Transforming Collaborative Business Process Models into Web Services Choreography Specifications

Pablo David Villarreal[1], Enrique Salomone[1,2], and Omar Chiotti[1,2]

[1] CIDISI, Universidad Tecnológica Nacional - Facultad Regional Santa Fe, Lavaisse 610, 3000, Santa Fe, Argentina
pvillarr@frsf.utn.edu.ar
[2] INGAR-CONICET, Avellaneda 3657, 3000, Santa Fe, Argentina
{salomone, chiotti}@ceride.gov.ar

Abstract. Languages for web services choreography are becoming more and more important for B2B integration. However, the development of web services-based systems is complex and time-consuming. Enterprises have to agree on collaborative business processes and then derive their respective web services choreographies in order to implement B2B collaboration. To support it, this paper presents a MDA approach for collaborative processes. We describe the components and techniques of this approach. We show how collaborative process models defined with the UP-ColBPIP language can be used as the main development artifact in order to derive choreography specifications based on WS-CDL. The transformations to be carried out are also discussed. The main advantage of this MDA approach is that it guarantees that the generated web services choreographies fulfill the collaborative processes agreed between the partners in a business level.

1 Introduction

Web services are having more and more interest to implement B2B information systems for carrying out inter-enterprise collaborations. The need to specify these collaborations based on this technology has led to the requirement of web services choreography languages. There are several standards proposed to describe web services-based business processes (also known as web services composition). They can be classified according to three types of web services composition that can be defined [3]: Choreography, Behavioral Interface and Orchestration.

Choreography describes collaborative processes involving multiples services, i.e. it describes the global view of the interactions between the services of the partners without considering private details of processing required by the partners. The languages ebXML Business Process Specification Schema (BPSS) [14] and Web Services Choreography Description Language (WS-CDL) [9] support the definition of choreographies. BPSS is not oriented to web services. It allows defining business transactions into binary collaborations. WS-CDL is focused on web services and supports multi-party collaboration. *Behavioral Interface* describes collaborative processes from the point of view of one partner, i.e. the order in which a partner sends messages to and receives messages from its partners, and hence it describes the public aspects of a web service including its observable behavior. It is also known as conversation protocol [2], business protocol or abstract process [4]. In B2B collaborations, behavioral interfaces of the partners should be derived from choreographies or collaborative processes agreed between them. *Orchestration* describes both public aspects (derived from the behavioral interface) and private aspects of the web services, i.e. the service business logic (e.g. internal rules) that supports the partner's behavior in the interaction with other partners.

In this paper we focus on the generation of web services choreographies in order to define the logic of the collaborative processes in a technological level. Currently, web services choreographies can be specified using the language WS-CDL. However, as it has been recognized by other authors, choreographies are more a design that an implementation artifact [3]. They are not intended to be directly executed. Therefore, for design purposes, the use of an XML-based language to specify choreographies is less useful than a graphical modeling language. Moreover, business aspects cannot be captured with WS-CDL. A WS-CDL choreography is defined as one that describes a collaboration between services in order to achieve a common goal. But support is not provided to define common goals or other business aspects. In this way, a graphical modeling language should be provided as well as a procedure for generating automatically choreographies in a technical language such as WS-CDL.

Due to the fact that choreographies describe interactions as part of the collaborative processes that partners agree to achieve common goals, they should be derived from collaborative business process models. These models are designed in a business level by business engineers and system designers, who are not acquainted with the technical details of the collaboration. Hence, collaborative process models should be independent of the technology to enable their implementation by using different B2B standards according to the technological requirements of the partners [1].

To support the above issues, the Model-Driven Architecture (MDA) Initiative [12] has been identified as a key enabler to support the modeling and specification of collaborative processes [17, 18]. A MDA approach enables both the design of collaborative processes independent of the idiosyncrasies of particular B2B standards, and the automatic generation of B2B specifications based on a B2B standard from conceptual collaborative process models. As part of this approach, the UML Profile for Collaborative Business Processes based on Interaction Protocols (UP-ColBPIP) has been defined [15,18]. This language allows business engineers to define several views of the collaboration, from the definition of the partners, their roles and the common goals, up to the definition of the interaction protocols that realize the collaborative processes and the provided and required business interfaces of the roles.

In a previous work, we described the generation of technological solutions based on ebXML BPSS using a MDA approach [17]. We showed how most of the concepts used by BPSS can be derived from the conceptual elements provided by UP-ColBPIP. In another previous work, we described the generation of technological solutions based on web services composition using a MDA approach [19]. In this case we show how, for each partner involved in a B2B collaboration, BPEL abstract processes can be derived from collaborative process models defined with UP-ColBPIP. In this way, we had applied a MDA approach to generate B2B specifications and we also have validated the UP-ColBPIP language against standard languages. The conclusions were most of the UP-ColBPIP concepts can be represented in BPSS or BPEL.

This paper discusses the application of a MDA approach for collaborative processes to derive WS-CDL specifications. The purpose is to validate that UP-ColBPIP provides the required conceptual modeling elements to generate specifications based on web services choreographies. We do not aim to provide a standard language but to study suitable conceptual elements a collaborative process modeling language should provide, in order to support the modeling in a business level and enable the automatic generation of specifications based on standard languages in a technological level.

This paper is organized as follows. Section 2 describes the MDA approach we propose for collaborative processes. Section 3 describes the transformation of UP-ColBPIP models into WS-CDL and WSDL specifications. A brief description of the conceptual elements provided by UP-ColBPIP is also provided. Section 4 discusses related work. Section 5 presents conclusions and outlines future research directions.

2 MDA Approach for Collaborative Business Processes

The OMG's MDA initiative proposes a conceptual framework along with a set of standards (UML, MOF, XMI, etc.) to build model-driven development methods. In MDA, the development process consists of: defining platform or technology-independent models (PIMs), selecting the platform-specific models (PSMs) and executing the transformations that generate PSMs from PIMs, and finally generating the code from the PSMs. In MDA, the concept of system can refer to software, an enterprise, a set of enterprises and so on. In the domain of collaborative processes, the system to be built includes the specifications of: collaborative processes and partners' interfaces, both defined with a technology-specific language.

Figure 1 shows the corresponding components of the MDA approach we are proposing along with the techniques we provide to build the components:

- **Collaborative Business Process Models based on UP-ColBPIP.** They are the technology-independent process models and are built through the modeling language UML Profile for Collaborative Business Processes based on Interaction Protocols (UP-ColBPIP) [15, 17, 18]. This UML Profile is based on UML 2.
- **WS-CDL and WSDL models.** WS-CDL models represent the technology-specific collaborative process models. Web services composition standards are based on the Web Service Description Language (WSDL), which is used to define the web services of the partners. WSDL models represent the technology-specific partners' interfaces. To build WS-CDL and WSDL models we define their corresponding metamodels, which can be derived from the XML schemas provided by these standards. Thus, a model corresponds to a XML document. Although the XML code may be directly generated from UP-ColBPIP models, this intermediate representation allows the transformation be more modular and maintainable.
- **Transformations of UP-ColBPIP models into WS-CDL and WSDL models.** These transformations define a set of transformation rules to allow the generation of WS-CDL models from UP-ColBPIP models. They define the correspondence between UP-ColBPIP concepts and WS-CDL and WSDL concepts. These transformation rules are implemented through a method and a tool for model transformations, which support the definition and automatic execution of the rules.

- **WS-CDL and WSDL Specifications.** The final outputs of the transformations are the XML files of the WS-CDL process specifications and the WSDL partners' interfaces specifications. The transformation of technology-specific models into the corresponding specifications is almost direct. This can be supported through XML production rules that convert a UML class models into a XML version.

In this work, we focus on UP-ColBPIP models and the definition, in a conceptual way, of the transformations of UP-ColBPIP into WS-CDL. Transformations of UP-ColBPIP into WSDL were described in [19]. The other techniques and components are also out of the scope of this paper. They can be found in [15].

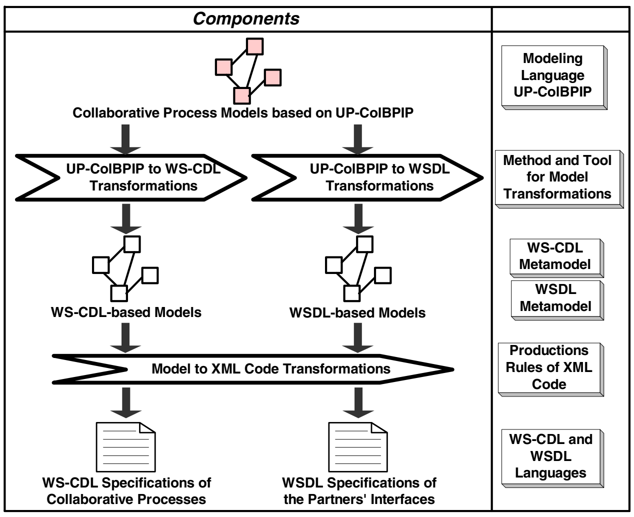
Components
Collaborative Process Models based on UP-ColBPIP
UP-ColBPIP to WS-CDL Transformations
UP-ColBPIP to WSDL Transformations
WS-CDL-based Models
WSDL-based Models
Model to XML Code Transformations
WS-CDL Specifications of Collaborative Processes
WSDL Specifications of the Partners' Interfaces
Modeling Language UP-ColBPIP
Method and Tool for Model Transformations
WS-CDL Metamodel
WSDL Metamodel
Productions Rules of XML Code
WS-CDL and WSDL Languages

Fig. 1. MDA Approach for Collaborative Processes

3 The Modeling Language UP-ColBPIP

UP-ColBPIP is a modeling language to design technology-independent collaborative processes. It encourages a top-down approach and provides the conceptual elements to support the modeling of four views:

- *B2B Collaboration View*, which defines the partners, the roles they fulfill, their relationships, the collaborative agreement parameters and the hierarchy of common business goals to be fulfilled by the partners in a B2B collaboration.
- *Collaborative Processes View*, which defines the processes partners have to perform. They are defined informally extending the semantics of use cases.
- *Interaction Protocols View*, which defines the interaction protocols that realize the collaborative processes.
- *Business Interfaces View*, which defines the business interfaces required by the partners to support the exchange of messages of the interaction protocols.

The first and second views correspond to the analysis stage, in which the requirements of collaborative processes are defined. The last views correspond to the design stage of collaborative processes. From these last views, technological solutions can be generated. Following we describe the Interaction Protocols View.

3.1 The Interaction Protocols View

This view focuses on the design of interaction protocols that realize the behavior of the collaborative processes. In the B2B collaborations domain, an *interaction protocol* describes a high-level communication pattern through a choreography of business messages between partners playing different roles [16]. The purpose of modeling collaborative processes based on interaction protocols is to fulfill the requirements of B2B collaborations [15,16]: enterprise autonomy, decentralization, peer-to-peer interactions, global view of the collaboration and support for negotiations.

In contrast to activity-oriented processes, interaction protocols focus on the exchange of business messages representing interactions between partners. Activities each partner performs for processing the information to be received or producing the information to be sent are not defined in the interaction protocols.

In addition, B2B interactions cannot be restricted to mere information transfer [7]. They also have to express the communication of actions between the partners. Communicative aspects can be represented in interaction protocols through the use of speech acts [13]. In an interaction protocol, a business message has a speech act associated, which represents the intention that a partner has with respect to an exchanged business document through the message. Furthermore, decisions and commitments done by the partners can be known from the speech acts. This enables the definition of complex negotiations in collaborative processes.

UP-ColBPIP extends the semantics of UML2 Interactions to model interaction protocols. Hence, they are defined using UML2 Sequence Diagrams. Following we describe the main conceptual elements used to define interaction protocols.

As an example, we describe the interaction protocol *Demand Forecast Request*, which realizes a simplified process to manage collaborative demand forecasts. Figure 2 shows the sequence diagram of this protocol, in which partner “A” plays the role *supplier* and partner “B” plays the role *customer*. They are defined by lifelines.

The basic building block in an interaction protocol is a *business message*. It defines an interaction between two roles, a sender and a receiver. A business message contains a *business document* and its semantics is defined by its *speech act* associated. In this way, a business message expresses the sender has done an action, which generates the communication of a speech act representing the sender’s intention with respect to the exchanged business document. Also, the message indicates the sender’s expectative that the receiver then acts according to the semantics of the speech act. For example, in the message *request(DemandForecast)*, its associated speech act indicates the supplier’s intention of requesting a demand forecast to the customer. It also implies the customer has to respond with a suitable speech act, such as *agree* or *refuse*. The suitable speech acts to be used depend on the speech act library selected.

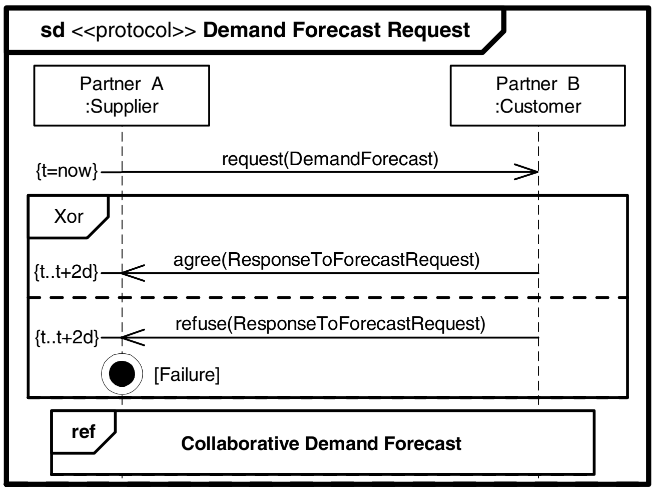
sd <<protocol>> Demand Forecast Request
Partner A :Supplier
Partner B :Customer
{t=now}
request(DemandForecast)
Xor
{t..t+2d}
agree(ResponseToForecastRequest)
{t..t+2d}
refuse(ResponseToForecastRequest)
[Failure]
ref
Collaborative Demand Forecast

Fig. 2. Sequence Diagram of the Interaction Protocol *Demand Forecast Request*

A business message is a one-way asynchronous communication and is managed by the receptor just as a signal to be interpreted for activating its internal behaviors. This feature is essential in B2B interactions because the sender's internal control should not be subordinated to the receiver's response.

In addition, a business message may require the sending of a receipt and/or a read acknowledgment by the receiver towards the sender, for indicating to the sender that the message has been received and/or understood by the recipient. It is defined in the atributes *isReceiptAcknowledgementRequired* and *isReadAcknowledgementRequired.*

A *business document* represents the information conveyed by the message. In the example, the business document *DemandForecast* contains details about the period, products and time horizon required for the forecast.

A *Control Flow Segment* represents complex message sequences in the interaction protocol's choreography. It contains a control flow operator and one or more interaction paths. An *interaction path* can contain any protocol element: messages, terminations, interaction occurrences, and nested control flow segments. The stereotype *control flow segment* extends the semantics of the *combined fragment* of UML2 to provide suitable control flow operators for defining collaborative processes.

The semantics of a control flow segment depends on the operator used: Xor, Or, And, If, Loop, Transaction, Exception, Stop and Cancel. The *And* operator represents the execution of parallel interaction paths in any order. The *Xor* operator represents that only one path, of a set of alternative paths, can be executed in case its condition is evaluated to true. The *Or* operator represents the selection of several paths from several alternatives. Each path is executed in case its condition is evaluated to true. The *If* operator represents a path that is executed when its condition is true, or nothing is executed. This can also have an *else* path, which is executed when the condition of the first path is false. The *Loop* operator represents a path that can be executed while its condition is satisfied. Two types of *Loop* segments can be defined: a loop "For" with the condition "(1,n)", where its path must be executed at least one time; and a loop "While" with the condition "(0,n)", where its path can be executed zero or *n* times. The *Transaction* operator indicates messages and paths of the segment have to be done atomically, and messages cannot be interleaved with messages of other paths. The *Exception* operator represents a path to be followed if an exception occurs according to the path's condition. The *Stop* operator represents paths that manage exceptions and require the abrupt termination of the protocol. The *Cancel* operator represents paths to be followed to manage exceptions. Different to *Stop* and *Exception* operators, the exception to be managed can occur in any point of the interaction protocol. A segment with this operator has to be defined at the end of the protocol.

The protocol of Figure 2 has a control flow segment with the operator *Xor* and two paths. This segments indicates the *customer* can respond in two way: it accepts the supplier's request and commits to carry out the demand forecast in the future (message *agree*); or it rejects the supplier's request and then the protocol ends.

Conditions represent logical expressions that constraint the execution of a message or a path in a control flow segment. They are defined in natural language or using OCL (Object Constraint Language) expressions.

Duration and Time constraints are used to define duration and deadlines on messages or protocols. They can be defined using relative or absolute dates. In Figure 2, the last two messages have a relative time constraint, which indicates the messages have to be sent before two days, after the occurrence of the first message.

An *Interaction Occurrence* represents the invocation of another interaction protocol referred as the nested protocol. In Figure 2, if the *customer* agrees on the *supplier's* request, then the nested protocol *Collaborative Demand Forecast* is invoked, in which the customer and the supplier agree on a common forecast.

Finally, an interaction protocol can have implicit or explicit *terminations*. Explicit termination events are: *success* and *failure*. *Success* implies the protocol ends in a successful way. Failure implies the protocol ends in an unexpected way, but from the point of view of the business logic. In Figure 2, if the customer rejects the supplier's request, the protocol ends with a failure. Else, after invocating to the nested protocol, the protocol finishes successfully in an implicit way.

4 Generation of WS-CDL Specifications from UP-ColBPIP Models

In this section we discuss the generation of WS-CDL specifications from models of collaborative processes defined with UP-ColBPIP. First, we describe the generation of type definitions of a WS-CDL document. Then, we describe the transformation of UP-ColBPIP elements into WS-CDL elements required to generate choreographies from the protocols defined in a UP-ColBPIP model. We describe the output of the transformations using the protocol defined in the above section. Figure 3 shows some parts of the WS-CDL document that is generated from this interaction protocol.

4.1 Generation of the Package and Type Definitions of a WS-CDL Document

The root element of a WS-CDL document is the *package*, which contains type definitions. In UP-ColBPIP, the root element is the B2B collaboration. Therefore, a B2B Collaboration is mapped into a package in WS-CDL (see line 1 in Figure 3).

Package Information and Type Definitions

```
1    <package name=”B2BCollaborativeForecasting” ...>
2      <informationType name=”DemandForecast” .../>
3      <informationType name=”ResponseToForecastRequest” .../>
         ...
4      <roleType name=”Customer”>
5        <behavior name=”request_DemandForecast”
6                 Interface=”customer.wsdl”/>
7      </roleType>
8      <roleType name=”Supplier”>
9        <behavior name=”agree_ResponseToForecastRequest”
10                Interface=”supplier.wsdl”/>
11       <behavior name=”refuse_ResponseToForecastRequest”
12           Interface=”supplier.wsdl”/>
13     </roleType>
14     <relationshipType name=”CustomerSupplier>
15       <role type=”supplier”/>
16       <role type=”customer”/>
17     <relationshipType/>
18     <participantType name=”Partner B”>
19       <role type=”customer”/>
20     <participantType/>
21     <participantType name=”Partner A”>
22       <role type=”supplier”/>
23     <participantType/>
24     <channelType name=”CustomerChannel”
25       action=”request”>
26       <role type=”Customer”/>
27       <reference>
28         <token name=”customerRef”/>
29       </reference>
30     </channelType>
           ....
```

Choreography Definitions

```
31   <choreography name=”DemandForecastRequest”
32        complete=”cdl:globalizedTrigger(‘termination’,
33             ‘customer’,’SuccessTermination’,‘supplier’)or
34             cdl:globalizedTrigger(‘termination’,‘customer’,
35             ’FailureTermination’,‘supplier’)”
36        isolation="false" root=”true” coordinated="true">
37     <relationship type=”CustomerSupplier”/>
38     <variableDefinitions>
39       <variable name=”customer_channel”
40           channelType=”CustomerChannel”
41           roleType=”Customer”/>
42       <variable name=”demandForecast”
43           informationType=”DemandForecastType”
44           roleType= “Customer,Supplier” />
45       <variable name=”FailureTermination”
46           informationType=”BooleanType”
47           roleType= “Customer,Supplier” />
             ....
48     </variableDefinitions>
```

Choreography Definitions (cont.)

```
49     <sequence>
50       <interaction name=”request_DemandForecast”
           ....
51       </interaction>
52       <choice>
53         <sequence> <!-- It represents the first interaction path -->
54              ...
55         </sequence>
56         <sequence> <!-- It represents the second interaction path -->
57          <interaction name=”refuse_ResponseToForecastRequest”
58                 operation=”refuse_ResponseToForecastRequest”
59                 channelVariable=”supplier_channel”
60                 align=”true” initiate=”false”>
61           <participate relationshipType=”CustomerSupplier”
62                      fromRole=”Customer” toRole=”Supplier”/>
63           <exchange name=”ResponseToForecastRequest”
64                informationType=”ResponseToForecastRequestType”
65                action=”request”>
66             <send variable=”cdl:getVariable(‘
67                    ResponseToForecastRequest,’Customer’)”/>
68             <receive variable=”cdl:getVariable(‘
69                    ResponseToForecastRequest,’Supplier’)”/>
70           </exchange>
71           <timeout time-to-complete=”t..t+2d”
72                fromRoleTypeRecordRef=”record-timeout”
73                toRoleTypeRecordRef=”record-timeout”/>
74                <record name=”record-timeout”
75                     when=”timeout”
76                     causeException=”TimeOutException”>
77                  <source variable=”True”/>
78                  <target variable=”cdl:getVariable(‘
79                       timeOutException’,’Supplier’,’’)”/>
80                </record>
81           </interaction>
82           <assign roleType=”Customer”>
83             <copy name=”copyToCompleteChoreography”
84               <source expression=”true”/>
85               <target variable=”cdl:getVariable(‘
86                  FailureTermination’,’’,’’)”/>
87             </copy>
88           </assign>
89         </sequence>
90       </choice>
91      <perform choreographyName=”CollaborativeDemandForecast”
92           block=”true”/>
93     </sequence>
94   </choreography
```

Fig. 3. Fragments of the WS-CDL document generated from the protocol of Figure 2

WS-CDL *informationType* definitions are derived from the *business document types* defined in the collaborative processes view of the UP-ColBPIP model. They represent a business document type from a content B2B standard and contain a reference to the XML Schema provided by the standard. They are used to indicate the type of the business documents exchanged in collaborative processes. In Figure 3, lines 2-3 define the business document types to be used for the protocol of Figure 2. Furthermore, general information types, such as *boolean*, can also be generated.

A *role* defined in a B2B collaboration is mapped into a *roleType* in the WS-CDL document. A roleType enumerates observable behavior a participant can exhibit in order to interact. This behavior is defined according to the operations provided by a WSDL interface of the participant. In the business interfaces view of a UP-ColBPIP model, *business services* of the *required and provided interfaces* of a role indicates the business messages that roles can send and receive, respectively. Hence, a *behavior* is generated in a roleType for each business service defined in the *provided business interface* of the role. For example, in Figure 3, lines 4-13 define the role types and its behaviors corresponding to the roles involved in the protocol of Figure 2.

A *relationshipType* in WS-CDL defines a relationship between two roles and optionally the subset of the behavior they exhibit. A relationshipType is derived from the B2B relationship connector defined in the B2B collaboration, which specifies a static relationship between two roles. As an example, see lines 14-17 in Figure 3.

A *partner* can play several roles in a B2B collaboration of a UP-ColBPIP model. Therefore, for each partner a *participantType* is defined in WS-CDL with the roles it has to play. For example, in Figure 3, lines 18-23 show the participants generated.

The last type definitions are *channelTypes*. There is not a corresponding element in UP-ColBPIP. However, a channelType is defined for each role to indicate the channel through which the role will receive messages (e.g.: see lines 24-30 in Figure 3).

4.2 Transformation of Interaction Protocols into WS-CDL Choreographies

An interaction protocol is mapped into a *choreography*. For the root interaction protocol of the UP-ColBPIP model, a root choreography is generated. Choreographies derived from interaction protocols are generated in the same package, because it corresponds to the B2B collaboration where the protocols were defined. Several attributes describes a WS-CDL choreography. The *isolation* attribute is set to “false” because variables (e.g.: business documents) defined in an interaction protocol can only be visible within the protocol. However, they can be correlated with variables of nested protocols as it is explained further on. The *coordination* attribute is set to “true” since an interaction protocol assumes the roles agree on how it is ended.

References to relationships between roles are generated matching the roles of the relationship types and the roles of the interaction protocol that is being transformed.

Then, choreography’s *variables* are defined. Business documents used in the interaction protocol are mapped into variables with the corresponding information type. A variable used in a condition of a control flow segment is also mapped into a variable in the choreography. Channel variables are generated for each channel type defined. Variables generated are made available to each role.

In Figure 3, lines 31-48 show the choreography generated from the interaction protocol of Figure 2. Some of the variables generated are also showed.

Once the choreography has been defined, its activities are derived from the elements that make up the interaction protocol’s choreography, as it is described below.

4.2.1 Transformation of Business Messages

A business message of an interaction protocol is transformed into an *interaction*. Both elements are the basic building block in their respective languages. An interaction in WS-CDL can be: a request, a response or a request-response (i.e a synchronous interaction). To represent the asynchronous communication of a business message, it is mapped into a request interaction.

Acknowledgements in a business message are used to assure the state synchronization between two roles when they exchange a message. WS-CDL does not support the definition of acknowledgments in an interaction. However, state synchronization can be achieved in WS-CDL defining an interaction as aligned, i.e. setting to “true” the *align* attribute. Another solution is to generate a request-response interaction where the response exchanges correspond to the acknowledgements.

The *operation* attribute of an interaction, which specifies what the recipient of a WS-CDL message should do when it is received, is derived from the signature of the business message, because it corresponds to one of the business services defined in the provided business interface of the receiver role. The *channelVariable* of the interaction is generated according to the channel in which the role is the target of the interaction. The *initiate* attribute is true if the business message represents the first interaction in the interaction protocol. The *participants* of the interaction are derived from the lifelines representing the receiver and sender role of the business message.

A business document conveyed in a business message is mapped into an *exchange*, which defines the information to be exchanged during the interaction. The *action* attribute is set to “request” to represent the asynchronous communication of the business message. In addition, the *send* and *receive* variables of the exchange are also generated from the business document of the business messages, in order to both roles save the exchanged business document.

If a business message has a duration or time constraint associated, a *timeout* is generated. If it has a duration constraint, the *time-to-complete* attribute contains the timeframe within which an interaction must complete after it was initiated. If the message has a time constraint, the *time-to-complete* attribute contains the deadline within which an interaction must complete after it was initiated. The duration and time constraints are those provided by UML. They can be defined in relative or absolute time and using intervals. However, in WS-CDL a timeframe or a deadline has to be defined in absolute time. Hence, the *time-to-complete* attribute is generated with the same value (in relative or absolute time) defined in the duration or time constraint of the business message. Then, the developers have to redefine the value of this attribute in case a relative time has been used. The *fromRoleTypeRecordRef* and *toRoleTypeRecordRef* attributes are also generated with a reference to a *record* of the interaction, which is used to notify both roles when a timeout exception occurs. In this case, the *when* attribute is set to “timeout”.

Finally, a business message of an interaction protocol can have multiple instances. However, multiple instances of an interaction are not supported in WS-CDL.

In Figure 3, lines 57-81 define the interaction generated from the business message *refuse(ResponseToForecastRequest)* of the protocol of Figure 2.

4.2.2 Transformation of Control Flow Segments

They are transformed into WS-CDL control flow activities according to the operator used (summarized in Table 1). WS-CDL control flow activities are: *sequence*, *parallel*, *choice* and *workunit*. They capture the basic control flow constructs.

A control flow segment (CFS) with the operator *Xor* is mapped into a *choice* activity. This type of CFS can represent an event-driven choice or a data-driven choice, such as the choice activity of WS-CDL. A CFS without conditions represents an event-driven choice, meaning that the choice depends on the occurrence of the first element defined in one of the interaction paths. Else, a CFS with conditions represents a data-driven choice. The first type of CFS is mapped into a choice activity, and a sequence activity is generated for each interaction path. The second type of CFS is mapped into a choice activity and a workunit activity is generated for each interaction path. The condition of the path is transformed into a XPath expression defined in the *guard* attribute of the workunit. Its *repeat* attribute is set to "false". Its *block* attribute is also set to "false" because variables used in the conditions of CFSs can be different in each interaction path and it is assumed that the variable information is available at the moment of the evaluation.

Table 1. Transformation of control flow segments into WS-CDL control flow activities

Control Flow Segment	WS-CDL Control Flow Activity
Xor (data-driven choice)	A Choice and a Sequence activity for each interaction path
Xor (event-driven choice)	A Choice and a WorkUnit activity for each interaction path
Or	A Parallel activity and a Choice for each alternative interaction path. In the Choice activities, two WorkUnits are defined.
And	Parallel activity
Loop (0,n)	WorkUnit. Guard="Repetition condition". Repeat="True"
Loop (1,n)	WorkUnit. Guard="True". Repeat="Repetition condition"
If	WorkUnit. If it contains two paths, it is mapped as a Xor
Stop	Exception WorkUnit
Cancel	Exception WorkUnit

A CFS with the operator *And* is mapped into a *parallel* activity and a *sequence* activity is generated for each interaction path.

A CFS with the operator *Or* cannot be mapped into a direct way in WS-CDL. Like BPEL, it does not support a construct representing several activities can be executed and at least one must be executed. However, this CFS can be represented in WS-CDL in the following way. A parallel activity is generated and within this activity a choice activity is generated for each alternative interaction path. Within the choice activity, two workunit activities are defined, one representing the activities to be carried out in case of the condition is evaluated to true and another one to represent the opposite case. This last activity is generated to guarantee the termination of the parallel activity because it completes successfully when the enclosed activities complete successfully.

A CFS with the operator *If* is mapped into a *workunit* if it has only one interaction path. The *block* and *repeat* attributes are set to "false" and the *guard* attribute contains the CFS's condition. If the CFS has two interaction paths, it is transformed into the same way that a CFS with the operator *Xor* and two interaction paths.

WS-CDL does not distinguish between loop "For" and loop "While". Loops can be defined with a *workunit*. To represent a CFS with a loop "For", a workunit activity is generated with the *guard* attribute settled to "true" and the *repeat* attribute with the repetition condition. To represent a CFS with a loop "While", a workunit is generated with the *repeat* attribute settled to "true" and the *guard* attribute with the repetition condition. In both cases repetition condition is derived from the CFS's condition.

A CFS with the operator *Transaction* cannot be mapped into any construct of WS-CDL. It does not support the definition of transactions for interactions, such as in ebXML BPSS. A CFS with the operator *Exception* cannot also be mapped into any construct of WS-CDL. It does not support the definition of exception blocks in a specific point of the interaction and that does not require the end of the choreography.

A CFS with the operator *Stop* or *Cancel* is mapped into an exception workunit defined within the *exceptionBlock* of the choreography. Although WS-CDL does not support the definition of exception blocks in specific points of the choreography, the CFS with the *Stop* operator can be transformed into an exception workunit, where its *guard* attribute contains the exception condition derived from the path's condition of the CFS. If a CFS with the operator *Cancel* has an interaction path with the condition "TimeException", an exception workunit is generated with a *guard* using the WS-CDL function *hasExceptionOccurred*. The parameter *exceptionType* of this function contains the name of the caused exception according to the defined in the timeout elements of the generated interactions. In the other cases, for each interaction path defined in the CFS with the operator *Cancel*, an exception workunit is generated with the *guard* attribute containing the exception condition defined in that path. The above transformation indicates that the *Stop* or *Cancel* CFSs have the same semantics, except that a Stop CFS can be defined in a specific point of the protocol.

In Figure 3, lines 52-56 and 89-90 show the activities generated from the CFS with the operator *Xor* of the protocol of Figure 2.

4.2.3 Transformation of Terminations

An interaction protocol can have an implicit termination, such as the WS-CDL choreographies. Hence, it is not necessary to generate any WS-CDL element if the protocol has an implicit termination. Also, an interaction protocol can have explicit terminations: *success* or *failure*. However, there is not a corresponding construct in WS-CDL that represents the semantics of these terminations. Moreover, WS-CDL does not provide a construct to define terminations in a specific point of the choreography, such as the used in ebXML BPSS (*success* and *failure* states) and BPEL (*terminate* activity). To represent an explicit termination in WS-CDL, we have to add a condition in the *complete* attribute of the choreography. A termination variable is added to the choreography and available to both roles. After the activity where the termination should occur, an *assign* activity is added to set to "true" the value of the termination variable in order to the expression of the choreography's *complete* attribute evaluates to true and the choreography completes. A termination variable can be added to the choreography to represent a success termination and another one to represent a failure termination. In Figure 3, lines 32-35 and lines 82-88 show an example.

In addition, a WS-CDL *finalizerBlock* does not have a correspondence with any modeling element of UP-ColBPIP, therefore it cannot be derived from interaction protocols. However, this construct can only be used in non-root choreographies and it is not clear when a successfully completed choreography can require further action. Due to finalizer blocks cannot be generated, *finalize* activities are also not used. Furthermore, since in an interaction protocol the roles agree on whether it completes successfully, with a failure or with an exception occurrence (like in a WS-CDL coordinated choreography), finalizer blocks are not required.

4.2.4 Transformation of Interaction Occurrences

An interaction occurrence is transformed into a *perform* activity, which allows the definition of nested choreographies. If the interaction occurrence is defined within a CFS, an expression should be generated for the *choreographyInstanceId* attribute of the perform activity representing the interaction occurrence. The *block* attribute always is set to "true", because in UP-ColBPIP a protocol must wait for the termination of its nested protocols in order to continue after their execution. The business documents of an interaction protocol can be correlated with the business documents defined into the nested protocol. Correlations are defined with a comment stereotyped *Correlation* associated to the interaction occurrence. Correlations are mapped into bindings between the variables (representing the business documents) of the enclosing choreography (the protocol) and the enclosed choreography (the nested protocol). In WS-CDL, correlations are called binding and they are defined with the *bind* element.

For example, in Figure 3, lines 91-92 show the perform activity representing the interaction occurrence *Collaborative Demand Forecast* of the protocol of Figure 2.

4 Related Work

There are several Model-Driven Development (MDD) approaches proposed to generate technological solutions based on web services composition [2,6,10]. They focus on the modeling and automatic code generation of behavioral interfaces. Hence, they support the modeling and specification of the behavior of collaborative processes but only from the point of view of one partner. As a result they do not support the global view of the interactions, which is required in the modeling of collaborative processes and the specification of web services choreographies.

In [11], transformation of WS-CDL choreographies into BPEL abstract processes is defined. This approach starts from WS-CDL choreography definitions and it does not consider the use of a graphical modeling language to define choreographies or collaborative processes. However, according to MDA principles, a language to define technology-independent models should be provided, as it is proposed in this paper.

In [8, 17], choreographies are generated from collaborative process models. They focus on the generation of choreographies based on the language ebXML BPSS, which is based on business transactions instead of using web services. In [8], the language UN/CEFACT Modeling Methodology (UMM) has been used and validated to generate ebXML BPSS specifications. In [17], we used and validated UP-CoLBPIP to generate ebXML BPSS specifications. The implementation of transformation rules were carried out by using a model transformation tool we are developing. However, the transformations presented in this work have not been implemented yet in that tool.

Furthermore, other approaches focus on the generation of the behavioral interfaces of partners from collaborative process models [5] [19]. The target language used is BPEL. These approaches focus on the definition of behavioral interfaces in the technological level while in the business level the focus is on the collaborative process models. Hence, the use of choreographies in the technological level is not considered. This is possible because a web service choreography cannot be implemented in a direct way but it is implemented through the definition of behavioral interfaces and orchestrations for each partner. This is one of the arguments against the use of a XML syntax to define choreographies [3] because they are more useful as design artifact than an implementation artifact. However, collaborative process models and choreographies can be used as design artifact. The former should capture the business aspects and the later should capture the technical aspects according to the target technology.

Finally, we are not aware of other MDA approaches that focus on the generation of WS-CDL choreographies from collaborative process models. In the MDA approach proposed in this paper, the objective is to support the definition of technology-independent collaborative processes in a high abstraction level. This is different to the idea of generating graphical notations for technology-specific languages such as WS-CDL. We consider that the main artifacts of the development in B2B collaboration should be the collaborative process models. However, the use of standards is required in order to partners' systems can interoperate and implement collaborative processes. Currently there are different standards, new versions of them can appear and new standards can be proposed. Therefore a technology-independent modeling language for collaborative processes is required and should provide the main conceptual elements to define theses processes in a business level and to generate processes specifications based on the different standards.

5 Conclusions and Future Work

In this paper we have described how web services choreographies based on WS-CDL can be generated from collaborative process models defined with the modeling language UP-ColBPIP, according to the principles of a MDA approach for collaborative processes. The main advantage of using this MDA approach is that it guarantees that web services choreographies generated in the technological level fulfill the collaborative processes defined in a business level. The transformation rules proposed in this paper reduce the risk of inconsistence between collaborative process models and their corresponding web services choreographies based on WS-CDL.

The use of a UML Profile for modeling collaborative processes in a business level allows partners focus mainly on the business aspects of the B2B collaboration. Also, the use of interaction protocols to model collaborative processes supports the requirements of enterprise autonomy, decentralization and peer-to-peer interactions, as well as it supports business aspects such as the definition of negotiations and commitments through the use of speech acts. These business aspects are not captured in standards such as WS-CDL, which focus mainly on the technical aspects of the collaborative processes based on web services technology. However, WS-CDL could be enriched with conceptual elements more oriented to business aspects such as speech acts. Also, UP-ColBPIP could also be enriched with several characteristics provided by WS-CDL, such as the concepts of channel, tokens and finalizer blocks.

As result of the transformation rules of a UP-ColBPIP model into a WS-CDL specification, most of the conceptual elements provided by UP-ColBPIP can be represented in WS-CDL. Although WS-CDL only provides the basic control flow activities, most of the control flow operators used in UP-ColBPIP can be represented in WS-CDL. However, multiple instances of interactions and explicit terminations of the choreography are not supported in WS-CDL.

An open issue is to determine if WS-CDL specifications generated from UP-ColBPIP models are well-formed. But we cannot to determine if a WS-CDL specification is well-formed, as well as it is not possible to determine yet this with UP-ColBPIP models. To guarantee correctness of these models and specifications a formalized model has to be generated. We are working on the formalization of UP-ColBPIP models using Petri Nets. The purpose is to enable the verification of collaborative processes, previous to the generation of B2B specifications.

References

1. Baghdadi, Y.: ABBA: An architecture for deploying business-to-business electronic commerce applications. Electronic Commerce Research and Applications, 3(2004) 190-212
2. Baïna, K, Benatallah, B., Cassati, F., Toumani, F.: Model-Driven Web Service Development. CaiSE'04, Springer (2004) 290-306.
3. Barros, A., Dumas, M., Oaks, P.: A Critical Overview of the Web Service Choreography Description Language (WS-CDL). BPTrends Newsletter 3 (2005).
4. BEA, IBM, Microsoft, SAP, Siebel: Business Process Execution Language for Web Services. http://www-106.ibm.com/developerworks/library/ws-bpel/, 2003.
5. Bruno, G. and La Rosa, M.: From Collaboration Models to BPEL processes through service models. BPM Workshops 2005, WSCOBPM 2005, 75-88, 2005.
6. Gardner, T.: UML Modelling of Automated Business Processes with a Mapping to BPEL4WS, 17th ECOOP, Darmstadt, Germany (2003).
7. Goldkuhl, G., Lind,M.: Developing E-Interactions – a Framework for Business Capabilities and Exchanges. Proceedings of the ECIS-2004, Finland, 2004.
8. Hofreiter B., Huemer C.: ebXML Business Processes - Defined both in UMM and BPSS. Proc. of the 1st GI-Workshop XML Interchange Formats for Business Process Management, Modellierung 2004, Germany, 81-102, 2004.
9. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y.: Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation (2005), W3C.
10. Koehler, J., Hauser, R., Kaporr, S., Wu, F., Kurmaran, S.: A Model-Driven Transformation Method. 7th International Enterprise Distributed Object Computing, 2003.
11. Mendling, J. and Hafner, M.: From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL. OTM 2005 Workshops. LNCS 3762,506-515, 2005.
12. Object Management Group: MDA Guide V1.0.1, 2003. http://www.omg.org/mda.
13. Searle, J.R.: Speech Acts, an Essay in the Philosophy of Language, Cambridge University Press, Cambridge, 1969.
14. UN/CEFACT and OASIS: ebXML Business Specification Schema Version 1.10, http://www.untmg.org/downloads/General/approved/ebBPSS-v1pt10.zip (2001)

15. Villarreal, P.: Method for the Modeling and Specification of Collaborative Business Processes. PhD Thesis. Universidad Tecnológica Nacional, Santa Fe, Argentina (2005).
16. Villarreal, P., Salomone, H.E. and Chiotti, O.: B2B Relationships: Defining Public Business Processes using Interaction Protocols. Journal of the Chilean Society of Computer Science, Special issue on the Best Papers of the JCC 2003, Vol. 4(1) (2003).
17. Villarreal, P., Salomone, H.E. and Chiotti, O.: Applying Model-Driven Development to Collaborative Business Processes. Proceedings of the 8th Ibero-American Workshop of Requirements Engineering and Software Environments, Chile, 2005.
18. Villarreal, P., Salomone, H.E. and Chiotti, O.: Modeling and Specifications of Collaborative Business Processes using a MDA Approach and a UML Profile. In: Rittgen, P. (eds): Enterprise Modeling and Computing with UML. Idea Group Inc, (in press).
19. Villarreal, P., Salomone, H.E. and Chiotti, O.: MDA Approach for Collaborative Business Processes: Generating Technological Solutions based on Web Services Composition. Proceedings of the 9th Ibero-American Workshop of Requirements Engineering and Software Environments, Argentine, 2006, in press.