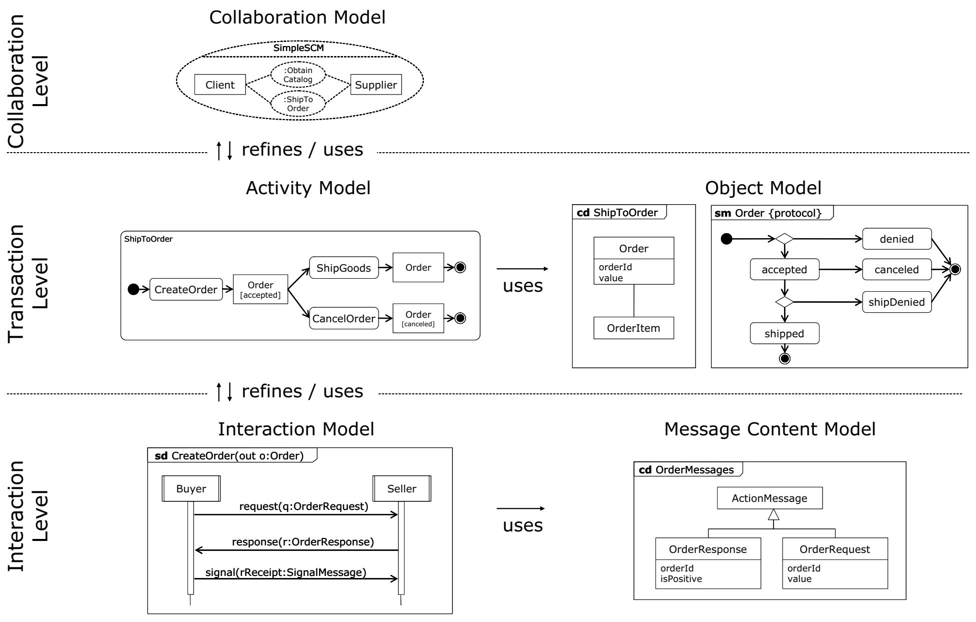

Fig. 1. Abstraction levels and kinds of models

The following sections discuss how UML is employed to realize the models introduced above, i.e., how a subset of UML 2 is used and what the specific interpretation of the employed UML concepts is.

3 Collaboration Level

UML collaboration diagrams have been chosen for modelling of the business and market layers, because they can express the roles of participants interacting within an overall collaboration. Furthermore, behavior models that refine a collaboration model are always a specification of behavior emerging from the behavior of the individual participants [12], as opposed to specification of executable behavior which is based on a centralized control. Naturally, collaboration protocols are a specification of emergent behavior.

All of the modelling concepts of UML's collaboration diagrams are used in the collaboration model. Of particular interest is the nesting of collaborations, supporting the definition of composite services or of business areas.

Example 1. Fig. 2 (left) shows the "SimpleSCM" (Simple Supply Chain Management) collaboration between two roles, client and supplier. The collaboration uses two sub-collaborations, the specification of one of them is included in Fig. 2 (right). There is no behavior attached to the "SimpleSCM" collaboration, meaning that the two sub-collaborations can be performed independent

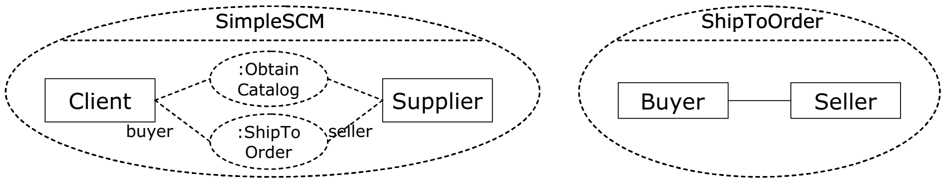

Fig. 2. A simple supply chain collaboration (left) and the sub-collaboration "ShipToOrder" in detail (right)

of each other. The sub-collaboration "ShipToOrder" specifies that the two roles, "Buyer" and "Seller", communicate with each other. The behavior of that collaboration will be further discussed in Example 2.

4 Transaction Level

4.1 Activity Model

The activity model specifies the behavior of a collaboration in terms of transactional processes, using UML activity models. A UML activity is used to define a transactional process, each action within that process representing a transaction performed collaboratively by two or more participants of the overall collaboration. Each transaction may in turn be refined by another activity model, or by an interaction.

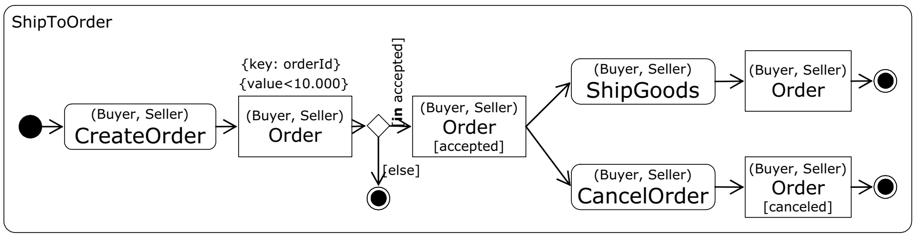
ShipToOrder
(Buyer, Seller) CreateOrder
{key: orderId}
{value<10.000}
(Buyer, Seller) Order
[in accepted]
[else]
(Buyer, Seller) Order [accepted]
(Buyer, Seller) ShipGoods
(Buyer, Seller) Order
(Buyer, Seller) CancelOrder
(Buyer, Seller) Order [canceled]

Fig. 3. Activity model of the "ShipToOrder" collaboration

The input and output parameters of an action define its pre- and post-conditions in terms of collaborative objects (cf. Section 4.2) consumed and produced by the action, respectively. In UML, object nodes are used to define the input and output parameters. Like actions, also object nodes must be assigned to at least two participants, meaning that the pre- and post-conditions apply to those participants.

We emphasize that - opposite to the usual interpretation of activity diagrams - a collaboration activity does not prescribe centralized control. Rather, it represents emergent behavior of all the participants, i.e., each of the participants must behave such that the resulting behavior of the overall collaboration corresponds to the defined collaboration activity. To ensure that a collaboration protocol can be realized without central control, certain restrictions must be met. In particular, a control flow can only be modelled between actions which share at least one participant, otherwise no one of the participants of the succeeding action would have knowledge about when to start. Similarly for object flows. An object flow is only allowed if the target object node is assigned to a subset of the participants that the source object node is assigned to, ensuring that the pre-condition represented by the target object node is known to the participant who is obliged to it.

As an extension to standard UML, we introduce the *key constraint* which specifies for an object node that some of the attributes of the objects contained in the node must be unique among all concurrent instances of the activity. In other words, the key attributes must unambiguously identify the activity instance. This is corresponding to BPELs concept of correlation set. Since key constraints are not natively supported by UML, the proprietary notation "{key: field$_1$, field$_2$, ...}" is introduced.

Example 2. The activity shown in Fig. 3 defines the behavior of the "ShipToOrder" collaboration. It comprises the "CreateOrder" action which, in case of success, is followed either by "ShipGoods" or "CancelOrder", based on a non-deterministic choice. Object nodes are used to define pre- and post-conditions on the actions, in particular, state constraints and a key constraint.

An activity may be used to specify the behavior of a collaboration. If the collaboration is used as a top-level collaboration, its activity must not have input and output parameters (e.g., as in Fig. 3). Furthermore, if the collaboration is composed of sub-collaborations, the composite collaboration's activity must include (i.e., invoke) its constituent collaborations' activities.

4.2 Object Model

The object model specifies the objects (i.e., both data structure and their behavior) that the collaborative transactions operate on. These objects represent the knowledge common to some or all participants of the collaboration. Note that objects are different from messages. Messages specify the data exchanged among participants, whereas objects specify requirements on the participants' data resources. In many cases, messages represent updates to the objects. Note that this kind of model has not been considered in eCo nor in ebXML or BPEL, but naturally arises by our proposed way of using UML.

For the purpose of collaboration protocol modelling, only those attributes and states are necessary which are needed for defining the coordination logic,

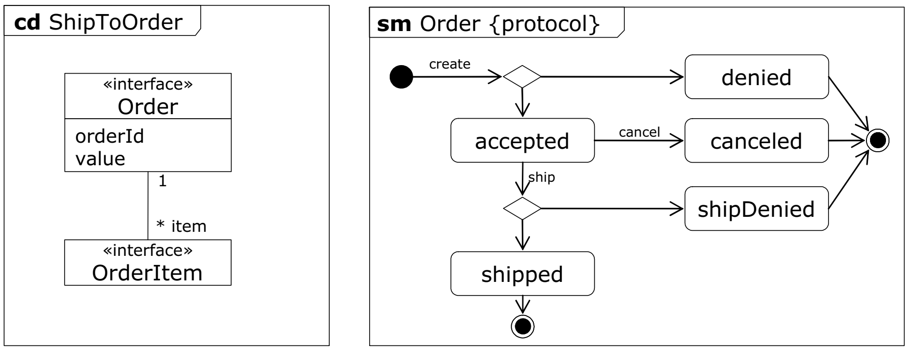

Fig. 4. A data structure used in the collaboration (left) and its permissible behavior (right)

including control flow and data flow constraints. If no decision conditions and no constraints on data are needed, the collaboration object model can be omitted.

Since the collaboration object model must not prescribe the objects used by the participants internally, only interfaces and protocol state machines are used, depicted in a class diagram and state machine diagrams respectively. An *interface* specifies the structure of collaboration-relevant data, whereas a *protocol state machine* specifies the states and permissible state transition of such an interface. The states are used as pre- and post-conditions of transaction specifications, and the transitions are used for consistency checks with the use of objects in activity models. Therefore neither triggering events nor transition conditions need to be specified formally.

Example 3. Fig. 4 (left) shows a class diagram for the data relevant to the "ShipToOrder" collaboration. The specification of the permissible states of an "Order" is shown in the protocol state machine in Fig. 4 (right). Note that transition events are modelled only informally, transition conditions are modelled not at all.

5 Interaction Level

5.1 Interaction Model

The interaction model specifies the interactions among the participants of a transaction in terms of asynchronous message exchanges. The behavior of individual participants is considered, i.e., the notion of a shared state is no longer maintained, but rather different states of the participants and the means of synchronization and coordination need to be defined. Interaction models are intended to be used at a low level of granularity and complexity with request/response as minimal interaction patterns. They refine actions or collaborations. If an interaction model is used to refine an individual action, its parameters must be compatible.

Interaction models must specify how participants achieve a common outcome of the transaction, i.e., at the end of an interaction the participants must know the common outcome in terms of the interaction's output data and the transaction's state. For reasons of loose coupling, the tasks performed by the participants are out of scope of a collaboration model. It is only the overall transaction which represents a common/synchronous task.

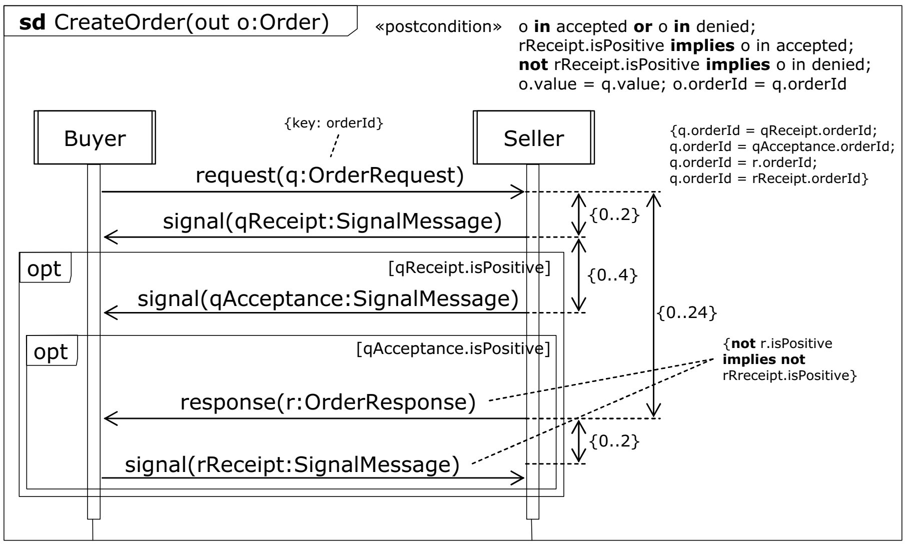
sd CreateOrder(out o:Order)
«postcondition» o in accepted or o in denied; rReceipt.isPositive implies o in accepted; not rReceipt.isPositive implies o in denied; o.value = q.value; o.orderId = q.orderId
Buyer
Seller
{key: orderId}
{q.orderId = qReceipt.orderId; q.orderId = qAcceptance.orderId; q.orderId = r.orderId; q.orderId = rReceipt.orderId}
request(q:OrderRequest)
signal(qReceipt:SignalMessage)
{0..2}
opt
[qReceipt.isPositive]
{0..4}
signal(qAcceptance:SignalMessage)
{0..24}
opt
[qAcceptance.isPositive]
{not r.isPositive implies not rRreceipt.isPositive}
response(r:OrderResponse)
{0..2}
signal(rReceipt:SignalMessage)

Fig. 5. Interaction refining the "CreateOrder" action

The contents of individual messages are interrelated by means of *data flow constraints* rather that operation input/output. In particular, a deterministic data flow constraint defines equality of attributes in different messages or parameters. They are required for key constraints and for specification of postconditions (see below). To ensure realizability of the collaboration protocol, data flow constraints must be defined based solely on interaction parameters and message arguments, such that both sender and receiver of the involved messages are able to observe the constraint variables.

Similar to the activity model, also the interaction model is extended with the *key constraint* which specifies, for a message, that some of the message's arguments must be unique among all concurrent instances of the interaction. In other words, the message must unambiguously identify the interaction instance based on the specified key. Sender and recipient (lifelines) may also be used as key components.

The relationship between messages in the interaction model and objects in the activity model is established by means of *post-conditions*. A post-condition specifies the outcome of the interaction in terms of data flow constraints on its output parameters. The post-condition constraints must be observable by all participants who are assigned to the respective output parameter, and the resulting value of all output parameters must be defined.

Example 4. The interaction depicted in Fig. 5 corresponds to a BPSS interaction pattern. It comprises a request and a response message, and accompanying acknowledgement signal messages. Positive acknowledgement messages are a pre-requisite for the interaction to proceed successfully. The data flow constraints shown on the right correlate the "orderId" attributes of the "OrderRequest" and "OrderResponse" messages, and, conditionally, the "isPositive" attributes of the "OrderResponse" and "rReceipt" messages. The post-condition defines the state and value of the output parameter. In particular, the "order" is in state "accepted" only if "rReceipt" was positive, otherwise it is in state "denied".

Within specific domains transactions will often be defined based on generic interaction patterns such as the one used in Example 4. Such generic interaction patterns can be supported by means of interaction templates having template parameters for specification of participants, message types, and timing constraints.

5.2 Message Content Model

The message content model specifies the requirements on message contents. The message model can be specified either completely or in an abstract way. A complete message model defines the message contents unambiguously, i.e., all exchanged documents are specified. Conversely, an abstract message model specifies only the minimal requirements, as needed for the collaboration specification. An abstract message model facilitates re-use of variants of complete message models, e.g., different business document standards could be supported.

The concepts used in the message model are interfaces and classes. *Interfaces* are used to specify an abstract message model, whereas *classes* are used to specify complete message models. For specifying XML-related characteristics of message contents, an appropriate UML profile has to be used [2].

Example 5. The example in Fig. 6 shows an abstract message model for the messages in the "ShipToOrder" collaboration. The messages are based on generic ebXML messages "SignalMessage" and "ActionMessage".

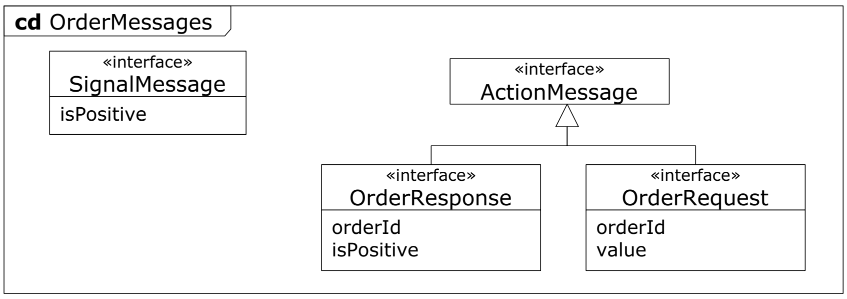

Fig. 6. Abstract messages used by the "CreateOrder" interaction

6 Related Work

Several approaches to graphically modelling collaboration protocols exist, related to BPSS and based on UML. Considering the web services area, most research deals with modelling of interfaces and implementation of individual web services rather than with collaboration protocols.

Kim [11] takes an approach very similar to ours in that he investigates how UML (version 1.x) diagrams can be used to graphically specify collaboration protocols with an automatic mapping to BPSS. His solution covers the transaction level and the interaction level. At the transaction level, activity diagrams are used. Furthermore, sequence diagrams are used also at the transaction level, with synchronous messages used to represent transactions. This way, behavior of multi-party collaboration protocols is modelled. At the interaction level, both interaction diagrams and class diagrams are used. The major difference of our approach is that we support data flow constraints, both at the interaction level and at the transaction level. Furthermore, our approach is not bound to the limitations imposed by BPSS and is therefore more expressive regarding possible interaction patterns and collaboration processes.

UMM [10] provides not only a rich set of UML-based modelling concepts for B2B collaboration protocols, but also methodological guidance ranging from requirements elicitation to implementation design. UMM has provided the conceptual foundation of BPSS, and since then it was further improved. In particular, it now supports so-called business entities, i.e., business domain objects which are modelled in terms of class diagrams and state diagrams. Furthermore, a business collaboration protocol (the equivalent to a BPSS binary collaboration protocol) can use business entities to define pre- and post-conditions of business transactions. Business entities and their use in business collaboration protocols are very similar to our object model and its use in the activity model. The difference is that business entities capture business semantics, whereas our object model is defined in technical terms. In particular, our object model is formally connected to the messages exchanged in the interaction level, thereby creating an aggregated form of data flow constraint, which is not the case with business entities. As a result, our approach can be directly mapped to the implementation level, which is not the case with UMM.

There exists also a mapping from a subset of UMM to BPEL [9]. It supports the UMM/BPSS interaction patterns, as well as the control flow of business collaboration protocols. In comparison to our approach, that mapping is elaborated in full detail. It considers, however, only a subset of the concepts defined in our models. Difficult mapping problems, e.g., non-deterministic choice, or mapping of failure handling, are still open issues.

The approach described in [8, 4] is not restricted to the business domain but supports web service collaboration protocols in general. Collaboration protocols are specified in terms of a state machine, with states representing the global state of the collaboration, and state transitions representing messages exchanged among its participants. A strong point of this work is that it supports formal verification of consistency between global behavior, i.e., the collaboration protocol, and local behavior of participants, i.e., the interface. In contrast to our approach, only the interaction level is considered, no notion of transaction is provided. Furthermore, it is a purely conceptual model without graphical notation.

7 Summary and Outlook

We have presented a technique for modelling collaboration protocols, which seeks to support the main concepts of both BPSS and BPEL by exploring the features of UML 2. The modelling technique generalizes some of the key concepts of BPSS and UMM, resulting in a language which is no longer specific to the B2B domain but rather supports generic transactional collaboration protocols.

However, several important issues remain open for further research:

- Specification of failures and failure handling has not yet been addressed. At the transaction level, extensions to the activity model are required that cope with failure handling in order to realize transactional properties of long running transactions. At the interaction level, failures of the messaging system have to be considered. Without these extensions, the corresponding aspects need to be defined at the implementation level.
- Non-functional properties such as security and transactional characteristics are still missing. In particular, the respective requirements of BPSS are of interest.
- The mapping to the target technologies has to be elaborated in full detail, to enable automatic code generation. For BPSS this will be straight forward, a mapping to BPEL however amounts to the specification of a protocol implementation. In addition to BPEL and BPSS, support for WS-CDL would be logical but has not yet been considered.

- Finally, to support better integration in a software development process, it would be interesting to consider the relationship of collaboration protocol models to models of web service interfaces and deployments.

References

1. BEA, IBM, Microsoft, SAP, Siebel (2003) Business Process Execution Language for Web Services, Version 1.1. `http://ifr.sap.com/bpel4ws/BPELV1-1May52003Final.pdf`
2. Bernauer M, Kappel G, Kramler G (2004) Representing XML Schema in UML – A Comparison of Approaches. In: Proceedings of the 4th International Conference on Web Engineering (ICWE2004)
3. Bernauer M, Kappel G, Kramler G, Retschitzegger W (2003) Comparing WSDL-based and ebXML-based Approaches for B2B Protocol Specification. In: Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC 2003)
4. Bultan T, Fu X, Hull R, Su J (2003) Conversation specification: a new approach to design and analysis of e-service composition. In: WWW 2003
5. DERI (2004) Web Service Modeling Ontology - Standard, WSMO Working Draft. `http://www.wsmo.org/2004/d2/v02/`
6. eCo Working Group (1999) eCo Architecture for Electronic Commerce Interoperability. `http://eco.commerce.net/rsrc/eCoSpec.pdf`
7. Gardner T (2004) UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. In: Object-Oriented Technology: ECOOP 2003 Workshop Reader, ECOOP 2003 Workshops, Darmstadt, Germany, July 21-25, 2003, Final Reports. Springer LNCS 3013
8. Hanson J E, Nandi P, Kumaran S (2002) Conversation support for Business Process Integration. In: Proceedings 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC-2002)
9. Hofreiter B, Huemer C (2004) Transforming umm business collaboration models to bpel. In: Proc. of the OTM Workshop on Modeling Inter-Organizational Systems (MIOS 2004)
10. Hofreiter B, Huemer C, Naujok K D (2004) Un/cefact's business collaboration framework - motivation and basic concepts. In: Proc. of the Multi-Konferenz Wirtschaftsinformatik (MKWI 2004)
11. Kim H (2002) Conceptual Modeling and Specification Generation for B2B Business Processes based on ebXML. In: SIGMOD Record Volume 31
12. OMG (2003) UML 2.0 Superstructure Specification. OMG Adopted Specification ptc/03-08-02
13. UN/CEFACT and OASIS (2001) ebXML Business Process Specification Schema, Version 1.01. `http://www.ebxml.org/specs/ebBPSS.pdf`
14. W3C (2002) Web Service Choreography Interface (WSCI) 1.0, W3C Note. `http://www.w3.org/TR/wsci`
15. W3C (2004) Web Services Choreography Description Language Version 1.0, W3C Working Draft. `http://www.w3.org/TR/ws-cdl-10/`