The Business Choreography Language (BCL) - a Domain-Specific Language for Global Choreographies

Thomas Motal, Marco Zapletal, Hannes Werthner
Vienna University of Technology
Favoritenstrasse 9-11/188
A-1040 Vienna, Austria
{lastname}@ec.tuwien.ac.at

Abstract

UN/CEFACT's Modeling Methodology (UMM) is a modeling approach for describing the choreography of B2B processes. UMM is developed by the United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT) and currently defined as a UML profile. Thereby, it constrains the UML for the specific needs of B2B. As we learned, using UML as the underlying notation for UMM results in several shortcomings. Furthermore, some workarounds are required to fit the concepts of UMM to the UML meta model. Thus, in this paper we examine an alternative notation for UMM following the concepts of a domain-specific language (DSL). The contribution of this paper is twofold: (i) we identify general concepts for modeling global choreographies by taking UMM as a starting point. (ii) We introduce the Business Choreography Language (BCL), a domain-specific language designed to efficiently support the prior identified concepts. The concepts of the BCL are exemplified by an implementation using the Microsoft DSL Tools for Visual Studio. In fact, the BCL is an approach tailored to support the specific needs of global B2B choreographies.

1 Introduction

In the recent past, the notion of choreography gained a lot of attraction in the fields of service-orientation as well as business process modeling. In [1], Peltz introduced one of the first definitions for the terms orchestration and choreography. We would like to refine the definitions for these terms as follows: An orchestration describes a process that is executed internal to a company. A local choreography is a projection on an orchestration - it contains only those tasks of a process that are visible to the outside world. These tasks as well as their order describe the required flow of message exchanges in order to interact with the process. Thus, a local choreography has always a partner-specific viewpoint. A global choreography describes a process from a neutral viewpoint by capturing the observable behavior between two or more partners – i.e., between their complementary local choreographies.

For the purpose of modeling global choreographies languages such as BPMN [2], iBPMN [3], Let's Dance [4], WS-CDL [5], ebXML BPSS [6], and BPEL4Chor [7] have been proposed. A further promising approach for capturing global choreographies between enterprises is UN/CEFACT Modeling Methodology (UMM). UMM is developed and standardized by the United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT), known for its work in the area of electronic data interchange (EDI). In order to overcome the limitations of traditional data-centric EDI approaches, UN/CEFACT developed a modeling language for B2B interactions – so-called business collaborations. Thereby, UMM concentrates on the interactions between enterprises and describes them from a neutral perspective – it does not consider on the business processes internal to a company.

UN/CEFACT's UMM is built upon the UML. The most recent version [8] – which we co-authored – is specified as a UML profile for UML 2.0 [9]. A UML profile specifies a set of stereotypes, tagged values and constraints for customizing UML [10]. Thus, the general-purpose language UML is customized for the specific purpose of modeling inter-organizational systems. Thereby, UMM puts UML in a very strict corset. The definition of UMM on top of UML imposes some shortcomings and requires some workarounds. As a consequence, many stakeholders of UMM have assessed it as being too complex. Furthermore, the readability and graphical visualization of UMM models are criticized. Another negative argument is that UMM models result in an overwhelming amount of modeling artifacts.

In order to overcome these shortcomings, this paper pro-

poses to define the UMM concepts by means of a domain-specific language (DSL). The resulting language is called Business Choreography Language (BCL). A DSL is a high-level software implementation language that supports concepts and abstractions that are related to a particular (application) domain [11]. The concept of domain-specific languages is pretty mature. Parnas introduced an early view on DSLs in the late 70s by the means of program families [12]. In 1985, Bentley concluded in [13] that the main task of programmers is to develop small languages to solve problems of a particular kind. Thus, Bentley had an early idea of what a DSL should be. In general, a DSL model provides its own notation, which follows a set of well-defined rules [14]. It utilizes the concepts and names of the domain under consideration.

The remainder of this paper is structured as follows: in section 2 we introduce the UML profile of UMM by a simple example. Section 3 points out required concepts for the modeling of global choreographies. Then, in section 4 we introduce the Business Choreography Language (BCL) that reflects the concepts captured before. Finally, section 5 provides an evaluation of our approach by comparing the BCL with the original UMM and outlines future work.

2 UMM by Example

When the development of UMM was started by UN/CEFACT in 1998, tool support was identified as one of the critical issues in order to guarantee user acceptance of the UMM. Due to the growing tool support of the Unified Modeling Language (UML), UN/CEFACT opted for UML as the underlying modeling notation. At this time, UMM specified its own conceptual meta model and provided guidelines on creating compliant artifacts using the UML. In late 2004 it was decided to define UMM 1.0 as a formal UML profile [15], i.e. a set of stereotypes, tagged values and constraints - in order to customize the UML meta model to the special purpose of modeling global B2B choreographies. At this time the UML version of choice by UN/CEFACT was UML 1.4 [16]. We guided the editing of the UMM foundation module 1.0, which became a UN/CEFACT standard in October 2006. It should be noted that the UML 1.4 meta model required several workarounds to represent the UMM concepts, which resulted in several shortcomings. The identification of these shortcomings led to the development of UMM 2.0 [8], which is defined as a UML 2.0 profile [9] .

The UMM follows a well-defined development process producing a set of well-defined artifacts. The development process runs through three major phases, which correspond to the three top level packages of UMM: the *business requirements view*, the *business choreography view*, and the *business information view*. The interested reader is referred to the UMM paper in [17] as well as to the specification in [8]. In this paper we do not elicit requirements using the *business requirements view* of UMM, but limit ourselves to the relevant concept for building a domain-specific language for global choreographies. In the following, we focus on artifacts of two sub-views of the *business choreography view*: the *business transaction view* and the *business collaboration view*. The *business transaction view* models the basic building blocks of a choreography which correspond to a single *business document* exchange and returning an optional *business document* as a response. The *business collaboration view* models a global choreography built by these basic building blocks.

In order to exemplify the described concepts we will use a simple order from quote example. The buyer has to send a quote to the seller. Depending, whether the quote will be accepted or not, the buyer will be able to order products. The order will be confirmed by an order response. Our example *business collaboration* takes place between the buyer and the seller, and consists of two *business transactions*: request for quote and place order. In the following, we outline how to model this example B2B scenario using the UML profile of UMM 2.0.

2.1 Business Transaction View

The goal of a *business transaction* is to synchronize the state between two partner's information systems. The synchronization takes place by exchanging business information between exactly two parties. Depending on the business intent, it is either a uni-directional exchange (e.g., a notification) or a bi-directional exchange including a response (e.g., a request/response interaction). As further explained in section 3, a *business transaction* always follows the same pattern.

The requirements of a *business transaction* are described in a *business transaction use case*. Our example involves two *business transaction use cases*: request for quote and place order. According to figure 1, each *business transaction use case* and the participating actors are placed in their own *business transaction view*. Figure 2 depicts the *business transaction use case* request for quote which involves a quote requestor and quote responder.

The requirements of a *business transaction* are further elaborated using the concept of an activity diagram. For each *business transaction use case* an activity diagram is created and placed as a child underneath the respective use case, e.g., in figure 1 the *business transaction use case* request for quote (A) is refined using the activity diagram (B).

The basic building blocks of a *business transaction* are activity partitions, which are used to denote the *authorized roles*, participating in the transaction. Furthermore, a *business transaction* contains exactly two actions - a *requesting*

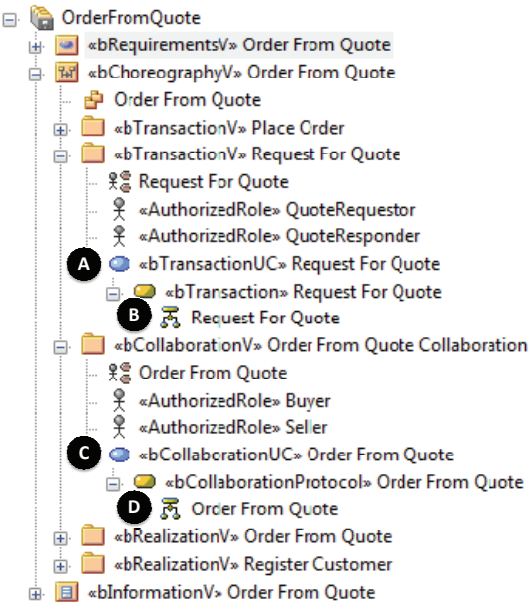

Figure 1: Package Structure of the Order From Quote Example

Figure 2: Business Transaction Use Case

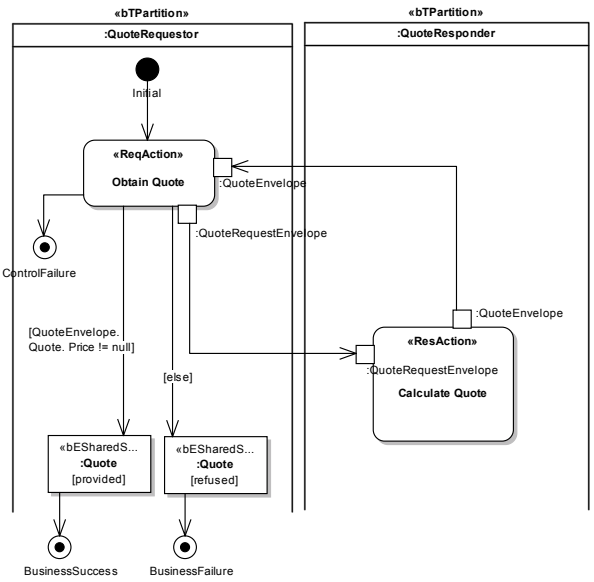

Figure 3: Business Transaction: Request for Quote

action and a *responding action* - one on each business partner's side. Between the different actions the business information exchange is denoted using the concepts of object flows and action pins. There is always exactly one object flow from the *requesting action* to the *responding action*. In a one-way *business transaction* there is no flow in the reverse direction. In case of a two-way *business transaction* there are one or more object flows in the reverse direction. In case of two or more object flows they are considered as alternatives. The type of the action pins in the *business transaction* is set using *business documents* from the *business information view*, which is not elaborated in this paper.

Figure 3 shows the *business transaction* request for quote. On the left hand side the *business transaction partition (bTPartition)* of the *requesting role* is shown and on the right hand side the one of the responding role. The type of a *business transaction partition* is determined by the *authorized roles* participating in the *business transaction use case*, which the *business transaction* refines. In figure 3 the type of the *requesting partition* is set by the *authorized role* quote requestor and the type of the *responding partition* is set by the *authorized role* quote responder.

The *requesting partition* contains a so called *requesting action (ReqAction)* and the *responding partition* a *responding action (ResAction)*. In the example shown in figure 3 the quote requestor starts the *business transaction* by sending a quote request envelope to the quote responder.

After the quote responder has processed the request from the quote requestor he replies with a quote envelope. Within the quote envelope, products are quoted or not. In the former case, the *business transaction* results in a success (i.e., both share the state that a quote is provided). In the latter case, the synchronized state between the two partner's system is that the quote was refused - reflecting that the *business transaction* resulted in a business failure. In case of a technical exception, the *business transaction* results in a control failure as shown on the left hand side of figure 3.

At the lower side of figure 4 the tagged values containing the different business signal information of the *requesting action* and the *responding action* are shown e.g. *time to acknowledge receipt* indicates the maximum time within the responding party has to confirm a successful/unsuccessful syntax, grammar, and sequence validation. Further tagged values are: *is authorization required*, *is non-repudiation required*, *time to perform*, *time to acknowledge receipt*, *time to acknowledge acceptance*, *is non-repudiation of receipt required* and *retry count*. These tagged values are considered as self-explanatory and are explained in detail in the UMM 2.0 specification [8].

As shown in figure 1, the order from quote example consists of two *business transactions*: request for quote and place order. Since *business transactions* follow the same pattern, we refrain from detailing the flow of place order. Instead, we show the resulting *business transaction* in figure 4. One should note that place order contains two alternative response documents - an order acceptance envelope and

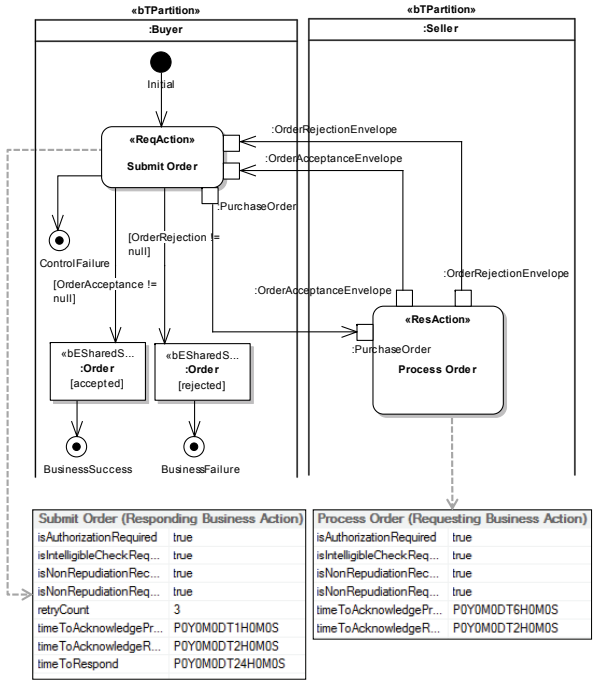

Figure 4: Business Transaction: Place Order

an order rejection envelope. Consequently, the responding *business document* affects the outcome of the overall *business transaction* as depicted in figure 4.

2.2 Business Collaboration View

After the identification of the different *business transactions* the modeler creates *business collaborations*. A *business collaboration* describes the control flow between the different *business transactions* and *business collaborations*. Thus, a *business collaborations* can also be nested recursively.

Each *business collaboration view* contains exactly one *business collaboration use case* and the *authorized roles* participating in the use case. By definition, a *business collaboration* consists of different *business transactions* and/or *business collaborations*. Included *business transactions/-collaborations* are denoted using the concept of *include dependencies*. Each included *business transaction* is defined in its own *business transaction view* and each included *business collaboration* is defined in its own *business collaboration view*.

As shown in figure 5 the *business collaboration use case* order from quote includes two *business transactions*, namely request for quote and place order. Similar to the concept of a *business transaction use case* a *business collaboration use case* is further elaborated using the concept of a so called *business collaboration protocol*. For each *business collaboration use case* a *business collaboration*

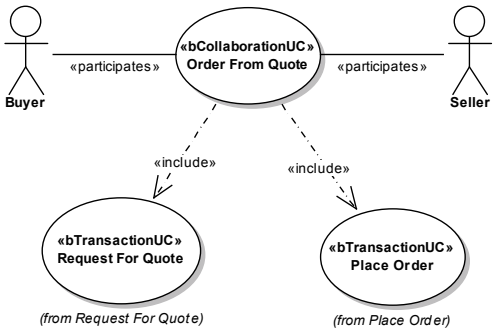

Figure 5: Business Collaboration/Transaction Use Case

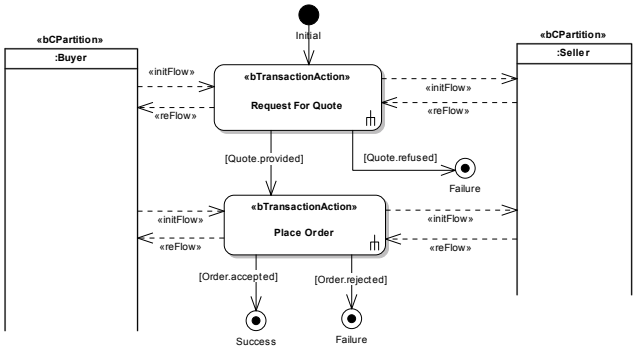

Figure 6: Business Collaboration Protocol Order From Quote

protocol is created and placed as a child under the respective use case e.g. in figure 1 the *business collaboration use case* order from quote (C) is refined using the *business collaboration protocol* (D). Consequently a *business collaboration use case* is always the parent of exactly one *business collaboration protocol*.

The main aim of a *business collaboration protocol* is to describe a *business collaboration* on a formal basis. Thereby, a *business collaboration protocol* is built using *business transaction actions* and *business collaboration actions*. Thus, each action calls the respective transaction or collaboration. In order to depict the *authorized roles* participating in a *business collaboration*, a *business collaboration protocol* uses the concept of partitions. For each *authorized role* exactly one partition is created.

In order to denote the execution order of different *business transaction actions* and *business collaboration actions* the concept of *initFlows* and *reFlows* is used. They denote, which role of the *business collaboration* initiates and responds in an underlying *business transaction*. Thereby, an *initFlow* connects a partition representing an *authorized role* with a *business transaction action/business collaboration action*. The same applies to *reFlows*.

The *business collaboration protocol* in figure 6 defines the exact choreography of the order from quote collaboration. Using the concept of two *business collaboration*

partitions (bCPartition) the two *authorized roles* buyer and seller participating in the *business collaboration* are shown. The *business collaboration* order from quote starts with the *business transaction* request for quote. The *initFlow* dependency between the buyer and the *business transaction action* request for quote in figure 6 indicates, that the buyer initiates the *business transaction*. Consequently, the buyer of the *business collaboration protocol* plays the quote requestor in the *business transaction* request for quote. The other *initFlow* leads from the *business transaction action* request for quote to the partition of the seller. This declares that he plays the quote responder in the request for quote *business transaction*. The *reFlow* dependencies are not required for the unambiguous mapping between collaboration roles and transaction roles. However, they are used to visualize already on the *business collaboration protocol* that an underlying *business transaction* is a two-way transaction (which was a request by stakeholders). In regard to the *business transaction action* place order, the mapping of roles follows the similar procedure.

If the *business transaction* request for quote fails because the seller has refused the request, the *business collaboration* order from quote also fails. In figure 3 this is indicated by the control flow with the guard condition quote.refused leading from the *business transaction action* to the final state failure. Please note, that the guard conditions of the control flows directly match the resulting states denoted in the underlying *business transaction* (see figure 4). In case the *business transaction* request for quote was successful, the guard condition quote.provided evaluates true and the *business transaction* place order starts.

3 Concepts

In the previous section, we learned about the UMM by means of the order from quote example. The notation that is used in the example is the UML profile for UMM. It is an evident goal that a domain-specific language (DSL) for UMM must at least be as expressive as the UML profile, including the UMM concepts. Thus, we must first identify the concepts of UMM for modeling global choreographies independent of the underlying syntax. The following listings sketches these concepts (denoted as *C1* to *C8*) detached from the UML profile introduced in section 2.

- A *business collaboration (C1)*: A *business collaboration* is a business process that is jointly conducted by two or more participants. A *business collaboration* is a complex long-running transactions and is composed of one to many *business transactions*. In fact, the *business collaboration* governs the control flow between *business transactions*. Thereby, *AND*, *OR*, and *XOR split/join nodes* may be used in order to realize the five basic workflow control-flow patterns [18].

- A *business transaction (C2)*: A transaction is the basic building block in UMM for defining global choreographies. It synchronizes the states between two partner's information systems realized by an uni- or bi-directional information exchange. A *business transaction* follows always the same pattern. The *business transaction pattern* defines the type of a legally binding interaction between two decision making applications as defined in Open-edi [19].

- *Business transaction patterns (C3)*: UMM distinguishes between two uni-directional (notification, information distribution) and four bi-directional transaction patterns (*request/response, query/response, request/confirm, commercial transaction*). In case of the latter, multiple alternative *responding business document* types may be specified. The *business transaction patterns* differ in their settings for the *quality of service parameters*.

- *Quality of service parameters (QoS) (C4)*: *Business transactions* hold a set of *QoS parameters* as introduced in section 2.1. These *QoS parameters* define security and time-out constraints as well as the exchange of business signals upon the receipt of *business documents*.

- *Business documents (C5)* convey business information in a *business transaction*. The initiator of the *business transaction* communicates a *requesting business document (C5a)* and the requestor replies with a *responding business document (C5b)*.

- *Shared states (C6)*: The notion of a state is an inevitable concept in modeling global choreographies. In order to jointly conduct a *business collaboration*, the states of the participating information systems must be in sync. State changes in one system must be communicated to the information systems of other participants. The synchronization of states requires the communication of the corresponding information. As introduced before, we build upon the concept of *business transactions* (C2) to convey information in *business collaborations*. In case of a two-way *business transaction*, the *responding business document* determines the outcome of the *business transaction* - i.e., the shared state. Considering figure 4, the communication of an order acceptance envelope lets the *business transaction* result in the shared state that the order is accepted. The shared state determines the control flow in the further *business collaboration* - e.g., an order rejection envelope leads to a rejected order, which results in an overall failure of the *business collaboration* as shown in figure 6.

- *Re-use of business transactions (C7)*: One of the key goals of UMM is to foster re-use. For this purpose, *business transactions* may be re-used across different *business collaborations*. In UMM 2.0's UML profile, the same *business transaction* may be called by different *business transaction actions*, since a *business transaction action* extends a UML call behavior action.

- *Role mapping (C8)*: The concept of roles in modeling *business collaborations/business transactions* allows to abstract from concrete business parties. In order to realize the re-use of *business transactions* across different *business collaborations*, it is required to specify which role of a *business collaboration* takes up which role in a *business transaction*. The same applies when re-using *business collaborations* by hierarchically nesting them.

Furthermore, we have identified some concepts that are not yet part of UMM, but are considered to be useful for modeling global choreographies. These concepts are related to the notion of events as found in the Business Process Modeling Notation (BPMN) [2]. In BPMN, an event happens during the course of a business process and affects its further flow. There are several types of events defined in BPMN - in the following we describe those of them which we use in our DSL:

- *Timer events (C9):* The concept of a *timer event* is used to represent delay mechanisms in a *business collaboration*. After a certain timing condition evaluates to true - which can be either relative or absolute - the event is triggered and the process flow continues. Please note that the semantics of a *timer event* may already be realized in UMM 2.0, since UML activity diagrams permit time-based guard conditions on control flows. However, since a DSL should describe a problem domain as expressive as possible, we decided to visualize *timer events* explicitly.

- *Compensations (C10):* An exception might happen in a *business transaction BT2*, which is part of a *business collaboration BC*. In such a case, an already successfully executed *business transaction BT1*, which precedes *BT2* in *BC*, may require a rollback. Consequently, a compensating *business transaction BT3* needs to be performed that undoes the effects of *BT1*.

- *Event-based XORs (C11):* The semantics of an *event-based XOR* split differs to a regular XOR split as follows: While the latter decides upon which subsequent branch is activated based on available data, the *event-based XOR* split bases its decision on external events. In other words, multiple alternative events may happen at a certain branching point in a process. Whatever event occurs first (e.g., the receipt of a message), the respective branch is activated. This construct realizes the behavior of a *deferred choice* as specified in [18]. Please note that the initiation of a *business transaction* (i.e., sending/receiving the *requesting business document*) is also considered as an event in that sense. Thus, two to many *business transaction* may follow an *event-based XOR* split.

In the subsequent section we introduce the Business Choreography Language (BCL), a DSL-based approach tailored to the specific needs for modeling global choreographies. The BCL implements all the concepts of UMM 2.0 (C1 – C8) as well as the additional concepts C9 – C11 as outlined above.

4 The Business Choreography Language (BCL)

From a conceptual point of view, a domain-specific language is a platform-independent approach tailored to the specific needs of a specific domain. In order to provide an appropriate modeling environment and tool support, the conceptual DSL needs to be implemented on a certain platform. Recently, two popular platforms for implementing a DSL-based modeling tool emerged - the Eclipse Modeling Project and the Microsoft Domain-Specific Language Tools (DSL Tools). Our Business Choreography Language (BCL) is based on the latter, which allows to implement a modeling tool on top of the Visual Studio IDE. In this paper, we do not elaborate on the details of creating a DSL for Visual Studio, but concentrate on the concepts of BCL. For more information on the DSL Tools, we refer the interested reader to [20].

We demonstrate the concepts of the BCL again by our order from quote example as known from section 2. In order to demonstrate the new concepts C9 – C11 as introduced in section 3 we extend the example. The resulting example modeled by means of the BCL is shown in figure 7.

Again, the process starts with the buyer sending a request for quote to the seller. If the seller provides a quote the buyer is able to order products within an agreed time-frame. If he does not place the order during that time-frame, the quote expires and the *business collaboration* ends prematurely. Otherwise, the buyer sends an order, which the seller can either accept or reject. In the latter case, the collaboration ends. Alternatively, the buyer is able to cancel the order as long as the order has not been shipped. The rounded dots in figure 7 highlight the particular concepts for modeling global choreographies as captured in section 3. In the following, we go through each of the concepts by means of our example and detail how we implemented them in the BCL.

A *business collaboration* (C1) comprises an arbitrary number of *business transactions* (C2). In our scenario the *business collaboration* order from quote consists of four *business transactions* - i.e., request for quote, place order, notify shipment, and cancel order. Please note that nesting *business collaborations* is not detailed in this paper. However, it follows an analogue approach to modeling *business transaction*. Furthermore, the diagram of a *business collaboration* comprises the roles that participate in a certain collaboration. The roles are depicted by the means of a *participants container* (C8a). The container is displayed as a rectangle on the lower right side of figure 7. According to our example scenario, the *participants container* defines a buyer and seller.

A *business transaction* is displayed as a rounded rectangle (C2). The *business transaction shape* is designed to visualize the most important information. The name of the transaction (e.g., request for quote) is displayed on the top of the transaction shape. Beneath it provides information about the *initiating role* and the *responding role*. The direction is indicated with an arrow, thereby the arrow allows leads from the *initiating role* on the left hand side to the *initiating role* on the right hand side. In the BCL, there exist only two generic role types for each *business transaction*: *initiator* and *responder*. Using the properties of a *business transaction*, the roles participating in a *business collaboration* (C8a) are mapped to the *initiating role* and the *responding role* of a *business transaction* (C8b), respectively. Considering request for quote as an example, the buyer is mapped to the *initiator* of the business transaction and the seller takes up the role of the *respondent*.

Beside the participants of a *business transaction*, the *business transaction pattern* is visualized within the shape as well (C3a). Similar to the mapping of participants, the corresponding *business transaction pattern* is set in the properties menu (C3b). The *business transaction* request for quote is a *commercial transaction*. The direction is visualized by two concurrent arrows. In case of a unidirectional transaction only one arrow will be displayed (e.g., for notify shipment).

The properties of a *business transaction* are directly linked with the *business transaction shape*. A change of a property value will immediately affect the visualization of the *business transaction*. In addition, custom code can be used in order to realize interdependencies between two or more properties. For example, there exists an interlink between the *business transaction pattern* property and the uni- and bi-directional icon visualized in the shape. A change of the *business transaction pattern* property will automatically change the icon. Furthermore, the *role mapping* mechanism has also constraints implemented in order to ensure an error-free mapping. These constraints ensure that the modeler is not able to choose other roles than specified within the *participants container* or to assign the same collaboration role to both transaction roles. In addition, the properties window is used to set non-visual information such as *quality of service parameters (QoS)* (C4).

A *business transaction* comprises two types of *business documents* (C5): *requesting documents* (C5a) and *responding documents* (C5b). The resulting shared state of a *business transaction* (which is a result of the *responding document*) governs the further control flow of a *business collaboration protocol* (C6a and C6b). In our example this is illustrated twofold: (i) the place order *business transaction* contains two *responding documents*: order acceptance envelope and order rejection envelope. Each of the two documents is the source of exactly one arc representing the control flow if the corresponding *responding document* has been received (C6a). In this case, the shared state is only given implicitly and not visualized on the diagram. (ii) The request for quote *business transaction* comprises only one *responding business document* although there are two alternative resulting shared states - quote.provided and quote.refused - as shown in figure 3. This means that not the type of the *responding business document*, but the actual content of the document decides upon the resulting shared state. In figure 7, we use shared states also explicitly (C6b) to govern the control flow after request for quote. In this case, the determination of states requires the setting of conditions in the properties windows, which is beyond the scope of this paper. The *event-based XOR* decision (C11) allows to model a case, which was not possible using UMM 2.0: the buyer can cancel his order until the seller sends a notification of shipment. The initiation of either *business transaction* in the example corresponds to an event, which allows to model them as successors of the *event-based XOR* node. This example demonstrates deferred choice behavior. Another deferred choice is used before the place order *business transaction* in order to reflect a possible time-out of the quote. In other words, the provided quote is only valid for a certain time-period represented by the *timer-event* (C9). If the time-period expires, the corresponding *timer-event* is triggered, which results in an overall business failure.

In case an exception occurs during the execution of a *business transaction*, the effects of prior *business transactions* may be required to be reversed. Since a classical roll-back (like for databases) is not possible for inter-organizational systems, compensation behavior is required (C10). Thus, we introduce the concept of *compensating business transactions*. Considering our example, there is a need for compensating behavior when an exception occurs in notify shipment (e.g., when the seller has troubles to communicate with the shipper). In this case, the effects of place order need to be reversed by a *compensating business transaction*. Therefore, we assign a *compensating business*

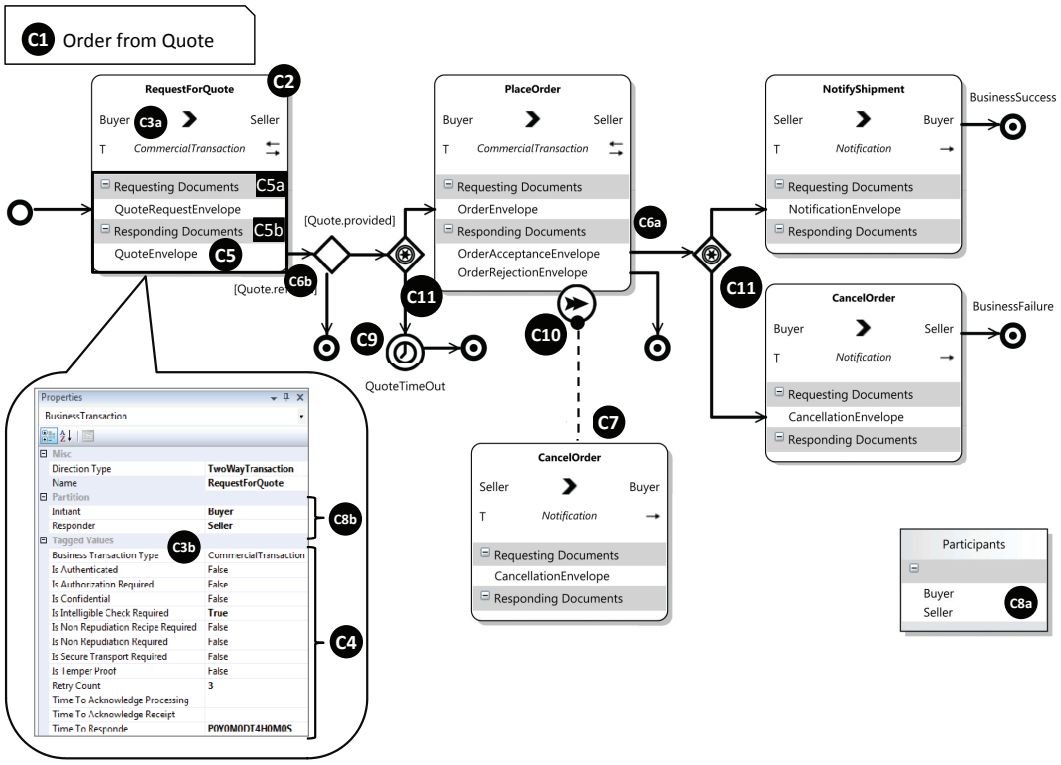

Figure 7: The Business Choreography Language: Order From Quote Business Collaboration

transaction (cancel order) to place order using a *compensation event* and a *compensation flow*. The *compensation event* is visualized as rounded port with a double arrow. The *compensation flow* is displayed as a dashed line between the port and the cancel order *business transaction*.

The duplicate use of cancel order also demonstrates the concept of re-use (C7). The cancel order *business transaction* has already been modeled once - see the cancel order *business transaction* on the right hand side of figure 7. Thus, we do not have to model it again (involving the definition of the QoS parameters), but can re-use it. We only have to set the roles when re-using *business transaction*, which results in switching buyer and seller, since the seller has to send the cancellation to the buyer when place order is going to be compensated.

5 Evaluation and future work

In this paper, we dealt with the problem space of abstracting global B2B choreographies by graphical models. The contribution of this paper is twofold: (i) we presented a detailed list of criteria reflecting required concepts for modeling global choreographies. (ii) Based on this criteria, we developed the Business Choreography Language (BCL). The design of the BCL was heavily influenced by UN/CEFACT's Modeling Methodology (UMM), which builds upon the UML.

However, as a result of our work on the UMM we have come to the conclusion that using UML as the underlying notation is not an optimal solution: The reason is that UML profiles may indeed constraint the UML meta model, but must evidently still adhere to it. This led into several criticisms of UMM by stakeholders: (i) UMM comprises an extensive set of modeling elements (i.e., stereotyped UML elements). (ii) UMM models result in an overwhelming amount of modeling artifacts. (iii) UMM necessitates a many re-occurring tasks (e.g., for creating the flow of business transactions). (iv) The visualization of UMM has a lack in expressiveness and readability.

Due to these deficiencies, we followed an alternative approach to represent the concepts. This approach permits us to define our own language including a graphical visualization in order to efficiently abstract from the problem domain. The DSL has been carefully designed in order to comprise all the relevant concepts for modeling global choreographies. At the same, by using our own notation we are able to avoid unnecessary modeling elements and modeling steps - resulting in an overall streamlined process of designing global choreographies. This reduces the complexity and enhances at the same time the readability of BCL models compared to the UMM. This makes BCL understandable for all stakeholders - including business own-

ers, business analysts, and technicians.

Another advantage of the DSL approach concerns validation. Although UML profiles may define constraints on the UML meta model by means of the Object Constraint Language (OCL) [21], on-the-fly validation of OCL constraints within UML tools is currently not supported. In contrast, the validation of domain-specific models can be done on a much higher level using the DSL Tools for Visual Studio. Due to the abstracted view on the solution syntactical correctness can be "hard-wired" into the model. These so called hard-constraints are modeled directly at the meta-level and ensure that instances of the model have to be modeled according to the meta-model constraints. This leads to more efficiency during the development cycle since development time decreases while quality increases. For example, a *business collaboration* may only contain *business transaction elements*, *control flow elements*, and a *participants container*. Already by design, it cannot hold any other modeling elements. Another example is the simplified mapping of roles, which ensures that the modeler can only choose valid roles for the *initiating role* and the *responding role* of a *business transaction*, respectively.

The BCL introduced in this paper yields the following benefits compared to the UML profile for UMM:

- Increased readability and expressiveness using customized shapes
- *Business collaborations* and *business transactions* can be modeled in a single diagram
- Simplified mapping between roles of a *business collaboration* and roles of *business transactions*
- Reduced set of modeling elements
- Reduced amount of resulting modeling artifacts
- Simplified model validation

Future work concentrates on generating code from BCL models. The DSL Tools for Visual Studio include the Text Template Transformation Toolkit (T4) that supports the generation of code artifacts. Candidate platforms for code generation are, for example, Web Services (i.e., BPEL [22] and WSDL [5]) and Windows Workflow.

References

[1] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, Oct. 2003.

[2] *Business Process Modeling Notation Specification (BPMN)*, Object Management Group (OMG), Jan. 2009, Version 1.2.

[3] G. Decker and A. Barros, "Interaction Modeling Using BPMN," in *Business Process Management Workshops*, ser. Lecture Notes in Computer Science, vol. 4928. Springer Berlin / Heidelberg, 2008, pp. 208–219.

[4] J. Zaha, A. Barros, M. Dumas, and A. ter Hofstede, "Let's Dance: A Language for Service Behavior Modeling," in *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, ser. Lecture Notes in Computer Science, vol. 4275. Springer Berlin / Heidelberg, 2006, pp. 145–162.

[5] *Web Services Choreography Description Language*, World Wide Web Consortium (W3C), Nov. 2005, version 1.0, W3C Candidate Recommendation.

[6] UN/CEFACT, *UN/CEFACT - ebXML Business Process Specification Schema Version 1.10*, Nov. 2003.

[7] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for Modeling Choreographies," in *Proc. IEEE International Conference on Web Services ICWS 2007*, Jul. 9–13, 2007, pp. 296–303.

[8] *UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model 2.0*, UN/CEFACT, 2008, public Draft V2.0.

[9] *Unified Modeling Language*, OMG, February 2009, technical Specification V2.2.

[10] B. Selic, "A systematic approach to domain-specific language design using uml," pp. 2–9, May 2007.

[11] E. Visser, *Generative and Transformational Techniques in Software Engineering II*, ser. Lecture Notes in Computer Science. Springer, 2008, ch. WebDSL: A Case Study in Domain-Specific Language Engineering, pp. 291–373.

[12] D. Parnas, "On the design and development of program families," *Software Engineering, IEEE Transactions on*, vol. SE-2, no. 1, pp. 1–9, March 1976.

[13] J. Bentley, "Programming pearls: little languages," *Commun. ACM*, vol. 29, no. 8, pp. 711–721, 1986.

[14] J. Greenfield, K. Short, S. Cook, and S. Kent, *Software Factories*, 1st ed. Wiley Publishing Inc., September 2004.

[15] *UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module*, UN/CEFACT, Mar. 2006, technical Specification V1.0.

[16] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modelling Language Reference Manual*. Addison-Wesley, 1999.

[17] C. Huemer, P. Liegl, T. Motal, R. Schuster, and M. Zapletal, "The development process of the un/cefact modeling methodology," in *ICEC '08: Proceedings of the 10th international conference on Electronic commerce*. New York, NY, USA: ACM, 2008, pp. 1–10.

[18] N. Russell, A. ter Hofstede, W. van der Aalst, and N. Mulyar, "Workflow control-flow patterns: A revised view," BPM Center Report, Tech. Rep., 2006, bPM-06-22, BPMcenter. org.

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

[20] S. Cook, G. Jones, S. Kent, and A. C. Wills, "Domain-specific development with visual studio dsl tools," June 2007.

[21] *Object Constraint Language Specification*, OMG, April 2009, version 2.0.

[22] *Web Services Business Process Execution Language (BPEL)*, OASIS, April 2007, Version 2.0.