Towards Using UML 2 for Modelling Web Service Collaboration Protocols

Gerhard Kramler[1], Elisabeth Kapsammer[2], Werner Retschitzegger[2], 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`

Abstract. 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 [13] or conversation policies [7], can be specified using languages like ebXML Business Process Specification Schema (BPSS) [11] and Web Services Choreography Description Language (WS-CDL) [13].

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, 12], 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 [6] 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 [10] 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. The contributions of our work are as follows:

– 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.
– Different levels of abstraction are supported, thereby supporting both top-down and bottom-up development and addressing problem (3).
– Our modelling constructs support much of the expressive power of BPSS and BPEL, such that a mapping to the target technologies can be defined. Although a detailed mapping is subject of further work, a sketch of the mapping to the target technologies is presented

The levels of abstraction are introduced in Section 2. The used UML diagrams and specific semantics are elaborated in sections 3–5. A mapping of UML to the target technologies is sketched in Section 6. A brief comparison of our approach to related work is given in Section 7. For the reader not mature in the used technologies, the appendix provides a brief overview of the major concepts of BPEL, BPSS, and UML 2.

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.

Our proposed modelling technique supports modelling of collaborations at three different levels of abstraction, as shown in Fig. 1. 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* 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* 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 *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 *activity model*, a UML activity diagram, defines transactional processes, which refine collaborations as defined in the collaboration level.

The *interaction level* specifies the messages actually exchanged among participants, thus being at a low level of abstraction. Again two kinds of models are proposed. The *message content model* is a class diagram defining the content of messages. 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 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

All of the modelling concepts of UML's collaboration diagrams are used in the collaboration model, with specific interpretation and restrictions as follows.

A *collaboration* models the overall situation of collaborating participants at an abstract level. A collaboration can be *used* within other collaborations as a sub-collaboration, with the roles of the sub-collaboration bound to roles of the using collaboration. A behavior specification can be attached, either as an activity model or an interaction model. In case collaborations are defined solely by sub-collaborations, the behavior specification may be omitted, meaning that there are no behavioral dependencies among the sub-collaborations (except of any pre- and post-conditions defined by the sub-collaborations' behaviors). In that case, the collaboration acts as an abstract grouping mechanism which can be used to capture, e.g., business areas.

A *role* represents a collaboration participant. An optional read-only constraint means that a particular role must be fulfilled by the very same participant throughout the whole collaboration, i.e., it must not change dynamically,

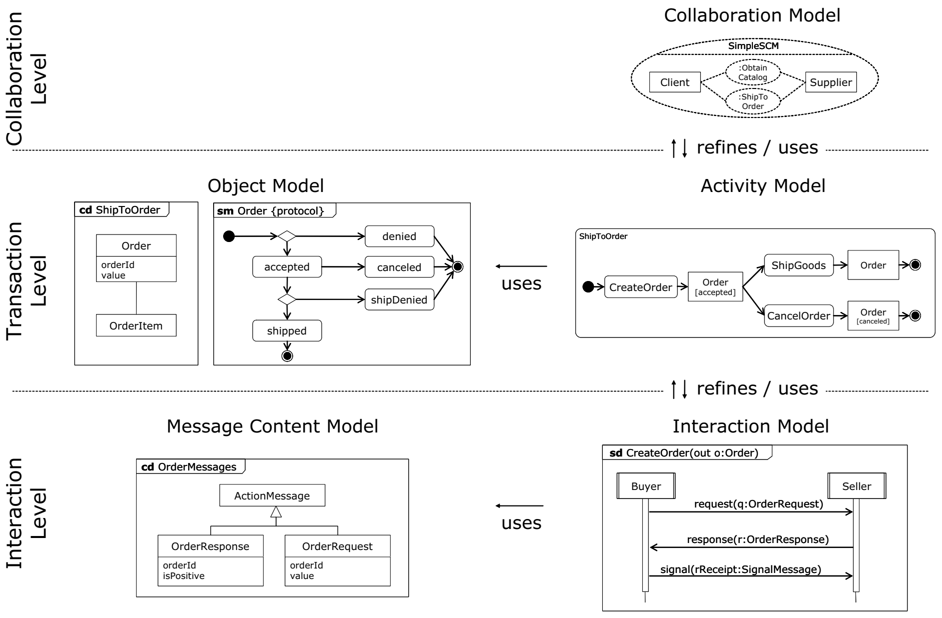

Fig. 1. Abstraction levels and kinds of models

although the participant may not be known at the very beginning of the collaboration.

A *connector* between a set of roles means that these roles communicate with each other directly.

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 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 3.

4 Transaction Level

4.1 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

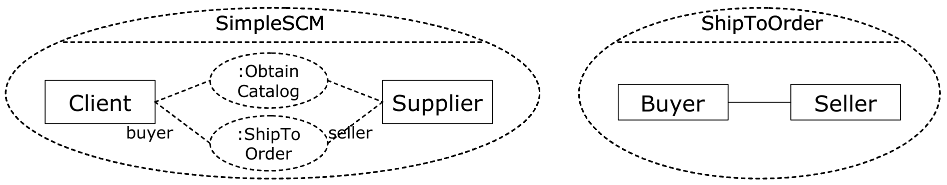

Fig. 2. A simple supply chain collaboration (left) and the sub-collaboration "ShipToOrder" in detail (right)

among participants, whereas objects specify requirements on the participants' data resources. In many cases, messages represent updates to the objects.

For the purpose of collaboration protocol modelling, only those attributes and states are necessary which are needed for defining the coordination logic, 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. Only attributes and associations are used, no operations. A *protocol state machine* is used to specify the states and permissible state transition of such an interface. The states will be used as pre- and post-conditions of transaction specifications (see below), and the transitions and any transition conditions will be 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 2. Fig. 3 (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. 3 (right). Note that transition events are modelled only informally, transition conditions are modelled not at all.

4.2 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.

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

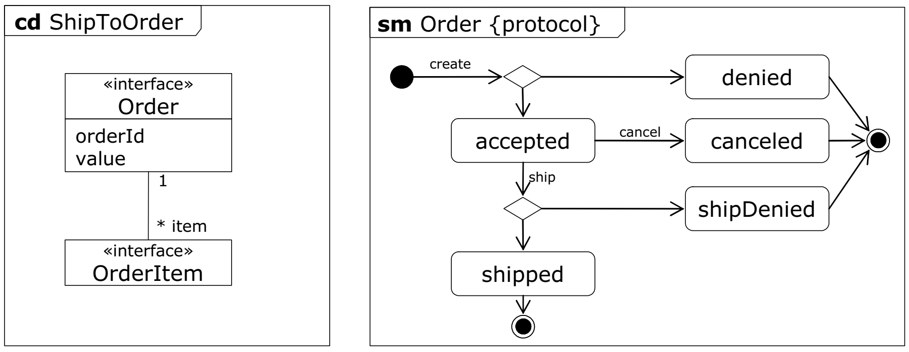

Fig. 3. A data structure used in the collaboration (left) and its permissible behavior (right)

behave such that the resulting behavior of the overall collaboration corresponds to the defined collaboration activity.

An *activity* models the behavior of a collaboration or the behavior of a composite transaction in terms of a process of actions among the participants (cf. Fig. 4). Its input and output parameters define pre- and post-conditions in terms of collaborative objects (with state constraints) consumed and produced by the activity, respectively. Additional pre- and post-conditions on object values can be defined. Top-level activities, i.e., activities used to define the behavior of a collaboration, must not have parameters.

An activity *partition* represents a collaboration participant, which can be associated with actions and object nodes. Since modelling concepts are associated with more than one participant, we cannot use the traditional notation of swimlanes; we use the textual notation of UML2 (cf. "(Buyer, Seller)" in Fig. 4).

An *action* represents a transaction between two or more participants (e.g., "CreateOrder" in Fig. 4). Therefore, it must be associated with at least two partitions. Only behavior invocation actions are allowed, and invoked behavior has to be specified either in terms of another collaboration activity model which thus defines a complex sub-transaction, or in terms of an interaction model defining a simple sub-transaction. The pins of an action must correspond to the parameters of the invoked behavior. Furthermore, the participants involved in the invoked behavior must be the same as the ones the action is associated with.

Object nodes, namely *pin*, *centralBuffer*, and *parameter* represent pre- and post-conditions of actions and activities (e.g. "Order" in Fig. 4). They must be typed by one of the classes or interfaces of the object model. Like actions, object nodes must be assigned to at least two participants, meaning that the pre- and post-conditions apply to those participants. Pins in particular must be associated to a subset of the participants that the pin's action is associated with; parameters similarly. If a parameter is of direction *inout*, the state constraining the outgoing object node must be reachable from the state constraining the incoming object node, i.e., integrity of the state machines must be preserved.

A *key constraint* is a specific kind of 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. A key constraint defined for one object node implies the same constraint for all object nodes reachable by object flow, i.e., through data flow edges or through actions with inout parameters. Since key constraints are not natively supported by UML, the proprietary notation "{key: field_1, field_2, ...}" is introduced.

All of UML's *control nodes* are applicable. Regarding the join node, we introduce an extension to support specifying how objects (of potentially different types) that flow from the multiple input flows are joined into a single object flowing out of the join. This extension is necessary since the work-around of using an explicit action to do the transformation conflicts with the semantics of an action in our approach. Therefore, our proposed solution is to add a transformation behavior on the join node, with one input parameter for each incoming flow, and one output parameter delivering the joined object. A similar extension could be designed for the fork node, although in that case the transformation capability of object flows can be used for a work-around. Note that these extensions are not specific to our approach but generally applicable.

The basic *control flow* edges are applicable, however certain restrictions must be met to produce realizable collaboration protocols. 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 forks, joins, decisions, and merges. We did not yet consider UML's advanced concepts such as interrupts, edge weight, etc.

Also *object flow* edges can only be used with restrictions. First, 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. Second, in case of non-deterministic choice, i.e., when multiple data flows originate from an object node, the one action will be selected which is initiated first. To allow for unambiguous decision detection, at least one participant must be involved in all of the alternative actions, and the actions must be distinguishable by the first message that is exchanged (cf. interaction level). Regarding the advanced features of data flow edges, transformation behavior is required to support chaining of actions with differently structured output and input objects; other features such as multi-cast and edge weight have not been considered.

Example 3. The activity shown in Fig. 4 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.

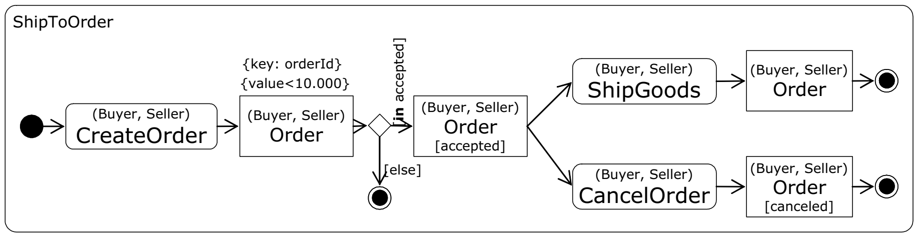

Fig. 4. Activity model of the "ShipToOrder" collaboration

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. 4). Furthermore, if the collaboration is composed of sub-collaborations, the composite collaboration's activity must include (i.e., invoke) its constituent collaborations' activities.

5 Interaction Level

5.1 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. It will be aligned with the object model, by using the same attributes and, if appropriate, by directly including parts of the object model. An abstract message model facilitates re-use of variants of complete message models, e.g., different business document standards could be supported. In that case, the abstract message model must only include a subset of the data supported by the different document standards.

The concepts used in the message model are interfaces and classes. *Interfaces* are used to specify an abstract message model. Interfaces can have attributes, associations, and operations that implement queries based on the interfaces. *Classes* are used to specify complete message models. Again, message classes can have attributes, associations, and query operations. The relationship to an abstract message model is specified by implementation relationships. A class implements the interface's attributes and associations. For specifying XML-related characteristics of message contents, an appropriate UML profile has to be used [2].

Example 4. The example in Fig. 5 shows an abstract message model for the messages in the "ShipToOrder" collaboration. The messages are based on generic ebXML messages "SignalMessage" and "ActionMessage".

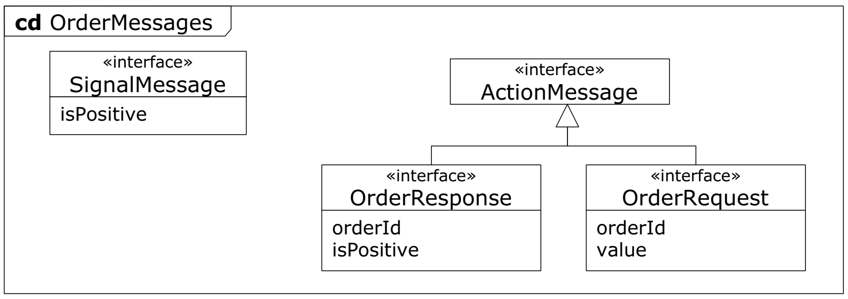

Fig. 5. Abstract messages used by the "CreateOrder" interaction

5.2 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. The tasks performed by the participants are, however, out of scope of a collaboration model. It is only the overall transaction which represents a common/synchronous task.

Modelling concepts are those of UML interaction diagrams, with a few exceptions, as well as some additional constraints.

A *lifeline* represents a participant from the collaboration.

A *message* defines a communication between two participants. Only asynchronous messages must be used, as synchronous messages make a statement about message processing, which by definition is out of scope of a collaboration protocol. Message arguments are limited to be either constants, or parameters of the interaction, or symbolic names representing wildcards. If, within a single interaction, the same symbolic name is used in different messages, it means that the contents is the same. Symbolic names may refer to lifelines, meaning that a reference to the corresponding participant is being communicated.

Regarding interaction *fragments*, i.e., control flow structures, the following constraints must be met in order to obtain a realizable collaboration protocol. Guards on interaction fragments must be completely specified (using OCL), based on interaction parameters and on message arguments that are sent from or received by the participant on which the guard is specified, i.e., the participant that initiates the first message in the interaction fragment. For simplicity, the model is limited to LL(1)-like traces, i.e., for each lifeline each incoming message must unambiguously determine the following control path.

A *data flow constraint* is a UML constraint used to relate the contents of individual messages to each other as well as to the input and output parameters of the interaction. We distinguish three classes of data flow constraints. Deterministic data flow constraints define equality of data items in different messages or parameters. They are required for key constraints (see below) and for specification of post-conditions. Non-deterministic constraints may also use inequations. Conditional constraints use implications. They are required to specify post-conditions in case of alternative control flow paths. 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.

A *participation constraint* is used to interrelate data flow and participants. A participation constraint relates a lifeline with the contents of a message or a parameter, meaning that a reference to the concrete participant playing that role is being communicated. Participation constraints are an extension to UML's capability of using parameters as lifelines, and to using lifelines as symbolic names of message arguments, as defined above.

A *key constraint* specifies, for a message, that some of the message's arguments (or argument's attributes) 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. Note that sender and recipient (lifelines) may also be used as key components. The same set of key attributes typically appears in a series of succeeding messages. Only the first message of such a series must have a key constraint, the applicability to succeeding messages is implied by deterministic data flow constraints. Data flow constraints also specify the flow of key constraints on input and output parameters, as defined in the activity model. If a key constraint is already defined on an input parameter, no additional message key constraint is required for the same set of attributes. Since key constraints are not natively supported by UML, proprietary notation "{key: field_1, field_2, ...}" is introduced.

A *time constraint* must be specified such that there is at least one participant able to observe it. Observable are interaction parameters and messages sent to and received by the participant.

A *post-condition* specifies the outcome of the interaction in terms of constraints on the output parameters. The same rules as to data flow constraints apply. Furthermore, the post-condition constraints must be observable by all participants who are assigned to the respective output parameter. The resulting value of all output parameters must be defined, and in case of alternative control flows, each of the flows must lead to a defined output. In case of alternative output parameter sets, for each set there must be a flow defining it completely.

Concepts of UML related to the specification of individual participants are not used, other than message send and receive. Although it would be possible to use concepts such as participant attributes, sate, and execution occurrences, it is not necessary for collaboration specification and therefore not used to maintain loose coupling.

Example 5. The interaction depicted in Fig. 6 corresponds to a BPSS interaction pattern. It comprises a request and a response message, and accompanying acknowledgement signal messages. Positive acknowledgement messages are a prerequisite 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".

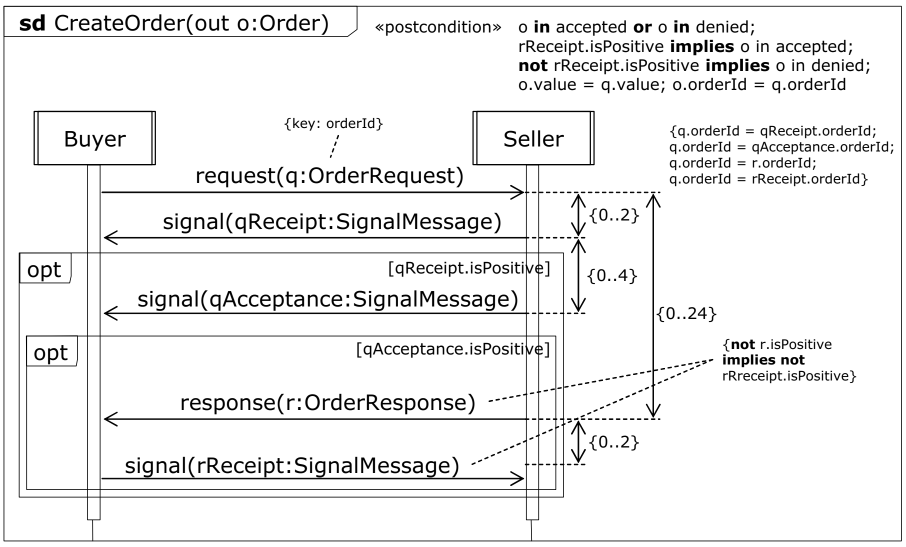
sd CreateOrder(out o:Order)
«postcondition» o in accepted or o in denied;
rReceipt.isPositive implies o in accepted;
not rReceipt.isPositive implies o in denied;
o.value = q.value; o.orderId = q.orderId
Buyer
{key: orderId}
Seller
{q.orderId = qReceipt.orderId;
q.orderId = qAcceptance.orderId;
q.orderId = 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. 6. Interaction refining the "CreateOrder" action

Presumably, transactions are often defined based on generic interaction patterns such as the one used in Example 5. Such generic interaction patterns can be supported by means of interaction templates having template parameters for specification of participants, message types, and timing constraints.

6 Mapping to Target Technologies

In this section we want to clarify the relationships between the proposed modelling concepts and concepts of the target technologies by sketching a possible mapping between the two. The mappings, as shown in Table 1, demonstrate feasibility and limitations of such an attempt, but are not elaborated in full detail.

Designing the mapping to BPSS is relatively straight forward, although not all concepts can be mapped (cf. Table 1). The mapping to BPEL is much more difficult, since it actually involves two steps. First, the collaboration protocol has (at least conceptually) to be transformed into a set of interfaces for the participating web services, and then each of these interfaces has to be transformed to BPEL. In particular, this means that for each partition of a top-level activity a separate BPEL specification has to be produced, which covers only a part of the activity; similarly for interactions. Furthermore, detection and to some extent also handling of failures depends on implementation decisions, therefore a mapping can only be defined by assuming some generic implementation.

When roughly comparing the expressiveness of our approach to BPSS, BPSS has several limitations, e.g., the focus on 2 participants, predefined interaction patterns, no key constraints, only limited constraints on message data. On the contrary, our approach lacks specification of atomicity, failure handling, and the "onInitiation" control flow dependency. In comparison to BPEL, BPEL lacks a support of re-usable behavior specification, and has restrictions on message properties which limit expressiveness of key constraints. Also, non-deterministic choice with different initiators is cumbersome to express in BPEL. Our approach, on the other hand, lacks support for exception handling, compensation, and event handling.

7 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 [10] 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.

M	UML	BPSS	BPEL
c	collaboration	-	binary collaboration protocol, or multiparty cp., or business transaction
	role	process	party
	connector	partner link type and partner (limited to binary connectors)	-
	collaboration use	- (see nesting of activities)	composition of business activities (limited to 2 participants)
o	interface	XSD type	-
	protocol sm.	-	-
a	activity	-	binary cp. (2 participants only) or multi-party cp. (restrictions on control flow and data flow)
	partition	abstract process, or scope if a nested activity	authorized role
	action	scope including the mapping of the invoked activity/interaction	binary cp. or business transaction (limited to 2 participants)
	object node	variable	-
	key constr.	property and correlation set	-
	control node	structured activity (limited to regularly structured control flow)	equivalent control nodes, except of decision which is mapped to transition constraints
	control flow	structured activity	transition
	data flow	variable assignment	-
	non-determ. choice	a switch on opaque data, or a pick, depending on who is deciding	transitions without mutually exclusive guards
m	interface	message properties (limited to simple data types)	-
	class	XSD type	XSD type
i	interaction	-	business transaction (limited to predefined interaction patterns)
	lifeline	abstract process or nested scope	predefined
	message	invoke or receive+reply, and a variable for holding the message	predefined signal or business message
	fragment	structured activity	-
	data flow constr.	variable assignment (limited to equality constr.) and a correlation if related to a key constraint	-
	participation constr.	assignment	not allowed
	key constr.	correlation set and related property aliases (limited to simple data types and equality constraints)	-
	time constr.	-	predefined timeToPerform, etc.
	pre-/post-conditions	-	pre-/post-conditions (informal)

Table 1. Mapping of UML concepts in the different models to BPSS and BPEL

UMM [9] 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.

There exists also a mapping from a subset of UMM to BPEL [8]. 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 [7, 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.

8 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.

– 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 more detail, to enable automatic code generation. Furthermore, support for WS-CDL would be logical but has not yet been included.
– 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, and Siebel. Business Process Execution Language for Web Services, Version 1.1. `http://ifr.sap.com/bpel4ws/BPELV1-1May52003Final.pdf`, May 2003.
2. M. Bernauer, G. Kappel, and G. Kramler. Representing XML Schema in UML – A Comparison of Approaches. In *Proceedings of the 4th International Conference on Web Engineering (ICWE2004)*, 2004.
3. M. Bernauer, G. Kappel, G. Kramler, and W. Retschitzegger. Comparing WSDL-based and ebXML-based Approaches for B2B Protocol Specification. In *Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC 2003)*, 2003.
4. T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *WWW*, pages 403–410, 2003.
5. DERI. Web Service Modeling Ontology - Standard, WSMO Working Draft. `http://www.wsmo.org/2004/d2/v02/`, March 2004.
6. T. Gardner. 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, 2004.
7. J.E. Hanson, P. Nandi, and S. Kumaran. Conversation support for Business Process Integration. In *Proceedings 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC-2002)*, pages pp. 65–74.
8. B. Hofreiter and C. Huemer. Transforming umm business collaboration models to bpel. In *Proc. of the OTM Workshop on Modeling Inter-Organizational Systems (MIOS 2004)*, oct 2004.
9. B. Hofreiter, C. Huemer, and K.-D. Naujok. Un/cefact's business collaboration framework - motivation and basic concepts. In *Proc. of the Multi-Konferenz Wirtschaftsinformatik (MKWI 2004)*, mar 2004.
10. H. Kim. Conceptual Modeling and Specification Generation for B2B Business Processes based on ebXML. volume 31 of *SIGMOD Record*.
11. UN/CEFACT and OASIS. ebXML Business Process Specification Schema, Version 1.01. `http://www.ebxml.org/specs/ebBPSS.pdf`, May 2001.
12. W3C. Web Service Choreography Interface (WSCI) 1.0, W3C Note. `http://www.w3.org/TR/wsci`, August 2002.
13. W3C. Web Services Choreography Description Language Version 1.0, W3C Working Draft. `http://www.w3.org/TR/ws-cdl-10/`, October 2004.

9 Appendix - Review of Major Technology Concepts

As a prerequisite to the main part, we review the major specification concepts supported by the target technologies, as they define the requirements on the expressiveness of our platform independent modelling technique. Furthermore, we give a brief overview of the relevant concepts of UML 2 used in our approach.

We have chosen BPEL and BPSS as potential target technologies, for the following reasons. BPSS because it is a collaboration protocol language and it provides interesting higher-level concepts, i.e., transactions. And BPEL because it is well accepted and it provides powerful specification concepts for web service interfaces and implementations. Since BPEL supports interface and implementation rather than collaboration protocols, it is only an indirect target language. Nevertheless, its specification concepts should be supported also at the collaboration protocol level.

9.1 BPSS Concepts

BPSS, as a collaboration protocol specification language, supports the specification of emergent behavior, i.e., behavior that emerges from a group of collaborating web services. Behavior as specified by BPSS is not intended to be used as instructions for some execution engine, but rather defines constraints on the observable behavior of the participants in the collaboration. If all of the participants follow the constraints imposed on them according to their role in the collaboration protocol, the collaboration will be able to achieve the common goal.

The distinguishing feature of BPSS is that it is based on the concept of *business transaction*, which is an interaction between two parties leading to either a successful outcome or having no effect at all (atomicity property). A business transaction is defined by one out of a set of predefined interaction patterns. These interaction patterns are an extension of the request/reponse pattern, and can be configured in terms of the types of documents exchanged, timing and security constraints required, etc. Specification of failures is supported in two aspects. First, application-level failures can be defined in terms of negative responses. Second, system-level failures such as connection failure or protocol failure are supported by predefined exception states. In both cases it is assumed that the business transaction had no business effect, i.e., was rolled back.

A *binary collaboration protocol* specifies the collaboration between two parties, based on business transactions and nested binary collaboration protocols (these two are called business activities). Like a business transaction, a binary collaboration protocol exhibits the atomicity property. A binary collaboration protocol is defined by its constituent collaboration activities and the control flow dependencies between them. No data flow is supported at the level of binary collaboration protocols; however, the guard condition of a transition can be defined based on the documents exchanged in the preceding business transaction.

Failures in binary collaboration protocols can be specified in that it can lead to different result states, indicating either success or failure, and, in case of failure it is assumed that the overall binary collaboration protocol has no business effect, i.e., was rolled back. Failure handling is specified by means of specific transition conditions support conditional branches, depending on whether a preceding collaboration activity succeeded or failed. Using this simple mechanism, failure handling to the effect of compensating transactions can be designed, although no specific compensation concept is available.

A *multi-party collaboration protocol* defines a set of interrelated binary collaboration protocols. A Multi-party collaboration protocol supports no transactional properties and only limited means for behavior specification.

9.2 BPEL Concepts

BPEL, in combination with WSDL, supports specification of both implementation and interfaces of an individual web service.

BPEL is based on WSDL *port types* which define the operations provided by a web service. So-called *service link types* are introduced to define a mutual usage relationship between two port types.

A *process* specifies the executable behavior of a web service. A process may communicate with multiple *partner* web services, the relationship to each partner being defined by a service link type. A process is defined in a block-structured manner, with basic statements being variable assignment and the communication primitives receive, reply, and invoke. Available control flow structures are sequence, flow (parallelism), choice, and pick (choice based on event selection). A *scope* is a nested part of a process which supports exception handlers, a compensation handler, and event handlers. Using these mechanisms, it is possible to design user-defined failure handling. The coordination between collaborating web services in the presence of failures, however, is out of scope of BPEL.

Specific concepts are provided with regard to data handling. First, so-called *properties* provide abstract accessors to message contents, which can be refined for particular message types in terms of *property aliases*. Second, based on properties, *correlation sets* can be defined, which facilitate unambiguous association of a given message to its corresponding process instance. Finally, *partner references* are specific data types enabling run-time selection and communication of partners.

An *abstract process* is a view on an executable process, which specifies an interface of a web service. An abstract process is an incomplete behavior specification in that it is allowed to omit any parts of internal (non-observable) behavior. Specifically concerning data flow, so-called opaque data is supported, i.e., data which appears in the process but its computation is not defined.

9.3 UML 2 Concepts

UML 2 provides a set of specification concepts covering structural and behavioral aspects. We briefly review those relevant to our work. Note that we take advantage of the enhanced concepts of UML 2 opposite to UML 1, particularly collaboration, protocol state machine, activity, and interaction.

A *class* defines individual objects in terms of their structure, functionality, and associations to other objects. Furthermore, a behavior specification such as a state machine can be attached to a class. An *interface* is similar in concept, with the difference that it does not provide an instantiable specification but rather a set of constraints on an object. Behavioral constraints can also be attached in terms of a protocol state machine.

A *collaboration* defines the structural aspect of a set of collaborating objects in terms of the participating roles and their communication relationships. Like an interface, a collaboration is not directly instantiable but provides a specification for a set of objects that are intended to participate. Any kind of behavior specification such as an activity or an interaction can be attached. A behavior specification used in this way defines *emergent behavior* of all the participants rather than behavior executed by an individual object.

A *state machine* defines the executable behavior of an object, in terms of the object's state and permissible state transitions. A *protocol state machine* is a state machine restricted to defining *observable behavior*. In particular, transitions in a protocol state machine can only be defined in terms of pre- and post-conditions, rather than by actions performed during the transition.

An *activity* is a petri-net based behavior specification, comprising actions with input and output, and control flow and data flow between those actions. A set of predefined actions is available, including one to invoke other behavior specifications. Data flow tokens may be typed by a class or interface.

An *interaction* specifies the observable behavior of a set of collaborating objects, in terms of the participants and a set of allowed and forbidden traces of message exchanges among these participants. A particular focus is on timing of messages. An interaction cannot specify executable behavior; nevertheless, control flow structures such as loops and alternatives are available.