Integrating Quality of Service Aspects in Top-Down Business Process Development using WS-CDL and WS-BPEL

Florian Rosenberg, Christian Enzi, Anton Michlmayr, Christian Platzer, Schahram Dustdar
VitaLab, Distributed Systems Group
Technical University of Vienna
A-1040 Vienna, Argentinierstrasse 8/184-1, Austria
{florian, enzi, anton, platzer, dustdar}@infosys.tuwien.ac.at

Abstract

Developing cross-organizational business processes is a tedious task. The partners have to agree on a common data format and meaning as well as on the Quality of Service (QoS) requirements each partner has to fulfill. The QoS requirements are typically described using Service Level Agreements (SLAs) among the partners. In this paper, we propose a top-down modeling approach for Web service based business processes to capture the functional and non-functional aspects using a choreography language (WS-CDL) which describes the message interactions among the participants. The choreography is annotated with SLAs for the different partners. For each partner in the process, an orchestration (in WS-BPEL) and the necessary Web service templates are automatically generated. Additionally, the Service Level Objectives (SLOs) from the partner SLAs are automatically translated into policies which can then be enforced by a BPEL engine during execution.

1. Introduction

Service-oriented architecture (SOA) represents an emerging paradigm to develop flexible and large-scale software systems using the Internet as the main infrastructure. Web services are one realization of this paradigm by using well-established standards to describe and interact with other services[1]. Web services are "self-describing, open components that support rapid, low-cost composition of distributed applications" [13].

Many organizations are building their cross-organizational business processes based on Web services because of their platform-agnostic nature and the ease of integration. Currently available technologies such as composition engines using the Web Service Business Process Execution Language (WS-BPEL, or BPEL for short) [12] can be used to orchestrate business processes within an organization.

An engineering method for Web service based business processes involving multiple partners requires an agreement on the data that is exchanged which can not be achieved using BPEL. For this purpose, the Web Service Choreography Description Language (WS-CDL) [22] provides a XML-based language to describe the cross-organizational message exchanges from a global viewpoint. The different views (local vs. global) are described by the terms *choreography* and *orchestration*. Choreography can be defined as "processes involving multiple services where the interactions between these services are seen from a global perspective" [22]. A choreography does not describe any internal actions that occur within a participating service, such as internal computation or data transformation, but rather focuses on the observable public exchange of messages. In contrast to that, Peltz [14] defines *orchestration* as an "executable business process that can interact with both internal and external Web services. The interactions occur at the message level. They include business logic and task execution order, and they can span applications and organizations to define a long-lived, transactional, multistep process model."

These two concepts imply that such a choreography description can be used to generate the orchestration behavior (e.g., in the form of BPEL stubs) and the necessary WSDL templates automatically. Nowadays, service level guarantees and obligations among the service providers are becoming increasingly important as a mean to capture runtime quality requirements and guarantees for a partners' service (such as response time, availability as well as security). Such Quality of Service (QoS) requirements are generally specified in a Service Level Agreement (SLA) and need to be fulfilled during the service execution among the partners. Using existing Web service standards and proposals

[1] In this paper we use the term Web service and service interchangeably.

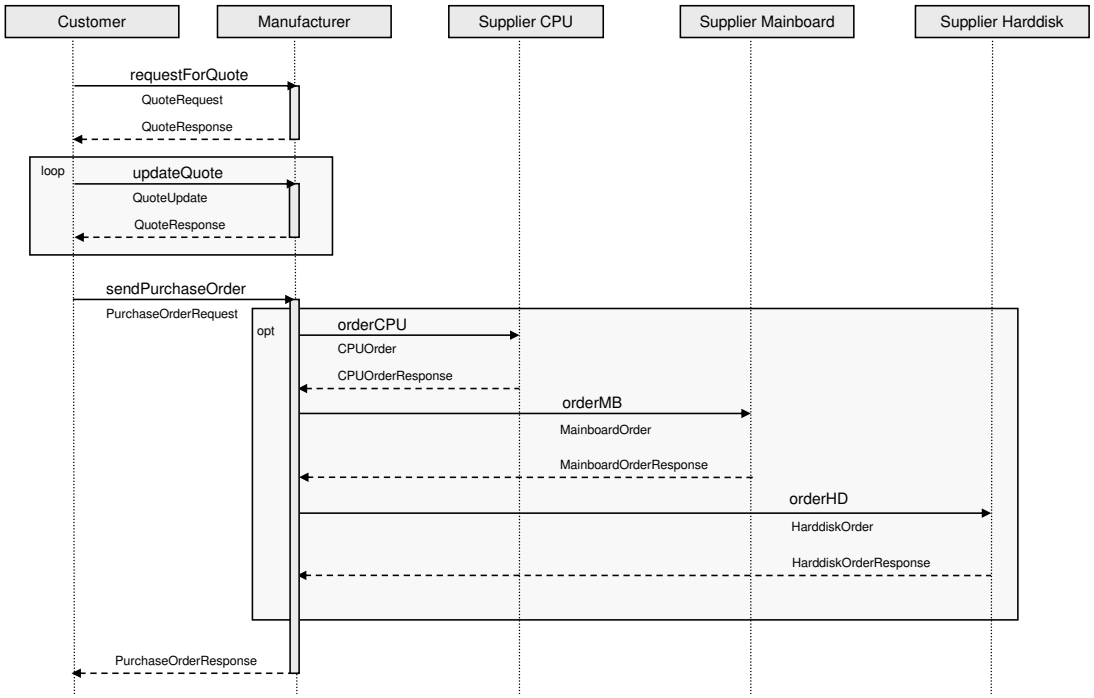

Figure 1. BTO Case Study

there is currently no integrated modeling method available to build such cross-organizational business processes considering SLA requirements as first class citizens from the starting point of the development process. Additionally, the orchestration parts and the WSDL files for each partner in the choreography should be automatically generated.

In this paper, we propose a top-down modeling approach to build such QoS-aware, Web service based business processes using currently available technologies such as WS-CDL and BPEL. It leverages a design approach for efficient development of cross-organizational business processes similar to the idea of model-driven software development (MDSD) [18]. A novelty of this approach is the consideration of SLA requirements from the beginning of the choreography development process. These SLA requirements are then automatically transformed and mapped to a WS-QoS policy and attached to the BPEL process of the affected partner allowing policy-aware middleware to check and enforce the SLA.

This paper is organized as follows: Section 2 describes the case study we implemented for evaluating the concepts of this work. Section 3 introduces the basic concepts of this paper whereas Section 4 describes our main approach for realizing QoS-aware business process development. The implementation of this approach is sketched in Section 5 followed by an evaluation in Section 6. Section 7 positions our approach among existing work and finally, Section 8 concludes the paper.

2. Case Study

In this case study we developed a *Build-to-Order* (BTO) scenario in the B2B area. The use case consists of a customer, a manufacturer, and suppliers for CPUs, main boards and hard disks. The manufacturer offers assembled IT hardware equipment to its customers. For this purpose the manufacturer has implemented a BTO business model. It holds a certain part of the individual hardware components in stock and orders missing components if necessary. In the implemented BTO scenario, the customer sends a quote request with details about the required hardware equipment to the manufacturer. The latter sends a quote response back to the customer. As long as customer and manufacturer do not agree on the quote, this process will be repeated. If a mutual agreement was achieved the customer sends a purchase order to the manufacturer. Depending on its hardware stock the manufacturer has to order the required hardware components from its suppliers. If the manufacturer needs to obtain hardware components to fulfill the purchase order he sends an appropriate hardware order to the respective supplier. In turn the supplier sends a hardware order response to the manufacturer. Finally, the manufacturer sends a purchase order response back to the customer. The interactions of the participants in our BTO scenario are illustrated in the collaboration sequence diagram shown in Figure 1.

The BTO scenario consists of six different Web service invocations which correspond to the following SOAP

operations: requestForQuote, updateQuote, sendPurchaseOrder, orderCPU, orderMB, orderHD. Each SOAP operation depicts a synchronous message request-reply scenario which will be illustrated exemplary for the requestForQuote operation. The customer invokes the operation requestForQuote on the service interface of the manufacturer sending the QuoteRequest message. The manufacturer receives the message request and replies to the service invocation be returning the QuoteResponse message. Contrary to this, an asynchronous message scenario would require additional callback operations on the service requestor side. In this case the manufacturer invokes an operation requestForQuoteCallback on the service interface of the customer to send back the QuoteResponse.

The definition of SLA and QoS plays a crucial rule in cross-organizational business processes. Each participant offers services to other partners over the Internet which the latter need to run their businesses. Therefore, a certain degree of reliability concerning response time, throughput, uptime, etc. is desired and has to be specified and explicitly expressed from the beginning of the modeling phase. In our scenario we distinguish four different relationships between the choreography participants. The customer interacts with the manufacturer, the manufacturer interacts with different suppliers. For each relationship an SLA is defined between the partners to regulate this degree the partners need for their business.

3. Basic Concepts

In this section we introduce the basic concepts and techniques we use in our approach including some illustrating examples.

3.1. An Overview of WS-CDL

WS-CDL represents a non-executable XML-based specification language which allows each involved party to describe its part in the message exchange by specifying details on collaborations, information handling and activities. In the following paragraphs we introduce the basic concepts by using the case study from Section 2 to illustrate some WS-CDL examples.

Collaborations. The collaborations of a choreography are specified by defining participantTypes, roleTypes, relationshipTypes and channelTypes. These declarations define the collaborating participants and their coupling.

A participant type declares an entity playing a particular set of roles in the choreography. Thus a participantType definition contains one or more roleType definitions.

A role type defines a role that enumerates the observable behavior a participant can exhibit in order to interact throughout a message exchange. A roleType definition declares a behavior interface which identifies a WSDL interface type.

The relations between roles are defined through relationshipType definitions. A relationship type always contains exactly two roleTypes, restricting the relationshipType definition to 1:1 relations.

A channel type definition specifies where and how information between participants is exchanged by defining a reference to a role type which is the target of an information exchange (either the receiver of a message request or the sender of a message reply). This role type reference indicates the behavior interface which is used throughout the information exchange.

Information Handling. The definition and handling of information within a choreography is performed by informationTypes and variables.

Information used within a choreography is specified by informationTypes which do not directly reference data types but rather reference type definitions. Such a referenced type definition can be either a WSDL 1.1 Message type, an XML Schema type, a WSDL 2.0 Schema element or an XML Schema element.

Variables capture information about objects in a choreography such as the information exchanged or the observable information of the roleTypes involved and are either bound to informationType or channelType definitions.

Activities. A choreography comprises three different types of activities, namely ordering structures, workunit, and basic activities.

Ordering structures are block structured, enclosing a number of activities or ordering structures which can be used recursively. Such activities include sequence for handling activities in sequential order, parallel for a parallel execution of activities, and choice for handling data or event-driven conditions.

Workunits prescribe the conditional execution of an activity. This conditional execution can either be repetitive (attribute repeat is set to true), competitive (multiple workunit activities are defined inside a choice activity) or blocking (attribute block is set to true). The conditional statement is defined by the attribute guard which specifies a Boolean conditional expression according to the XPath 1.0 lexical rules. In Listing 1, an example with competitive guard conditions from our case study is depicted. If there

are no CPUs in stock they are ordered from the supplier, otherwise available CPUs are selected.

```
<choice>
  <workunit name="Choice_CPUNotInStock"
    guard="cdl:getVariable('CPUNotInStock','','')&gt;0">
    <!— select available CPUs —>
  </workunit>
  <workunit name="Choice_CPUInStock"
    guard="cdl:getVariable('CPUNotInStock','','')=0">
    <!— order CPUs from supplier —>
  </workunit>
</choice>
```

Listing 1. Workunit Example

Basic activities define interactions, actions or variable assignments of the choreography flow. An *interaction* activity defines the information to be exchanged and by what means this information exchange will be performed. The attribute `channelVariable` binds the interaction to a `channelType` and therefore to a specific WSDL interface. The attribute `operation` corresponds to a SOAP operation which is defined throughout this WSDL interface description. The element `participate` defines the requesting and receiving part of the interaction. Finally the element `exchange` defines whether the interaction is a request or response and which variables will be used throughout the message exchange. Listing 2 illustrates an interaction activity which defines a message request from our case study. Throughout the message request the operation `requestForQuote` will be invoked at the corresponding WSDL interface of the `ManRoleType` to request a quote from the manufacturer. The message request is stored in the variable `QuoteRequest`. The response from the `ManRoleType` has to be modeled as another `interaction` (not shown in Listing 2).

```
<interaction channelVariable="tns:QuoteChannelInstance"
  name="RequestForQuote" operation="requestForQuote">
  <participate fromRoleTypeRef="tns:CustRoleType"
    relationshipType="tns:CustMan"
    toRoleTypeRef="tns:ManRoleType"/>
  <exchange action="request" name="request"
    informationType="tns:QuoteRequest" >
    <send variable="cdl:getVariable(
                    'QuoteRequest','','')"/>
    <receive variable="cdl:getVariable(
                    'QuoteRequest','','')"/>
  </exchange>
</interaction>
```

Listing 2. Interaction Activity

The other basic activities include `assign`, `silentAction` and `noAction`. The `assign` activity enables the creation and manipulation of variables within the choreography. The `silentAction` defines a non-observable behavior which is either performed by one or all participants in the choreography. A `silentAction` has to be further defined in the orchestration layer e.g., in the BPEL process of the corresponding participant.

A WS-CDL tool suite from Pi4soa [15] is available to allow the modeling of choreographies without the need to write the XML representation directly. We also used it to model our case study presented in Section 2.

3.2. An Overview of BPEL

BPEL defines a model and grammar for describing the behavior of a business process based on interactions between the process and its partners. A BPEL process defines how multiple service interactions with partners are coordinated to achieve a business goal [12].

Each partner interacting with a BPEL process is defined using a `partnerLink`. Two different roles (`myRole` and `partnerRole`) exist for a partner link to define the sending and receiving side of the process. The basic element in a BPEL process is an activity which come in two flavors, *basic* and *structured* activities. Basic activities mainly define communication primitives for interacting with the partner. For example, `invoke` to invoke a partner service, `receive` to receive a Web service invocation in a synchronized scenario. The `reply` activity is used to send a response message to a previously received Web service invocation message. Other basic activities include `onMessage`, `assign` and `empty`.

Additionally, *structured activities* are similar to control-flow constructs in imperative programming languages. In BPEL, a `sequence` activity is used to execute a given set of activities within a sequence. Parallelism can be achieved by using the `flow` activity. The `while` and `switch` activities are used to represent loops and conditional branches respectively.

The execution of a BPEL process is achieved using an orchestration engine, such as ActiveBPEL [1].

3.3. Service Level Agreements and Policies

In [9] the authors specify that "Service Level Agreements (SLAs) are agreements between a service provider and a service consumer and as such define the obligations of the parties involved." Such obligations are expressed by *Service Level Objectives (SLOs)* on performance and dependability related QoS attributes of Web services. Currently, two different proposals for specifying Service Level Agreements exist, namely WSLA [9] from IBM and WS-Agreement [8] mainly driven by the Grid community.

The WS-Policy family [21] defines an extensible framework to describe capabilities and requirements of services. For instance, using WS-Policy enables to specify if a service requires security or if it supports transactions.

4. Top-Down Modeling Approach

In Section 3, we discussed the concepts of WS-CDL to describe the participants and their message interactions within a choreography. In this section, we give an overview of our top-down modeling approach.

4.1. Overview

As shown in Figure 2, the language constructs of WS-CDL can be mapped to BPEL allowing a choreography description to be transformed into separate BPEL processes, one for each partner in the choreography, including corresponding WSDL descriptions.

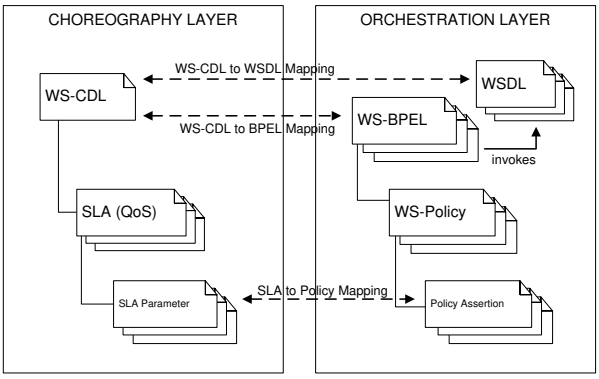

Figure 2. Modeling Approach

On the highest level of abstraction (the choreography layer), a number of models have to be specified which can then be used to generate specific parts for each participant in the business process on the orchestration layer.

This is achieved by transforming the models from the choreography layer to executable code in the orchestration layer as depicted in Figure 2. The models of the choreography layer include a choreography description in WS-CDL and one or more SLAs. The choreography is used to describe the partners in the process and the message exchanges. The SLAs define obligations and guarantees among the participants. They bridge the gap between the choreography description and the SLAs. We have annotated the choreography with the SLA references to allow a pairwise agreement on a specific SLA.

During the transformation, we map the WS-CDL choreography to a number of BPEL processes (the amount depends on the number of participants) and we generate the WSDL descriptions of the Web services each partner has to implement and provide to its business partners. The importance of QoS in cross-organizational business processes makes it necessary to consider these aspects from the beginning of the development process. Similarly, the SLAs are transformed to WS-QoSPolicy statements (our extension to WS-Policy) that are directly attached to the corresponding partner links in BPEL to allow an enforcement by a BPEL engine.

In the following paragraphs, we present each of these transformation steps, the QoS integration and WS-QoSPolicy in detail.

4.2. Mapping WS-CDL to BPEL

The main goal of transforming WS-CDL to BPEL is to allow the participants a rapid modeling and development process and generate relevant BPEL and WSDL documents which can then be used as a basis to implement the private (non-visible) business logic. The projection of such a global description to endpoint processes whose interactions precisely realize the global description is called *endpoint projection* [3].

```
<package>
  <choreography>
    <sequence>
      <!— ... —>
      <sequence>
        <!— ... —>
        <sequence>
          <interaction operation="sendPO" ...>
            <participate fromRoleTypeRef="Customer"
                  toRoleTypeRef="Manufacturer" .../>
            <exchange action="request" ...>
              <!— ... —>
            </exchange>
          </interaction>
          <interaction operation="sendPO" ...>
            <participate fromRoleTypeRef="Customer"
                  toRoleTypeRef="Manufacturer" .../>
            <exchange action="respond" ...>
              <!— ... —>
            </exchange>
          </interaction>
        </sequence>
      </sequence>
    </sequence>
  </choreography>
</package>
```

Listing 3. Choreography Example

In [10], Mendling and Hafner define the basic mapping rules from WS-CDL to BPEL. They use an recursive XSLT-based approach to generate the BPEL processes by iterating through each role type to check the relevance of the node. The authors consider a node as relevant if it contains activities with the attribute `cdl:toRoleTypeRef` and `cdl:fromRoleTypeRef`. However, this approach does not correspond with the endpoint projection definition given above, because more structured BPEL elements are generated than necessary. This is due to the fact that all parent nodes are considered during the mapping process even if they are not directly relevant (it can be considered as a

WS-CDL	BPEL	Semantics
	Activities	
workunit (nested in choice)	case	repeat and block attributes always false in this case
workunit (block = true)	(receive)	Concept of blocking condition not defined in BPEL [2]
workunit (all other cases)	while	repeat = true and block = false
sequence	sequence	Sequential execution of activity units
parallel	flow	Parallel execution of activities
choice	switch	If inspected `roleType` is referenced in the guard condition of the inner workunit
	onMessage (nested in) pick	If inspected `roleType` is not referenced in the guard condition of the inner workunit but referenced in an `interaction` activity
interaction		
action = request	invoke	`fromRoleType` attribute corresponds to inspected role type
action = request	receive	`toRoleType` attribute corresponds to inspected role type. If `cdl:interaction` inside `cdl:workunit` which is defined inside a `cdl:choice` generate a BPEL `onMessage`
action = response	reply	toRoleType attribute corresponds to inspected role type
action = response	receive	`receive` only in the asynchronous case. For synchronous interaction append `outputVariable` to corresponding BPEL `invoke` which is defined in case 1
perform	*no mapping*	Separately defined choreography is performed
assign	assign (for party in roleType)	Variable assignment
silentAction	sequence with nested empty	To be refined in the BPEL process
noAction	empty (for party in roleType)	Do nothing
finalize	compensationHandler	Finalizing activities after completion

Table 1. WS-CDL to BPEL Mapping

simple 1:1 mapping). Listing 3 depicts an example of this problem by using three nested `sequence` elements.

Therefore, we have used and adapted the rules from [10] and propose an extended endpoint projection mechanism based on a so-called *relevance mapping*. The basic idea is to map only those WS-CDL elements which are relevant in the BPEL process. To map the different ordering structures we need to distinguish between *child* and *descendant* relevance. The former describes that a relevant basic activity occurs as an immediate child of the respective ordering structure in the tree whereas the latter describes that a relevant basic activity is nested at an arbitrary level. The relevance of a WS-CDL basic activity is determined by the occurrence of a `cdl:interaction`, `cdl:assign` or `cdl:silentAction` where the `roleType` attribute is matching the `roleType` of the corresponding BPEL process.

If a node represents a relevant activity as described above, it is mapped to a BPEL activity according to Table 1, otherwise no mapping is generated. The basic algorithm for the relevance mapping is depicted in Listing 4.

From Line 2 to 8, we generate a BPEL process for each role type. The algorithm inspects the `cdl:choreography` tag of the WS-CDL document by iterating each activity. If the activity type is an ordering structure or a workunit a relevance mapping has to be performed. If the currently inspected activity is *descendant relevant* we have to consider all child nodes of this activity (Line 12). If an activity is *child relevant* (Line 16), we have to generate the corresponding BPEL mapping according to Table 1 (Line 17). Otherwise, we recursively visit all child nodes (Line 19).

```
1  void transform(WscdlDoc doc, List<RoleType> roles) {
2    for (RoleType role : roles) {
3      for (activity : n.getActivites()) {
4        if (isOrderingStructure(activity) ||
5            isWorkUnit(activity))
6          relevanceMapping(activity, role);
7      }
8    }
9  }
10
11 void relevanceMapping(Node n, RoleType role) {
12   if (isDescendanceRelevant(n)) {
13     if (n.getRelevantChildCount() > 1)
14       createBPELMapping(n);
15     for (child : n.getChildNodes()) {
16       if (isChildRelevant(child))
17         createBPELMapping(child);
18       else
19         relevanceMapping(child, role);
20     }
21   }
22 }
```

Listing 4. Relevance Mapping

For our BPEL mapping we implemented an additional optimization concerning the ordering structures. If a `cdl:parallel` or `cdl:sequence` ordering structure contains only one basic child activity, this ordering structure is ignored in the BPEL mapping (Line 13–14). For instance considering the example from Listing 3, only one BPEL sequence activity will be generated.

In Table 1 we have depicted a detailed overview of the WS-CDL to BPEL mapping rules. These rules are based on the mappings proposed by Mendling and Hafner [10] and adapted where necessary. These adaption mainly include `cdl:interaction` and `cdl:choice`. For the

`cdl:interaction` activity and `cdl:choice` ordering structure, we also have to consider the role types to determine the sending and receiving party. Additionally, we address the synchronous and asynchronous message exchange patterns properly in the `cdl:interaction` activity.

4.3. Generating WSDL Descriptions

The WSDL descriptions define a static structure which can be extracted from the choreography without analyzing the choreography flow in detail. The necessary element mapping from WS-CDL to WSDL is shown in Table 2. On the left side the structure of a WSDL file is used to show the corresponding elements of WS-CDL on the right side.

The knowledge from this mapping is then used to implement a WSDL generation algorithm as shown in Listing 5.

```
1  void generateWSDLs(WscdlDoc wscdl) {
2    for (roleType : wscdl.getRoleTypes()) {
3      if (isRoleTypeUsed(roleType))
4        createWSDL(wscdl, roleType);
5    }
6  }
7
8  void createWSDL(WscdlDoc wscdl, RoleType roleType) {
9    createFile(roleType.behaviorInterface + ".wsdl");
10   createNode("wsdl:definition");
11   for (i : wscdl.getInteractions()) {
12     if (roleType == i.getAttr("toRoleTypeDef"))
13       createNode("wsdl:message");
14   }
15   for (b : roleType.getBehavior()) {
16     ptNode = createNode("wsdl:portType");
17     bdNode = createNode("wsdl:binding");
18
19     for (i : wscdl.getInteractions()) {
20       if (roleType == i.getAttr("toRoleTypeDef")) {
21         ptNode.append("wsdl:operation");
22         bdNode.append("wsdl:operation");
23       }
24     }
25     sNode = createNode("wsdl:service");
26     sNode.append("wsdl:port");
27   }
28 }
```

Listing 5. WSDL Generation Pseudocode

The WSDL generation works as follows: In Line 2–5, we generate a new WSDL document for each `roleType` of the choreography if the service interface is invoked somewhere in the choreography flow. This check is done in the `isRoleTypeUsed()` method. The main idea is to check if the `roleType` is referenced within a `channelType` and a `variable` for this `channelType` exists that is used in an `interaction` with another partner. If this is the case, the `roleType` is in use and a WSDL needs to be generated. The WSDL document itself is created in the `createWSDL()` method (Line 8–28). The methods `createNode()` and `appendNode()` are used to build the WSDL document. For readability we omitted the generation of the XML attributes (which can be seen in Table 2).

WSDL		**WS-CDL**	
Element	*Attribute*	*Element*	*Attribute*
definitions	xmlns:tns	package	xmlns:tns
	targetNS		targetNS
	name	behavior	name
message	name	exchange	informationType
portType	name	behavior	interface
operation	name	interaction	operation
[input ‖ output]	name	exchange	action
	message		informationType
binding	name	behavior	name + “Binding”
	type		“tns:”+interface+“Binding”
operation	name	interaction	operation
soap:operation	soapAction	behavior	interface namespace
input		interaction	operation
soap:body	namespace	behavior	interface namespace
output			
soap:body	namespace	behavior	interfaces namespace
service	name	behavior	interface+“Service”
port	name	behavior	interface+“Port”
	binding		“tns:”+name+“Binding”

Table 2. WS-CDL to WSDL Mapping

4.4. SLA/QoS Integration

The integration of QoS parameters in Web service based business process development raises the need for appropriate techniques to consider QoS at the choreography and orchestration layer. Considering QoS at the choreography layer can be achieved by using SLAs which focus (among others) on performance and dependability aspects of the underlying QoS model. In contrast, the integration of QoS at the orchestration layer can be attained by the use of Web service policies. This section describes how WS-CDL and BPEL can be extended to support QoS attributes.

SLA Integration. As mentioned above, we use SLAs to integrate QoS at the choreography layer. For the definition of the SLAs we decided to use WSLA as it seems to be more suitable than WS-Agreement. For the actual integration, we extended WS-CDL with a construct which holds SLA references. We therefore leverage semantic annotations in WS-CDL constructs using the `description` element as shown in Listing 6.

WS-QoS Policy. In order to bring QoS aspects from the choreography to the orchestration layer, SLAs have to be mapped to the corresponding Web service policies. However, the current WS-Policy specification focuses on security (WS-SecurityPolicy) and reliable messaging (WS-RMPolicy), whereas performance and dependability are not addressed. Hence, we had to extend the WS-Policy framework by defining a WS-QoSPolicy. The WS-Policy Framework therefore provides a grammar for the definition of domain-specific policies.

```
<roleType name="ManRoleType">
  <behavior interface="b2o:manInterface"
            name="ManBehavior"/>
    <description type="semantics">
      <qosp:slaReference
        name="SLA1"
        uri="ManufacturerCustomerSLA.xml"
        serviceconsumer="CustRoleType"
      </qosp:slaReference>
    </description>
  </behavior>
</roleType>
```

Listing 6. SLA Integration in WS-CDL

Before, we go into the details of our mapping between SLAs and policies, we briefly sketch the underlying QoS model. In previous work [16], we have defined a QoS model for Web services by identifying different QoS attributes. Since some attributes are either dependent on external factors or derived from empirical values, not all attributes are determinable in advance. Table 3 illustrates the relevant QoS attributes in the context of Web service choreography.

QoS Attribute	Relevant	Reason (if not relevant)
Processing Time	YES	
Wrapping Time	YES	
Execution Time	YES	
Latency	NO	Represents external factor
Response Time	NO	Depends on external factor
Round Trip Time	NO	Depends on external factor
Throughput	YES	
Scalability	NO	Depends on external factor
Availability	YES	
Accuracy	NO	Depends on empirical values
Robustness	NO	Depends on empirical values

Table 3. QoS Attribute Relevance

According to this table, we consider *Processing Time*, *Wrapping Time*, *Execution Time*, *Throughput*, and *Availability* as possible QoS Attributes in the WS-QoSPolicy. However, guarantees on the *Execution Time* will usually be defined in SLAs instead of *Processing Time* and *Wrapping Time*.

The WS-QoSPolicy defines assertions for all QoS attributes. The normative outline of the assertions is shown in Listing 7. It defines `type`, `unit`, `predicate`, and `value` of the assertion. A concrete example for two such policy assertions is illustrated in Listing 8.

```
<qosp:[QoS]Assertion
          unit="xs:string"
          predicate="tns:PredicateType"
          value="xs:integer | xs:flow"/>
```

Listing 7. WS-QoSPolicy Assertions

```
<wsp:Policy>
  <wsp:All>
    <qosp:ExecutionTimeAssertion unit="seconds"
          predicate="Less" value="5"/>
    <qosp:ThroughputAssertion unit="requests"
          predicate="GreaterEqual" value="1"/>
  </wsp:All>
</wsp:Policy>
```

Listing 8. Assertion Example

SLA/QoS Mapping. Our extension of the WSLA schema restricts the SLA parameters to the pre-defined QoS attributes introduced in the previous section. Therefore, the SLA can be directly mapped to the WS-QoSPolicy which consists of the following two steps: Firstly, each SLA is mapped to a policy and secondly, each SLA parameter is mapped to a policy assertion.

As each SLA may consist of one or more SLOs, we identified three different patterns:

1. One SLO is defined for each SLA parameter.
2. One SLO consists of multiple SLA parameters.
3. SLA parameters are defined in multiple SLOs.

Each of these patterns can be successfully mapped to an equivalent policy. In the first case, one `All` operator is used to contain all policy assertions. For each SLO, exactly one policy assertion will be generated. For example, an SLO `SLOServiceExecutionTime` defines an SLA parameter which corresponds to the type `ExecutionTime`. This parameter will be mapped to the corresponding policy assertion according to the WS-QoSPolicy.

In the second case, SLA parameters are grouped through an SLO by using the logical operators `And`, `Or`, `Not`, `Implies`. Table 4 shows how these constructs can be mapped to equivalent WS-QoSPolicy operators.

SLA operator		WS-QoSPolicy operator
`And`	→	`All`
`Or`	→	`ExactlyOne`
`Not`	→	Reverse predicate
`Implies`	→	`ExactlyOne` and reverse predicate

Table 4. SLA operator mapping

For example, such a grouping of SLA parameters can be used to define an SLO `SLOServicePerformance` by combining `Throughput` and `ExecutionTime` in various way using the provided logical operators.

In the third case, for each SLO a time period has to be specified. Therefore, it is possible to define multiple SLOs for different time periods. For instance, during peak-hours the execution time of a service has to be less than a specific value. A detailed description of the SLA/QoS mapping algorithms can be found in [6].

WS-QoS Policy Integration. The definition of a QoS policy and QoS/SLA mapping rules are the fundamental concepts for considering QoS in Web service based business process development. Yet, the question remains how to integrate the generated QoS policies in the orchestration layer. Regarding the top-down modeling approach of Web services, two integration approaches can be differentiated: Policies can either be attached to service descriptions (WSDL) or be integrated in BPEL processes.

Attaching policies to WSDL descriptions following the WS-PolicyAttachment [20] specification has two main drawbacks. Firstly, service invocations are always subject to a policy, even if the service consumer has no corresponding SLA. Secondly, the service provider cannot differ between multiple policies for the same service since policies do not contain information about the participating parties. Therefore, following the second approach the policies should be integrated in BPEL processes.

Extensibility in BPEL is achieved by allowing elements from other namespaces. The BPEL `partnerLink` element is the place to integrate the policy. For this integration, both synchronous (request-reply) and asynchronous (callback) message exchange patterns have to be considered. In contrast to the asynchronous case, the service provider has no additional information about the service consumer in the synchronous case, because the `partnerLink` has no service consumer specific details. Therefore, the policy has to be integrated at the service consumer side as illustrated in Listing 9.

```
<process>
  <partnerLinks>
    <partnerLink name="POService"
                 partnerLinkType="ns1:POServiceLT"
                 partnerRole="POServiceRole">
          <wsp:Policy xmlns:qosp="..." xmlns:wsu="..."
                   wsu:Id="xs:QName"
                   qosp:operation="...">
            ...
          </wsp:Policy>
        </partnerLink>
    ...
  <partnerLinks>
  ...
</process>
```

Listing 9. Policy Integration in BPEL

5. Implementation

We have implemented the concepts and algorithms in Java 1.5 using a simple Swing-based graphical user interface. The architecture of the system is depicted in Figure 3.

It consists of three main parts:

- *Editing:* It allows the developer to load and inspect a choreography and add SLA references to a specific role type. We have not built an editor for modeling the choreography, this was out of scope and can be done for example with Pi4soa [15].

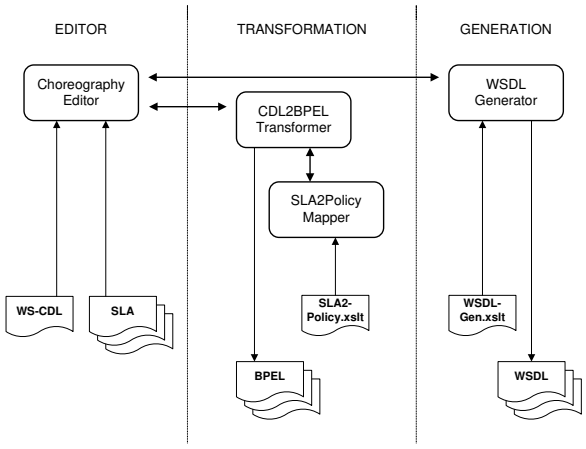

Figure 3. System Architecture

- *Transformation:* This part implements the algorithms for transforming WS-CDL to BPEL, and SLA to policies. The WS-CDL to BPEL transformation is implemented using the DOM4J API [11] whereas the SLA transformation is implemented using XSLT [19]. During the transformation step one BPEL document is generated for each partner including the policy references which conform to the SLAs in the choreography layer.

- *Generation:* The last part implements the generation of the WSDL files from a choreography. It simply generates all the WSDL files in a directory selected by the user according to the algorithm described in Section 4. This part is again implemented using an XSLT stylesheet.

In Figure 4, we have depicted a simple screenshot of our tool support. When a choreography description is loaded, the role type definitions can be seen in a tree-based view on the left side. After adding an SLA reference to a specific role type, the "Tools" menu item can be used to start the different transformations and the WSDL generation.

The decision to implement the tool-support with Java Swing was mainly due to simplicity reasons. As a future work the integration into the Pi4soa Eclipse plugin is envisioned to achieve better integration with existing WS-CDL tool support.

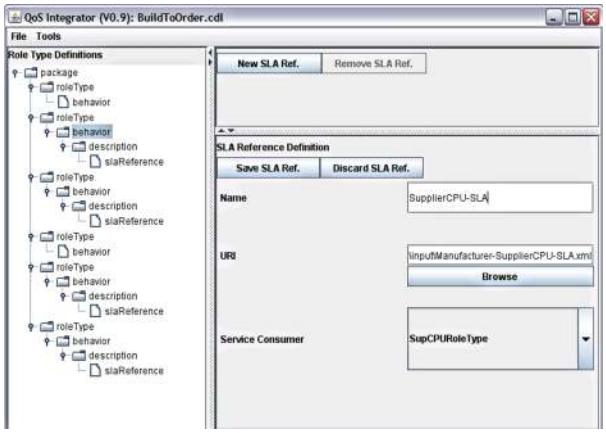

Figure 4. QoS Integrator

6. Validation and Discussion

We have implemented the case study from Section 2 to demonstrate the feasibility of our approach[2]. For modeling the choreography we have used the Pi4soa Eclipse Plugin. During the modeling phase, the important part is the identification of partners in the process and the messages that are exchanged among the partners. Most parts of the BTO scenario are implemented in the choreography itself. However, some non-observable implementation specific details cannot be considered from a choreography point of view but have to be implemented internally by the choreography participants.

The choreography itself is then used to generate the BPEL and WSDL templates for each partner in the choreography. The SLAs are modeled pairwise and independently among the partners. The partners agree on a set of runtime constraints that need to hold during the message interactions. In general, the SLAs are independent of the choreography itself, nevertheless the integration of an SLA in the development process can be achieved by adding an SLA reference to a specific `roleType` in the WS-CDL (compare Listing 6 for an integration example)

In our case study we have identified four different SLAs: one between the customer and the manufacturer, and one for every manufacturer-supplier pair. For example, an SLA between the manufacturer and the CPU supplier specifies the expected response time, throughput and execution time of a service including the periods where these obligations are valid. After the transformation of the WS-CDL to BPEL processes, the generated BPEL process contains a `partnerLink` annotated with the following WS-Policy statements to express the SLOs as enforceable policies as it can be seen in Listing 7 for execution time and throughput.

[2]The case study files can be downloaded from `http://www.vitalab.tuwien.ac.at/~florian/qosintegrator/`.

For our case study we summarize the different input files and the generated output files in Table 5. These files can be found online at the URL provided in the footnote.

Processing Step	Choreography Layer	Orchestration Layer
Transformation	BuildToOrder.cdl	Initiator.bpel Customer.bpel Manufacturer.bpel SupplierCPU.bpel SupplierHD.bpel SupplierMB.bpel
WSDL Generation	BuildToOrder.cdl	Customer.wsdl Manufacturer.wsdl SupplierCPU.wsdl SupplierHD.wsdl SupplierMB.wsdl

Table 5. Input and Output Files

The transformation of SLAs to policy assertions does not generate new files, instead, the policy assertions are directly embedded in the corresponding BPEL partner link.

After the transformation steps, the generated BPEL and WSDL files from Table 5 have to be taken as a starting point for implementation of the *private* business logic. It mainly deals with aspects which cannot be modeled from a global viewpoint in the choreography. These internal implementations are referred to as silent actions (containing the internal business logic) and have to be implemented during refinement of the BPEL code. After that, the services and the BPEL processes can be deployed to an orchstration engine.

Discussion. During the implementation of our case study we encountered several aspects which have to be considered when using such top-down modeling approach. Some of these issues seem inherent to the domain of model-driven development in general. On the one hand, our approach is based on choreographies representing a global viewpoint of the business processes which raises the need for precise modeling of the global behavior. To be more concrete, the business partners have to precisely agree on the message format used for their interaction. On the other hand, after the choreography was initially defined, the underlying business model may evolve and lead to significant changes. Such changes clearly affect the partner processes which causes the generation of new BPEL processes and corresponding WSDL files.

Another point of discussion is the use WS-CDL. Some may scrutinize why we use choreographies instead of following a bottom-up approach that builds on orchestration languages such as BPEL. In fact, both modeling approaches are feasible and have their strengths and weaknesses. However, BPEL is intended for modeling business processes without knowledge of global viewpoint. In contrast to this, we decided to stay close to the vision of cross-organizational choreography descriptions by using WS-CDL.

7. Related Work

Integrating QoS in Web service based business process development has not yet received much attention whereas modeling of choreographies is subject of various research activities (e.g., [4, 23]). We mainly discuss existing choreography modeling and transformation approaches as well as extensions of current Web services standards to include QoS attributes and the integration of policies in BPEL.

Choreography Modeling and Transformation. Mendling and Hafner [10] define mapping rules for the derivation of BPEL processes from a WS-CDL choreography description. For each WS-CDL ordering structure and activity the corresponding BPEL construct respective activity is determined. These mapping rules define the basis for the mapping rules used throughout the top-down modeling process in this work. Whereas the mapping of WS-CDL to BPEL is referenced in detail, the generation of WSDL interfaces using in the BPEL process is not addressed explicitly. In contrast to this work, no explicit endpoint projection rules are defined to determine which ordering structures are relevant for the participants of the choreography description. Finally, this work additionally defines mapping rules for the generation of WSDL descriptions which correspond to the service interface descriptions of the derived BPEL processes.

In [5], Diaz et al. use an intermediary model for the generation of BPEL processes from a WS-CDL choreography description concentrating on Web services where time constraints play a critical role. A choreography description is first transformed into a Timed Automata model which is verified and validated for correctness using formal model checking techniques. This model is then further used to generate BPEL processes. In contrast to this work the focus is laid on the generation and verification of the Timed Automata model. Detailed mapping rules for the derivation of BPEL processes out of this model are not specified. In the context of top-down modeling it seems more appropriate to perform a direct mapping between WS-CDL and BPEL instead of using an intermediary model.

Pi4soa [15] is a toolset from $\pi 4$ Technologies and one of the first WS-CDL implementations. They provide a designer tool as an Eclipse plugin, which we used for modeling our choreographies, and a possibility to generate Java services from a WS-CDL. The support for generating BPEL processes is currently in progress. In contrast to pi4soa our works considers QoS from the beginning of the development. It might be interesting to include the SLA/QoS related aspects into the pi4soa Eclipse plugin.

Decker et al. [4] propose an new extension to BPEL, called BPEL4Chor that allows modeling of choreographies within BPEL by leveraging an interconnected interface behavior model, whereas WS-CDL represents an interaction model. As stated in [4], it has not been investigated yet which of these two approaches is more appropriate for human modelers. While we follow a top-down approach by transforming WS-CDL into BPEL, the authors propose a bottom-up approach by introducing a new choreography layer on top of BPEL. However, in contrast to our work, the integration of QoS into Web service choreographies is not addressed.

QoS Integration Approaches. Several approaches exist which integrate QoS into the Web service stack. These approaches can be seen complementary to the work presented in this paper. Here we focus on the modeling part and left out the execution part of the developed orchestrations and their Web services.

In our previous work [16], we proposed a QoS model for performance and dependability related aspect of Web services and a client-side measurement tool determine the QoS from a client perspective.

In [17], a Web service broker (WSB) is used to perform QoS based service selection based on a set of defined QoS parameters. A client application sends a service request along with QoS requirements to a WSB. The WSB then receives all the providers and their QoS values to select the best one.

Garcia et al. [7] propose an architecture for QoS management by extending the current Web services standards UDDI and WS-Policy. This approach includes an extended UDDI information model specifying a QoS tModel and the use of WS-Policy to specify QoS policies. The authors propose a broker based architecture to select appropriate services (fulfilling functional and QoS requirements) in the UDDI registry and reports the selected service back to the consumer. The monitor component is used to intercept messages exchanged between the consumer application and the Web service to monitor the service execution and passes updated QoS information to the broker to update the QoS information. Similar to this work the WS-Policy framework is used to express QoS related aspects for Web services. However, no further details on the proposed QoS policy are asserted.

8. Conclusions

There have been some considerable debates as to the relationship between choreography and orchestration. Some people argue that there is no need for choreography and all business interactions can, and in fact, should be modeled in BPEL. Others advocate the use of modeling by using WS-CDL but then lament the lack of execution abilities. The prime motivation for the contribution of this paper is today's lack of modeling support for QoS-aware business processes.

In particular, the need for QoS-aware processes is apparent in inter-organizational business processes.

The novelty of our approach lies within the fact that we consider SLAs as first class entities while modeling service choreographies. Our approach allows for automatic generation of executable BPEL orchestrations and WSDL files for each partner in the choreography. A novel contribution is the mapping of QoS information specified in SLAs to WS-QoS policies which are attached to the BPEL process. As a consequence, a policy-aware middleware can verify and possibly enforce SLAs. The approach has been implemented and the feasibility is demonstrated using a simplified version of a Built-to-Order case study.

As future work we envision the implementation of our tool support within the Pi4soa Eclipse plugin to allow a better integration with existing modeling tools. Additionally, we need to study the applicability of WS-Agreement as an alternative way to specify the SLAs and transform them to WS-QoSPolicy.

References

[1] Active Endpoints. *ActiveBPEL Engine*, 2007. http://www.active-endpoints.com/ (Last accessed: May 07, 2007).

[2] A. Barros, M. Dumas, and P. Oaks. A Critical Overview of the Web Services Choreography Description Language (WS-CDL). *BPTrends Newsletter*, 3(3), Mar. 2005.

[3] M. Carbone, K. Honda, N. Yoshida, and R. Milner. Structured Communication-Centred Programming for Web Serices. In *Proceedings of the 16th European Symposium on Programming (ESOP'07), Barga, Portugal*, 2007.

[4] G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4chor: Extending BPEL for Modeling Choreographies. In *Proceedings of the IEEE International Conference on Web Services (ICWS'07), Salt Lake City, Utah, USA*, July 2007.

[5] G. Diaz, M. Cambronero, J. Pardo, V. Valero, and F. Cuartero. Automatic generation of Correct Web Services Choreographies and Orchestrations with Model Checking Techniques. In *Proceedings of the International Conference on Internet and Web Applications and Services (ICIW'06), Guadeloupe, French Caribbean*, Feb. 2006.

[6] C. Enzi. Implementing QoS in Web Service based Business Process Development Scenarios. Master's thesis, Technical University of Vienna, Austria, 2007. URL: http://www.vitalab.tuwien.ac.at/~florian/qosintegrator/da_enzi.pdf (Last accessed: August 1, 2007).

[7] D. Garcia and M. Toledo. A Web Service Architecture Providing QoS Management. In *Proceedings of the Fourth Latin American Web Congress (LA-WEB'06), Mexico*, Oct. 2006.

[8] Grid Resource Allocation Agreement Protocol (GRAAP) WG. *Web Services Agreement Specification (WS-Agreement)*, Nov. 2005. (Last accessed: May 7, 2007).

[9] IBM. *Web Service Level Agreement (WSLA) Language Specification*, Jan. 2003. http://www.research.ibm.com/wsla/, (Last accessed: May 7, 2007).

[10] J. Mendling and M. Hafner. From WS-CDL Choreography to BPEL Process Orchestration. *Journal of Enterprise Information Management*. forthcoming.

[11] MetaStuff Ltd. *DOM4J*, 2005. http://www.dom4j.org (Last accessed: May 07, 2007).

[12] OASIS. *Web Service Business Process Execution Language 2.0*, 2006. URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpe (Last accessed: Apr. 17, 2007).

[13] M. P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 3–12, Dezember 2003.

[14] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[15] pi4 Technologies Foundation. *pi4soa*, 2007. URL: sourceforge.net/projects/pi4soa/ (Last accessed: May 9, 2007).

[16] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06), Chicago, IL, USA*. IEEE Computer Society, 2006.

[17] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller. A Concept for QoS Integration in Web Services. In *Proceedings of the 1st Web Services Quality Workshop (WQW'03), Rome, Italy*, 2003.

[18] M. Völter and T. Stahl. *Model-Driven Software Development : Technology, Engineering, Management*. John Wiley & Sons, June 2006.

[19] W3C. *XSL Transformations (XSLT) - Version 1.0*, Nov. 1999. http://www.w3.org/TR/xslt (Last accessed: May 07, 2007).

[20] W3C. *Web Services Policy Attachment*, 2004. URL: http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polatt/ (Last accessed: May 9, 2007).

[21] W3C. *Web Services Policy Framework*, 2004. URL: http://www-128.ibm.com/developerworks/library/specification/ws-polfram/ (Last accessed: May 9, 2007).

[22] W3C. *Web Services Choreography Description Language (WS-CDL)*, Nov. 2005. URL: http://www.w3.org/TR/ws-cdl-10/ (Last accesssed: May 03, 2007).

[23] J. M. Zaha, A. Barros, M. Dumas, and A. ter Hofstede. Let's Dance: A Language for Service Behavior Modeling. In *Proceedings of the 14th International Conference on Cooperative Information Systems (CoopIS'06), Montpellier, France*. Springer, Oct. 2006.