MDA Approach for Collaborative Business Processes: Generating Technological Solutions based on Web Services Composition

Pablo David Villarreal1, Enrique Salomone1,2, Omar Chiotti1,2

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

Abstract. In order to establish B2B Collaborations with their partners, enterprises have to focus on the modeling and specification of collaborative business processes and partners’ interfaces that compose the B2B information system. Currently, standards for web services composition are having more and more interest to specify and implement these processes and interfaces. In this paper, we propose a MDA approach to support the modeling of collaborative processes and the automatic generation of technological solutions based on web services composition. Particularly, we discuss the generation of specifications based on the BPEL standard, from collaborative process models defined with the language UP-ColBPIP. This language supports the conceptual modeling of technology-independent collaborative processes as interaction protocols. The rules to transform UP-ColBPIP models into BPEL specifications are defined.

1. Introduction

Currently, enterprises are focusing on the setting up of Business-to-Business (B2B) Collaborations with their partners. Two abstraction levels have to be considered in these B2B collaborations: a business level and a technological level [15,18].

In the business level, partners have to focus on the definition of collaborative business processes, which describe the explicit behavior of the collaboration along with the partners’ responsibilities. Through the definition of these processes, partners undertake to jointly carry out decisions to achieve common goals, coordinate their actions and exchange information. In this level, collaborative processes have to be defined in an independent way of the implementation technology, because business analysts and system designers do not want to deal with the technical details.

In the technological level, enterprises have to focus on the implementation of the B2B information system that supports the collaborative processes. This system is made up of autonomous, heterogeneous and distributed components owned by enterprises to support the joint execution of collaborative processes. Interoperability between these components can be achieved by using B2B protocols [5] [6]. Two important interoperability aspects are supported by these protocols: the specification
of the collaborative processes in an executable language and the specification of the partners’ interfaces that compose the B2B information system. Currently, there are two main technologies proposed for the specification of B2B protocols [5]: standards based on Web Services Composition, such as BPEL (Business Process Execution Language) [4] and WS-CDL (Web Services Choreography Description Language) [10]; and standards based on Business Transactions, such as ebXML [14].

Hence, collaborative processes must be defined first in a business level using a technology-independent language and then in a technological level using a B2B standard. However, the development of these B2B solutions is complex and costly [1]. In addition, both definitions must have a mutual correspondence [5].

The Model-Driven Architecture (MDA) Initiative [12] has been identified as a key enabler to support the development of collaborative processes [18]. In this way, we proposed a method for collaborative processes [15] based on the principles of MDA. This MDA approach supports: the design of collaborative processes independent of the idiosyncrasies of particular B2B standards; and the automatic generation of specifications based on a B2B standard from conceptual collaborative process models. The main benefits of this approach are: (1) increase of the abstraction level because the main artifacts of the development are the technology-independent collaborative process models; (2) reduction of development time and costs because solutions are generated automatically from conceptual models of collaborative processes; (3) the generated technological solutions are consistent with the processes defined in the business level; (4) independence of the collaborative process models from the B2B standards, which increases the reuse of these models.

In previous work, we described the generation of technological solutions based on ebXML [18] using a MDA approach. However, web services are having more and more interest to implement B2B information systems. In addition, currently vendor support is stronger for web services than ebXML.

In this paper, we focus on the application of a MDA approach for the generation of technological solutions based on web services composition to implement collaborative processes. In particular, we use the BPEL standard. BPEL supports the specification of collaborative processes only from the point of view of one partner. However, an important requirement in the collaborative process modeling and specification is the representation of the global view of the interactions between the partners [18]. Hence, to generate solutions based on BPEL, a BPEL process for each partner has to be specified. Moreover, BPEL processes have to be consistent to each other to represent the semantics of a collaborative process. In this paper, we show how this consistence can be achieved through the use of a MDA approach. It allows BPEL processes of each partner can be automatically derived from the common collaborative process models defined using the language UP-ColBPIP [15,17,18].

This paper is organized as follows. Section 2 describes the MDA approach we propose for collaborative processes. Section 3 briefly describes the modeling language UP-ColBPIP used to model technology-independent collaborative processes. Section 4 gives an overview of BPEL and describes the transformation of UP-ColBPIP models into BPEL and WSDL specifications. Section 5 discusses related work. Section 6 presents conclusions and outlines future research directions.
2. An MDA Approach for Collaborative Business Processes

The MDA initiative [12] proposes a conceptual framework along with a set of standards (UML, MOF, XML, etc.) to build model-driven development methods. One principle of MDA is the building of systems can be organized around a set of models by imposing series of transformations between models, organized into an architectural framework of layers and transformations. In MDA, a 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: the collaborative processes in an executable language and the partners’ interfaces of the B2B information system.

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.

- **BPEL and WSDL models.** BPEL models represent the technology-dependent collaborative process models. Web services composition standards are based on the Web Service Description Language (WSDL), which is used to define web services of the partners. WSDL models represent the technology-dependent partners’ interfaces. To build these models we define their corresponding metamodels, which can be derived from the XML schemas provided by these B2B 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 to BPEL and WSDL models.** These transformations define a set of transformation rules to allow the generation of
BPEL process models from UP-ColBPIP models. They define the correspondence between the UP-ColBPIP concepts and the BPEL and WSDL concepts. Then, these transformation rules are implemented through a method and a tool for model transformations, which support the definition and automatic execution of the rules.

**BPEL and WSDL Specifications.** The final outputs of the transformations are the XML documents of the BPEL process specifications and the WSDL partners’ interfaces specifications. The transformation of technology-dependent models into the corresponding specifications is almost direct. This can be supported through XML production rules that convert a UML object model into a XML document.

In this work, we focus on UP-ColBPIP models and the definition, in a conceptual way, of the transformations of UP-ColBPIP to BPEL and WSDL. The other techniques and components are out of scope of this paper. They can be found in [15].

### 3. The Modeling Language UP-ColBPIP

UP-ColBPIP is a modeling language to design technology-independent collaborative processes [15,17,18]. It encourages a top-down approach and provides the conceptual elements to support the modeling of four views for collaborative processes:

- **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 services and interfaces required by partners to support the message exchange 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 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 to realize the behavior of the collaborative processes. In the domain of B2B collaborations, an interaction protocol describes a high-level communication pattern through a choreography of business messages between partners playing different roles [16]. The main objective of modeling collaborative processes through 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 process models, process models based on interaction protocols focus on the message exchange between enterprises. 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.
Modeling interaction protocols, the focus is on the representation of the global view of the interactions between the partners. The message choreography describes the peer-to-peer interactions and partners’ responsibilities of the roles they fulfill.

In addition, B2B interactions cannot be restricted to mere information transfer [8]. Instead, it is necessary to represent the communication and the creation of commitments between the partners. These communicative aspects can be represented through interaction protocols, because they are based on 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. In this way, designers of a collaborative process focus on the semantics of each message selecting the most appropriate speech act. Furthermore, decisions and commitments done by the partners can be known from the speech acts. This enables the definition of negotiations in collaborative processes.

UP-ColBP 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 example we describe the interaction protocol Demand Forecast Request, which supports a simplified collaborative demand forecast process between two partners.

3.2.1. Interaction Protocol. Figure 2 shows the sequence diagram of the Demand Forecast Request protocol. It is stereotyped <<protocol>>.

3.2.2. Role and Partner. The partners and their roles they fulfill in an interaction protocol are represented through the lifelines. In the protocol of Figure 2, the partner “A” play the role supplier, and the partner “B” play the role customer.

3.2.3. Business Message. It defines an interaction between two roles, a sender and a receiver. A business message contains a speech act and a business document. The semantics of a business message is defined by its speech act associated. 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 the receptor then acts according to the semantics of the speech act. As example, for the message request(DemandForecast), the associated speech act indicates the supplier’s intention of requesting to the customer a demand forecast. The speech act request also implies the customer cannot respond with any speech act but a suitable speech act, such as agree, or refuse. The suitable speech acts depend on the speech act library used to define the messages of the protocol and on the semantics of the speech acts.

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

3.2.4. Control Flow Segment. It represents complex message sequences in the choreography of an interaction protocol. It contains a control flow operator and one or more interaction paths. An interaction path (interaction operand in UML2) can contain any protocol element, such as: 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. It is represented as a box with a rounded-rectangle in the upper-left corner that contains a control flow operator.

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. Each path has a condition. This segments indicates the customer can respond of two different way: it accepts the supplier’s request and commits to carry out the demand forecast in the future (message agree(ResponseToForecastRequest)); or it rejects the supplier’s request. In this case the protocol ends.

Fig. 2. An Example of an Interaction Protocol

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. Each path has a condition. This segments indicates the customer can respond of two different way: it accepts the supplier’s request and commits to carry out the demand forecast in the future (message agree(ResponseToForecastRequest)); or it rejects the supplier’s request. In this case the protocol ends.
3.2.5. **Conditions and Time Constraints.** Conditions represent logical expressions that constraint the execution of a message or a path into a control flow segment. They can be defined in natural language or using OCL expressions. Time constraints are used to define duration or deadlines on messages or protocols. A time constraint can be defined using relative or absolute dates. In the protocol of Figure 2, the messages of the control flow segment have a relative time constraint, which indicates these messages have to be sent before two days, then of the occurrence of the first message.

3.2.6. **Interaction Occurrence.** It represents the invocation of another interaction protocol, which is referred as the nested protocol. In Figure 2, if the customer agrees on the supplier’s request, the protocol continues with the invocation of the nested protocol **Collaborative Demand Forecast**.

3.2.6. **Termination.** It represents an explicit end of a protocol. Two termination types can be defined: success and failure. Success implies the protocol has ended in a successful way. Failure indicates the protocol has not ended as it was expected. This only indicates the protocol did not follow the expected business logic. In Figure 2, if the customer rejects the supplier request, the protocol ends with a failure. Else, then of invoking to the nested protocol, the protocol finishes in an implicit way.

4. Generating BPEL specifications from UP-ColBPPIP Models

In this section we discuss how models of collaborative processes defined with UP-ColBPPIP can be used to generate BPEL specifications that enable the execution of these processes. First, we describe the transformations of UP-ColBPPIP concepts into WSDL concepts. Then, we give an overview of the BPEL standard. Finally, we describe the transformations of UP-ColBPPIP concepts into BPEL concepts.

4.2. **Transformation of UP-ColBPPIP to WSDL**

In UP-ColBPPIP, a role has a public business interface, which contains its required and provided business interfaces. These interfaces contain the business services that support the sending and receiving of business messages. These elements define the Business Interfaces View of a UP-ColBPPIP model [15].

Hence, a WSDL model is generated for each role defined in a UP-ColBPPIP model, because WSDL interfaces of the partners are derived in a direct way from the roles’ provided business interfaces. The number of UP-ColBPPIP to WSDL transformations to be done is equal to the number of partners defined in the UP-ColBPPIP model.

Table 1 shows the correspondence between the concepts. Due to in UP-ColBPPIP the business messages are asynchronous, WSDL operations are defined in an asynchronous way with an input message. Request/response operations are not generated. A business document is transformed into a message in WSDL.

Next code shows a fragment of the WSDL document that contains the interface of the role **customer**. As example, operation **request_ForecastDemand** was defined on the customer’s port type to support the reception of the first message of the protocol.
<wsdl:definitions ...>
<message name="Msg:DemandForecast"/>
<message name="Msg:ResponseToForecastRequest"/>
<portType name="PT:Customer_InterfaceWithSupplier">
  <operation name="agree_DemandForecast">
    <input message="Msg:DemandForecast"/>
  </operation>
</portType>

Table 1. Transformation of UP-ColBPIP concepts into WSDL concepts

<table>
<thead>
<tr>
<th>UP-ColBPIP Concept</th>
<th>WSDL Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role</td>
<td>Service</td>
</tr>
<tr>
<td>PublicBusinessInterface</td>
<td>PortType</td>
</tr>
<tr>
<td>BusinessService (from the Provided Interface)</td>
<td>Operation</td>
</tr>
<tr>
<td>BusinessDocument</td>
<td>Message</td>
</tr>
</tbody>
</table>

4.1. Overview of BPEL

The Business Process Execution Language (BPEL) [4] is an XML-based language intended to specify business processes in terms of a control flow that involves operations provided by one or several web services. It supports the definition of abstract and executable processes. An abstract process defines a business protocol, also known as conversation protocol [2]. This type of protocol defines the order in which a partner sends messages to and receives messages from its partners to achieve a business goal. An executable process defines the service business logic that supports the behavior of a partner in the interaction with other partners. This internal logic involves the manipulation of the partner’s private data, the rules that determines the alternative paths, and the invocations to operations of private web services.

An activity in a BPEL process is either primitive or structured. Primitive activities are: invoke, to invoke a web service operation; receive, to wait for the reception of a message; reply, to reply to a message; wait, to remain idle for a given time period; throw, to raise exception errors; assign, to copy data from one variable to another; and empty, to do nothing. Structured activities are defined using the following control flow constructs: sequence, to define a sequential order between activities; pick, for non-deterministic alternatives based on events, such as message reception or time alarm; switch, to represent conditional routing; flow, to represent parallel execution routing and it is also used to define dependences between activities through the use of control links, in which the target activity of a control link may only start when the source activity has ended; while, to represent iterations; and scope, to group activities to be treated by the same fault-handler and within a given transactional context.

The above BPEL concepts are used to define both executable and abstract processes. An executable process can be considered as a refinement of an abstract process and it contains a mix of private and public aspects of a partner.

4.3. Transformation of UP-ColBPIP to BPEL

An important feature of BPEL is both abstract and executable processes are defined from the point of view of one partner. Figure 3 represents the correspondence
between the process concepts we proposed in the business level and those proposed by BPEL in the technological level. The semantics of a collaborative process based on interaction protocol is represented by means of BPEL abstract processes, one by each partner involved in the protocol. This means that a BPEL abstract process represents the behavior of the role fulfilled by one partner in a collaborative process.

Several BPEL models have to be generated from a UP-ColBPIP model. On one hand, a BPEL model specifies just one process, whereas a UP-ColBPIP model can have a number of interaction protocols. On the other hand, an abstract process models have to be generated for each role involved in an interaction protocol. Therefore, the number BPEL abstract process to be generated from a UP-ColBPIP model is defined as follow: \( AP = \sum_{ip} R_{ip} \). \( NoIP \) is the number of interaction protocols of the UP-ColBPIP model and \( R_{ip} \) is the number of roles involved in each \( ip \) protocol.

In BPEL is necessary to indicate what WSDL interfaces of the other partners will be used in the process. This is done defining the partnerLinkType before the definition of the BPEL processes. As example, next code shows the partner link types.

```xml
<partnerLinkType name="PLT:CustomerSupplier">
  <role name="Customer">
    <portType name="PT:Customer_InterfaceWithSupplier"/>
  </role>
  <role name="Supplier">
    <portType name="PT:Supplier_InterfaceWithCustomer"/>
  </role>
</partnerLinkType>
```

### 4.3.1. Transformation of Interaction Protocols

As we discussed above, for each role involved in an interaction protocol, a BPEL abstract process is generated. Therefore, a BPEL abstract process can be considered as the implementation of the lifeline of the role fulfilled by a partner in an interaction protocol. The partner with its role represented by an abstract process is referred as the process owner. At the beginning of a process, partner links have to be defined to indicate the relationship between the process owner and its partners. A partnerLink indicates the partnerLinkType, the role of the process owner (attribute myRole), and the roles of the partners (attribute partnerRole).
As example, the next code shows the BPEL abstract process of the role *customer* fulfilled by a partner in the interaction protocol *Demand Forecast Request*. Also, it shows the partner links defined in this process. Due to space limitations we do not detail the peer process corresponding to the role *supplier*.

```xml
<process name="DemandForecastRequest_Customer" .... abstractProcess="yes">
   <partnerLinks>
      <partnerLink name="LinkToSupplier" partnerLinkType="PLT:CustomerSupplier"
                    myRole="Customer" partnerRole="Supplier"/>
   </partnerLinks>
</process>
```

### 4.3.2. Transformation of Business Messages.

They can be transformed into three types of activities: *receive* or *pick*, and *invoke*. When a business message is sent by the process owner, an activity *invoke* is generated with the corresponding operation defined in the port type of the role that receives the message.

When a business message is received by the process owner and it has not an associated time constraint, it is transformed in an activity *receive*. This activity contains the operation through the process owner receives information. The operation is part of the port type of the process owner.

When a business message is received by the process owner and does not have an associated time constraint, it is transformed in an activity *pick*. This activity is defined with two events: *onMessage*, which indicates the reception of the message through the corresponding operation of the process owner port type; and *onAlarm*, which is used to represent the time constraint. Only one of these events can occur. So, if the message is received, the *onMessage* event occurs. Else, if the message is not received within the deadline defined in the *onAlarm* event, this event occurs and the actions defined in it are executed. If the time exception occurs in a process, it has to be informed to the partner. To do that, in the port types of the partner, an operation *timeException* is defined. It is invoked to inform the exception. The invocation is done through an activity *invoke* defined in the event *onAlarm*.

As example, the next code shows the activities generated in the supplier’s process, from the *request* and *agree* messages of the protocol *Demand Forecast Request*.

```xml
<invoke partnerLink="LinkToCustomer"
         portType="PT:Customer_InterfaceWithSupplier"
         operation="request_DemandForecast" inputVariable="DemandForecast"/>

<pick>
   <onMessage partnerLink="LinkToCustomer"
              portType="PT:Supplier_InterfaceWithCustomer"
              operation="agree_ResponseToForecastReceipt"
              variable="ResponseToForecastReceipt"/>
   <onAlarm for="t..t+2"
            portType="PT:Customer_InterfaceWithSupplier" operation="timeException">
      <terminate/>
   </onAlarm>
</pick>
```

### 4.3.3. Synchronizing the Start of a BPEL Process.

A BPEL process starts with an activity *receive* or *pick*, with its attribute *createInstance* settled to “yes”. The process starts when the operation defined in this activity is invoked. However, it is necessary to synchronize the beginning of BPEL processes representing an interaction protocol. This is due to in BPEL there is not way to correlate instances of different processes.
To represent this synchronization, a “start” operation is added for each collaborative process in the port type of each partner. In the example, we generate the operation start_DemandForecastRequest in the port types of the customer and supplier. Then, it is necessary to identify the initiator of the protocol. In the BPEL process of the initiator, we add an activity receive with the operation start_DemandForecastRequest. This operation will be invoked by the initiator from an internal application, when it requires starting the collaborative process. In addition, an activity invoke is added, which contains the “start” operation defined in the port type of the other partner. This activity enables the instantiation of the process of the other partner. Therefore, in the process of this partner, an activity receive with the corresponding “start” operation is added.

Next code shows the start of the supplier’s process (the initiator of the protocol).

```xml
<receive partnerLink="LinkToSupplier"
    portType="PT:Supplier_InterfaceWithCustomer"
    operation="start_DemandForecastRequest"
    inputVariable="ProtocolInstanceNumber"/>
<invoke partnerLink="LinkToCustomer"
    portType="PT:Customer_InterfaceWithSupplier"
    operation="start_DemandForecastRequest"
    inputVariable="ProtocolInstanceNumber"/>
```

The next code shows the start of the customer’s process.

```xml
<receive partnerLink="LinkToSupplier"
    portType="PT:Customer_InterfaceWithSupplier"
    operation="start_DemandForecastRequest"
    inputVariable="ProtocolInstanceNumber"/>
```

### 4.3.4. Transformation of Control Flow Segments

They are transformed into structured activities, according to the operator used (Summarized in Table 2).

BPEL does not support distributed transactions among several processes. Hence, a segment with the operator Transaction cannot be represented in BPEL. This problem is recognized in BPEL and the WS-Transaction language is proposed for the definition of these transactions [4]. Also, a segment with the Cancel operator does not have a counterpart in BPEL, although it could be implemented through fault handlers that have to be synchronized with the fault handlers of the partners’ processes.

The next code shows the transformation of the control flow segment of the interaction protocol Demand Forecast Request into the customer’s process.

```xml
<switch>
    <case condition="bpws:getVariableData('RequestAccepted')=TRUE">
        <sequence> <!-- Code of the first interaction path -->
    </sequence>
    </case>
    <case condition="bpws:getVariableData('RequestRejected')=TRUE">
        <sequence> <!-- Code of the second interaction path -->
    </sequence>
    </case>
</switch>
```

### 4.3.5. Transformation of Terminations

The implicit and explicit termination of a protocol has also to be represented taking into account the synchronization of the BPEL processes of the partners, in a similar way to the synchronization of the start of these processes. Therefore, in the port type of the partners the operation
end_DemandForecastRequest is added. Then, in each BPEL process, an event handler is added indicating the possible reception of a message through the “end” operation.

Table 2. Transformation of Control Flow Segment into BPEL Structured Activities

<table>
<thead>
<tr>
<th>Control Flow Segment</th>
<th>BPEL Structured Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xor</td>
<td>Switch (One case for each interaction path)</td>
</tr>
<tr>
<td>Or</td>
<td>Flow (Activity dependencies using links with transition and join conditions, and activities empty)</td>
</tr>
<tr>
<td>And</td>
<td>Flow</td>
</tr>
<tr>
<td>Loop (0,n); Loop (1,n)</td>
<td>While (BPEL does not differentiate types of loop)</td>
</tr>
<tr>
<td>If</td>
<td>Switch (One case and optionally an otherwise)</td>
</tr>
<tr>
<td>Exception</td>
<td>Switch</td>
</tr>
<tr>
<td>Stop</td>
<td>Switch, followed by activities defining the process termination</td>
</tr>
</tbody>
</table>

Next code shows the failure termination of the protocol Demand Forecast Request in the customer’s process. It has an activity invoke that informs to the supplier’s process the termination. Then, the activity terminate ends the customer’s process.

```xml
<invoke partnerLink="LinkToSupplier"
  portType="PT:Supplier_InterfaceWithSupplier"
  operation="end_DemandForecastRequest"
  inputVariable="ProtocolInstanceNumber"/>
<terminate/>
```

Next code denotes the counterpart required in the supplier’s process that finishes it.

```xml
<eventHandlers>
  <onMessage partnerLink="LinkToCustomer"
    portType="Supplier_InterfaceWithCustomer"
    operation="end_BlanketPurchaseOrder">
    <terminate/>
  </onMessage>
  ...
</eventHandlers>
```

5. Related Work

MDD approaches have been proposed to generate technological solutions based on web services composition [2, 7, 11]. They focus on the modeling and automatic code generation of business or conversation protocols. Therefore, 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 a global view in the modeling of collaborative processes; and they do not intend to support the synchronization and the progress of the process instances of each partner.

Moreover, some MDD approaches are not based on a standard conceptual framework, such as MDA [2]. Other approaches do not consider all the components required in a MDA approach. In [7], a UML Profile is provided for the modeling of technology-dependent models, but it does not support technology-independent process models. As a result, this approach influences the adoption of one standard.

In [9], the transformation of collaborative processes defined with the UN/CEFACT Modeling Methodology (UMM) to BPEL specifications has been proposed. This
approach does not consider the application of a MDA approach to support the development of collaborative processes. This is the main difference with our approach. In addition, although from a UMM business collaboration, two or more BPEL processes are generated, this approach does not generate the extra XML code necessary for the synchronization of the BPEL processes’ instances of each partner.

Finally, in the context of web services composition standards, the specification of the global view of the interactions between the partners is only supported by the WS-CDL language [10]. However, WS-CDL is not an executable language and it is expected to be used in conjunction with BPEL. In this approach, partners use WS-CDL to agree on the choreography of messages to achieve common goals. This choreography then is used to generate BPEL abstract processes for each partner. However, there are two important limitations of the above approach. On one hand, WS-CDL is a technology-specific language and it does not provide a graphical notation to define services choreographies. As it is highlighted in [3], global service choreographies are more a design than an implementation artifact, and hence, a graphical modeling language is more useful than a XML syntax. In addition, collaborative processes should be defined independent of the technology [18]. On the other hand, it remains open how WS-CDL specifications can be mapped into BPEL and hence, it is not clear if all WS-CDL concepts can be mapped into BPEL [3].

In contrast to above approaches, through the modeling of interaction protocols, we focus on a whole view of the choreography of messages exchanged between partners. Also, BPEL abstract processes are derived from interaction protocols. This ensures BPEL processes of each partner, which implement an interaction protocol, are consistent between them and can be synchronized in run-time by partners’ systems.

6. Conclusions and Future Work

In this work, we have proposed a MDA approach for the conceptual modeling of collaborative business processes and the generation of technological solutions based on web services composition. In particular, we have described how it is possible to implement collaborative processes using the BPEL standard. The main contribution of this approach is the generated solutions based on web services composition are consistent with the UP-ColBPIP models defined in the business level. Thus, it allows decreasing the complexity, time and costs in the generation of this solutions.

In this MDA approach, we use the UP-ColBPIP language for the conceptual modeling of collaborative processes. The use of interaction protocols supports several important aspects of the B2B collaborations: enterprise autonomy, decentralization, peer-to-peer interactions and negotiations. Mainly, interaction protocols allow describing the commonly agreed global view of the interactions between the partners.

BPEL supports the definition of collaborative processes from the point of view of one partner. Therefore, the semantics of a collaborative process based on interaction protocol have to be represented in the technological level through several BPEL abstract processes, one by each partner. This adds an extra complexity to the generation of solutions based on web services composition, because BPEL processes representing an interaction protocol have to be consistent to each other. Through the
proposed MDA approach, this consistence is achieved due to BPEL processes are
generated from the common view defined in the interaction protocols.

In addition, as part of this MDA approach we have defined the transformations of
UP-ColBPIP models into BPEL specifications. They define the correspondence
between the concepts of both languages. As result of these transformations, on one
hand, most of the UP-ColBPIP concepts can be represented in BPEL. On the other
hand, BPEL does not provide concepts to define the synchronization of two different
abstract processes. Therefore, extra code, which does not belong to the process logic,
has to be added to support the synchronization of the processes’ instances
representing an interaction protocol instance.

Our ongoing work is about the implementation of these transformations through a
method and a tool that we have developed for model transformations. The objective
of this implementation is the execution and generation in an automatic way of the
BPEL and WSDL specifications from UP-ColBPIP models.

References
4. BEA, IBM, Microsoft, SAP, Siebel: Business Process Execution Language for Web
Record, Special Issue on Data Management Issues in E-Commerce, 31(1) (2002)
10. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y.: Web Services
Choreography Description Language Version 1.0. W3C Working Draft (2004), W3C.
14. UN/CEFACT and OASIS: ebXML Business Specification Schema Version 1.10,
Business Processes based on Interaction Protocols. Argentine Symposium on Information
Collaborative Business Processes. 8th Ibero-American Workshop of Requirements
Engineering and Software Environments, Chile, 2005.