TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

MASTERARBEIT

Integrating QoS in Web service based Business Process Development Scenarios

Thema

Ausgeführt am Institut für

Informationssysteme

der Technischen Universität Wien

unter der Anleitung von Univ.-Prof. Mag. Dr. Schahram Dustdar und Univ.-Ass. Dipl.-Ing.(FH) Florian Rosenberg

durch

Christian Enzi

Name

Urbangasse 6/3/25, 1170 Wien

Anschrift

Datum

Unterschrift (Student)

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Wien, am 06.Juli 2007

Christian Enzi

Abstract

The scope of this master thesis is the integration of Quality of Service (QoS) in Web service based business process development scenarios.

QoS comprises many different categories, ranging from secure and reliable messaging, as well as performance and dependability related aspects of Web services (such as throughput, response time or availability). Web service based business process development involves two composition techniques of the current Web services stack, namely choreography and orchestration. The engineering of Web service based business processes represents a top-down modeling approach of Web services in which private executable business processes are derived from a global choreography description. An integration of the above mentioned QoS aspects throughout the choreography and orchestration layer has not been considered yet. This work will focus on such an integration effort. On the one hand the consideration of QoS aspects at the choreography description layer will be revealed by the use of Service Level Agreements (SLAs). On the other hand the mapping of QoS aspects of the choreography description layer to the orchestration layer will be analyzed in detail. For these purposes a policy for the QoS domain will be defined allowing Web service policies to be derived from the QoS obligations stated in SLAs. Subsequently an integration of these policies throughout the orchestration layer will be revealed. Finally an implementation along with a use case will be provided as a proof of concept.

Zusammenfassung

Inhalt dieser Magisterarbeit ist die Integration von Dienstgüte (Quality of Service, kurz QoS) innerhalb des Entwicklungsprozesses von Web service basierten Geschäftsprozessen.

Der Begriff Dienstgüte umfasst eine Vielzahl von Kategorien. Auf der einen Seite wird Dienstgüte mit einem sicheren und zuverlässigen Nachrichtenaustausch assoziiert, auf der anderen Seite werden damit Leistungs- und Zuverlässigkeitsindikatoren (wie z.B. Antwortzeiten, Durchsatzraten oder Verfügbarkeit) von Web services definiert.

Die Entwicklung von Web service basierten Geschäftsprozessen umfasst einen Modellierungsprozess in welchem ausführbare Geschäftsprozesse aus einem globalen Modell abgebildet werden. Bei diesem Entwicklungsprozess von Web services kommen die zwei Kompositionstechniken Choreographie und Orchestrierung zum Einsatz welche jeweils eine verschiedene Modellierungsebene darstellen. Auf Choreographie-Ebene werden die Interaktionen aller Teilnehmer aus globaler Sicht betrachtet und in einem Modell (Choreographie) beschrieben. Hingegen werden auf Orchestrierungs-Ebene die Interaktionen des einzelnen Teilnehmers betrachtet und daraus ausführbare Geschäftsprozesse (BPEL Prozesse) erstellt. Aufgrund des unterschiedlichen Abstraktionsgrades beider Kompositionstechniken lassen sich aus einer Choreographie entsprechende BPEL Prozesse abbilden. Die Einbindung von Aspekten der Dienstgüte in den zuvor beschriebenen Modellierungsebenen stellt einen bis dato nicht berücksichtigten Ansatz dar.

Im Rahmen dieser Magisterarbeit wird der Fokus auf solch eine Integration von Dienstgüte gelegt. Die Berücksichtigung von Dienstgüte auf Choreographie-Ebene wird hierbei durch die Verwendung von Dienstgütevereinbarungen (Service Level Agreements, kurz SLAs) erzielt. In weiterer Folge wird gezeigt wie sich Aspekte der Dienstgüte von der Choreographie-Ebene auf die Orchestrierungs-Ebene abbilden lassen. Zu diesem Zweck wird eine Richtlinie (Policy) für die Dienstgüte von Web services definiert. Dadurch lassen sich aus den in Dienstgütevereinbarungen festgelegten Verpflichtungen entsprechende Web service Richtlinien ableiten. Daran anschliessend erfolgt eine Analyse in welcher Form derartige Richtlinien auf Orchestrierungs-Ebene eingebunden werden können. Abschliessend wird eine

Implementierung vorgestellt welche durch die Umsetzung eines Anwendungsfalles die Durchführbarkeit der vorgestellten Ansätze demonstiert.

Table of Contents

List of Figures

1 Introduction

Web services are not only a key factor in the field of enterprise application integration but also gain more and more importance for cross-organizational business-to-business (B2B) scenarios. These B2B scenarios involve multiple business partners interacting with each other by exchanging information in a structured way. This kind of information exchange is performed by the coordinated invocation of Web services which expose interfaces to internal business processes of the involved business partners.

The engineering of such Web service based business processes involves two composition techniques of the current Web services stack, namely choreography and orchestration. The concepts of choreography and orchestration represent different viewpoints in service composition.

Choreography represents a global viewpoint which “captures collaborative processes involving multiple services where the interactions between these services are seen from a global perspective” [9 p.2]. A choreography does not describe any internal action that occurs within a participating service such as internal computation or data transformation but focuses on the observable public exchange of messages encompassing all the interactions that are relevant with respect to the choreography’s goal. Choreographies can be defined using the Web Services Choreography Description Language (WS-CDL) [19] specification.

Orchestration represents a local viewpoint which “deals with the description of the interactions in which a given service can engage with other services, as well as the internal steps between these interactions (e.g., data transformations)” [9 p.2]. An orchestration captures a private executable business process (BPEL process) representing control from one party's perspective. In contrast to a choreography description such an orchestration defines the internal actions in which a service engages but does not consider the message exchange from a global point of view. Furthermore it is intended to be executed by an orchestration engine. BPEL processes can be defined using the Web Service Business Process Execution Language (WS-BPEL) [20] specification.

These concepts of choreography and orchestration imply a top-down modeling approach. “Business partners agree on a specific choreography description defined in

WS-CDL in order to achieve a common goal which is then used to generate BPEL process stubs for each party"[12 p.2].
Within the context of Web service based business process development Quality of Service (QoS) aspects play an essential role. QoS comprises many different categories, ranging from secure and reliable messaging, as well as performance and dependability related attributes of Web services (such as availability, response time or throughput). QoS related terms and conditions are generally specified in a Service Level Agreement (SLA) which needs to be fulfilled during the service execution among the business partners.

Prior to this thesis a case study for a B2B scenario was implemented. Based on the composition techniques choreography and orchestration a top-down modeling of Web service based business processes was performed. The design of the implemented case study aimed at simulating a Web service scenario in which one of the involved business partners acts as both, a service provider as well as a service consumer being dependent on services from other partners in order to process service requests. This dependency emphasizes the importance of QoS aspects in cross-organizational B2B scenarios. The realization of the case study revealed that current research has not considered the integration of QoS in Web service based business processes development scenarios.
Such a QoS integration approach is discussed in this thesis. The integration of QoS will be performed at different stages. Initially QoS requirements will be considered at the choreography layer. Subsequently these QoS requirements will be transformed and integrated at the orchestration layer. Neither WS-CDL nor WS-BPEL specifies the declaration of QoS attributes. Hence appropriate techniques are required for the integration of such information. This integration will be accomplished by the use of SLAs and Web service policies.
The definition of QoS plays a crucial rule in cross-organizational business processes. Each participant offers services to other partners which the latter need to run their businesses. Therefore, a certain degree of reliability concerning performance and dependability related QoS requirements is desired and has to be specified and explicitly expressed from the beginning of the modeling phase. SLAs provide proper methods for establishing agreements on such QoS requirements between the participants involved. In accordance to choreographies − where participants achieve

agreement on the data exchanged − SLAs allow the specification of QoS agreements on the business level. In contrast to SLAs Web service policies are more technology oriented by specifying requirements on the service level. On this account policies are a suitable technology for integrating QoS at the orchestration layer.
A primary concern of the QoS integration approach is the integration of SLAs from the beginning of the choreography development process. In accordance to the top-down modeling process of Web services SLA requirements are automatically transformed and mapped to equivalent Web service policies which are further attached to the generated BPEL process of the affected partner.

1.1 Problem Definition

The implementation of the QoS integration approach faces the following set of problems.

Endpoint projection

The QoS integration approach implies a mapping of choreography descriptions to BPEL processes. Whereas the mapping of WS-CDL language constructs to corresponding WS-BPEL language constructs has been actively discussed in literature an approach for a correct endpoint projection has not been provided yet. Such an endpoint projection is a requirement in order to generate BPEL processes whose interactions precisely realize the global choreography description model. This endpoint projection problem will be referenced by a relevance mapping algorithm.

Integration of SLAs at the choreography layer

WS-CDL does not discuss how QoS related aspects can be provided with a choreography description. As mentioned above SLAs will be used to include such information at the choreography layer. This integration approach requires two aspects to be considered. First, an appropriate language specification for the declaration of SLAs will be identified. Second, an extension of WS-CDL will be performed which enables an integration of existing SLA declarations.

Integration of policies at the orchestration layer

Similar to WS-CDL the WS-BPEL specification does not consider QoS related information. Following the approach used at the choreography layer an existing Web service standard (WS-Policy) will be used for the integration of QoS at the orchestration layer. Basically, an integration of policies at the orchestration layer can be performed at different stages. Policies can either be attached to service descriptions (WSDL) or be integrated in BPEL processes. An evaluation of both approaches will be provided highlighting the advantages of policy integration inside BPEL processes.

Mapping of SLAs to policies

In order to implement a continuous QoS integration approach it is necessary to map SLAs to corresponding Web service policies. However, the current WS-Policy specifications focus on security and reliable messaging related QoS aspects only − performance and dependability related QoS aspects (being subject to the obligations defined in SLAs) have not been considered yet. Therefore a QoS mapping implies an extension of the current WS-Policy framework by defining a policy for the QoS domain. In this thesis such a policy will be defined by the declaration of a WS-QoSPolicy. Subsequently appropriate SLA to policy mapping rules will be defined.

1.2 Organization of this thesis

The remaining part of this thesis is organized as follows. In Section 2 existing Web service technologies will be described which form the basis for the aspects discussed in the following sections. Section 3 presents the underlying concepts of Web service based business process development showing by what means executable business processes can be derived from a global choreography model description. In section 4 the actual integration of QoS throughout business process development scenarios will be expatiated. Section 5 demonstrates the implementation which has been designed to evaluate the QoS integration process for feasibility. Finally related work for the diverse concepts of this thesis will be discussed in section 6.

2 Basic Concepts

The integration of QoS in Web service based business process development scenarios is based on existing Web service specifications. These specifications involve the concepts of choreographies, orchestrations, Service Level Agreements and policies.
In the following an overview of the various specifications of these concepts will be provided. Furthermore the different aspects of QoS with regard to Web services will be revealed.

2.1 WS-CDL

"The purpose of the Web Services Choreography Description Language (WS-CDL), is to define multi-party contracts, which describe the externally observable behavior of Web Services and their clients (usually other Web Services), by describing the message exchanges between them" [9 p.7]. WS-CDL [19] represents a specification language which allows each involved party to describe its part in the message exchange by specifying details on collaborations, information handling and activities.

2.1.1 Collaborations

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

- ParticipantType

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

  A role 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.
- RelationshipType

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

- ChannelType

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

The definition and handling of information within a choreography is performed by information types and variables.

- `InformationType`

  Information used within a choreography is specified by information types 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. Listing 1 illustrates an information type definition. The information type is specified by the attribute `type` which contains a qualified name. In the example below the information type corresponds to the `QuoteRequest` definition from the namespace which is specified by the prefix `b2o`. This namespace would be defined in the root element of the choreography.

  ```
  <informationType name="QuoteRequest" type="b2o:QuoteRequest" />
  ```

  Listing 1 Information Type Definition

- `Variable`

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

2.1.2 Activities

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

Ordering Structures

Ordering structures are block structured, enclosing a number of sub-activities which can be used recursively.

– `Sequence`

  The `sequence` activity defines one or more activities that are executed in a sequential pre-defined order.

– `Parallel`

  The `parallel` activity defines one or more activities that are executed concurrently without any pre-defined order.

– `Choice`

  The `choice` activity specifies that only one of two or more activities will be performed. The `choice` activity actually captures two kinds of choices, namely data-driven or event-driven choices. In the first case the choice is based upon a boolean condition where data variables will be evaluated at the time the `choice` activity is reached. In the second case the choice holds until an event occurs. Such an event can be the occurrence of an interaction or an action throughout the choreography flow resulting in certain variables being populated [9]. The type of a `choice` activity depends on the definition of the enclosed `workunit` activities.

Workunit

A `workunit` prescribes 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.

Listing 2 illustrates three `workunit` scenarios. In the first scenario the enclosed activities of the `workunit` activity will be repeated as long as the variable `QuoteAccept` evaluates to false. In the second scenario the first `workunit` activity which defines a `guard` condition evaluating to true will be executed. In the thrid scenario the processing of the enclosed activities will be blocked as long as the variable `POAcknowledge` is not available.

```
<!-- workunit with repetition condition -->
<workunit guard="cdl:getVariable('QuoteAccept','accept','')=false()"
name="QuoteBartering" repeat="true()">
   <!-- some activities -->
</workunit>
```

```
<!-- workunits with competitive guard conditions inside choice -->
<choice>
   <workunit guard="cdl:getVariable('HWnotInStock','','')&gt;0"
       name="Choice_HWnotInStock">
      <!--some activities -->
   </workunit>
   <workunit guard="cdl:getVariable('HWnotInStock','','')=0"
       name="Choice_HWInStock">
      <!--some activities -->
   </workunit>
</choice>

<!-- workunit with blocking condition -->
<workunit
       guard="cdl:isVariableAvailable('POAcknowledge','',''tns:Customer"
       name="POProcess" block="true">
       <!--some activities -->
</workunit>
```

Listing 2 Workunit Scenarios

Basic Activities

Basic activities define the interactions, actions and variable assignments of the choreography flow.

– `Interaction`

The `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 3 illustrates two `interaction` activities which define a message request and response (corresponding to a synchronous messaging scenario). Throughout the message request the operation `"requestForQuote"` will be invoked at the corresponding WSDL interface of the `"ManRoleType"`. The message request is stored in the variable `"QuoteRequest"`. The `"ManRoleType"` will respond to the service invocation by returning the variable `"QuoteResponse"`.

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

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

Listing 3 Interaction Activity

– `Assign`

  The `assign` activity enables the creation or modification of variables based on XPath expressions or other variables.

– `Perform`

  The `perform` activity enables the invocation of another choreography within the context of the executing choreography.

– `SilentAction`

  The `silentAction` activity defines an action with non-observable behavior which is either performed by exactly one or all participants of the choreography (e.g., internal data processing). A `silentAction` will have to be further defined at the orchestration layer.

– `NoAction`

  The `noAction` activity explicitly defines the condition where a participant will not perform any action.

2.2 WS-BPEL

“The Business Process Execution Language for Web services (WS-BPEL or BPEL for short) 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” [20 p.10]. BPEL specifies details on the relationships and activities of a business process.

2.2.1 Relationships

The relationships of a business process are defined by `partnerLinks`. Partner links define different parties that interact with the BPEL process (whose services are invoked during process execution).

Listing 4 specifies the definition of the `partnerLinks` construct.

```
<partnerLinks>
   <partnerLink name="NCName"
      partnerLinkType="QName"
      myRole="NCName"?
      partnerRole="NCName"?
      initializePartnerRole="yes|no"? />+
</partnerLinks>
   <!-- ... -->
</process>
```

Listing 4 Partner Links Construct

Each partner link is related to a specific `partnerLinkType` that characterizes it. The attribute `myRole` indicates the role of the business process itself whereas `partnerRole` indicates the role of the partner. The usage of the role attributes depends on the messaging type. In the case of synchronous messaging a partner link always contains one of the following role attributes:

- "myRole" if the BPEL process receives an invocation
- "partnerRole" if the BPEL process invokes a partner service

In the case of asynchronous messaging a partner link contains both roles.

2.2.2 Activities

A BPEL process comprises two types of activities, namely structured and basic activities.

Structured Activities

Structured activities define the order in which a collection of activities can take place and can be used in a recursive way.

- `Sequence`

  The `sequence` activity defines one or more activities that are executed in a sequential pre-defined order (corresponds to `cdl:sequence`).

- `Flow`

  The `flow` activity defines one or more activities that are executed concurrently without any pre-defined order (corresponds to `cdl:parallel`).

- `Switch`

  The `switch` activity consists of one or more conditional branches which are defined by `case` elements with guard conditions (corresponds to `cdl:choose`).
- `While`

  The `while` activity is defined by a guard condition. All nested activities will be repeated as long as the guard condition evaluates to true (corresponds to `cdl:workunit` with repetition condition). The guard condition of the `switch/case` and `while` activity is defined by a Boolean conditional expressions according to the XPath 1.0 lexical rules.
- `Pick`

  The `pick` activity awaits the occurrence of a message to be received and then performs the activities which are defined for that message event. The message event and activities are defined by one or more `onMessage` activites.

Basic Activities

Basic activities define the Web service operations and variable assignments of the business process. The basic activities `invoke`, `receive`, `reply` and `onMessage` are bound to a specific SOAP operation of a WSDL interface through declaration of the attributes `operation`, `partnerLink` and `portType`.

- `Invoke`

  The `invoke` activity defines a Web service request operation from the prospect of the service requestor. The Web service request message is defined by the attribute `inputVariable`. In the case of synchronous messaging (request-response) a further variable `outputVariable` will be defined which stores the response message of the Web service provider. In the case of asynchronous messaging only the input variable will be defined as a response is not expected as part of the service invocation.
- `Receive`

  The `receive` activity defines a Web service request operation from the prospect of the service provider. The Web service request message is defined by the attribute `Variable`.

– `Reply`

  The reply activity defines a Web service response operation from the prospect of the service provider. The Web service response message is defined by the attribute `Variable`.

– `OnMessage`

  The `onMessage` activity corresponds to a special `receive` activity which is used in conjunction with a `pick` activity.

– `Assign`

  The `assign` activity enables data manipulation by the creation or modification of variables based on XPath expressions or other variables.

– `Empty`

  The `empty` activity explicitly defines a condition where no activity will be performed.

2.3 Service Level Agreements

"Service Level Agreements (SLAs) are agreements between a service provider and a service consumer and as such define the obligations of the parties involved" [21 p.2]. These obligations consist of Service Level Objectives on performance and dependability related QoS attributes of Web services. Two specification languages can be differentiated which enable the definition of agreements:

– WSLA (Web Service Level Agreement)
– WS-Agreement

2.3.1 WSLA

The WSLA language specification [21] is part of the WSLA framework developed by IBM which allows the definition and monitoring of SLAs for Web services. A final publication (WSLA v. 1.0) was published in January 2003

"In principle, there are many possible types of information and rules that can be included in an SLA; however there is consensus on the general structure of an SLA. WSLA embraces this structure by dividing an SLA into three sections: parties, service definition, and obligations" [15 p.7].

The parties section identifies the signatory parties of the SLA (service provider and service consumer). Furthermore supporting parties may be defined (e.g., a measurement service which monitors SLA enforcement).

The service definition section specifies the service and the corresponding service objects (abstraction of a service, e.g., a WSDL/SOAP operation) which are subject to the SLA. The service object(s) is bound to one or more SLA parameters which correspond to the attributes of the QoS model. These SLA parameters are further linked to metrics which define how SLA parameters have to be measured.

The obligations section comprises Service Level Objectives and action guarantees. A Service Level Objective (SLO) defines the guarantees and constraints that may be imposed on SLA parameters. An action guarantee defines an action to be performed if a particular state is reached (e.g., the guarantees of a SLO get violated).

The WSLA language specification provides an XML-based schema for defining the overall structure of an SLA. This basic structure of WSLA is illustrated in the UML diagram shown in Figure 1.

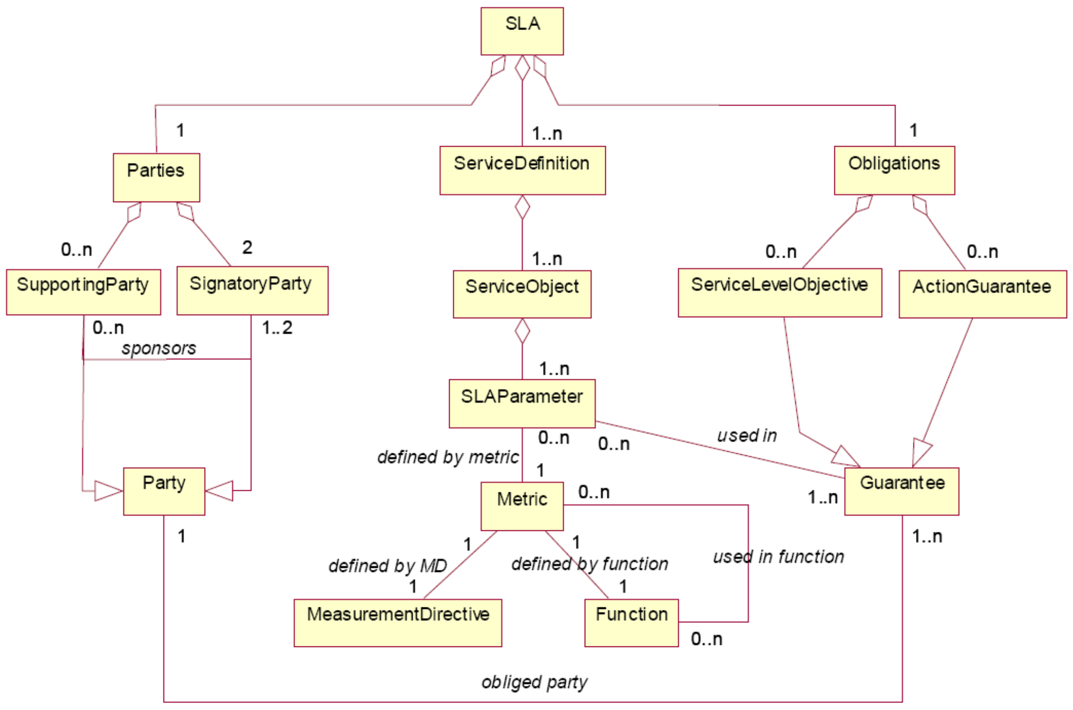


Figure 1 WSLA - Basic Structure [21]

An SLA contains Service Level Objectives about SLA parameters. SLA parameters are bound to a service object which is part of the service definition. These SLA parameters are linked to metrics which define how SLA parameters have to be measured. The following sections provide an overview of these basic structures of an SLA.

2.3.1.1 Service Definition

“The service definition part of an SLA provides language constructs to describe an SLAs ontology. A service object defines an abstraction for all conceptual elements for which SLA parameters can be defined. In the context of Web services, the most detailed concept whose quality aspect can be described separately is the individual service operation (in a binding) described in a WSDL specification” [14 p.13].

Such a service operation is bound to the service’s WSDL specification along with the corresponding SOAP binding and operation. For each service operation SLA parameters and metrics are defined accordingly. In case an SLA defines multiple service operations it is possible to logically group these service operations so that they share the same SLA parameters and metric definitions.

Listing 5 illustrates a sample service definition construct defining one operation which is bound to a specific operation of a specified Web service. The operation contains one SLA parameter which is linked to a specified metric. Details on SLA parameters and metrics are provided in the preceding sections.

```
<ServiceDefinition name="POService">
 <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType"
     name="sendPO">
  <SLAParameter name="p1" type="ExecutionTime" unit="seconds">
   <Metric>ExecutionTimeMetric</Metric>
  </SLAParameter>
  <!-- ... -->
  <Metric name="ExecutionTimeMetric" type="float">
   <Source>MeasurementService</Source>
   <MeasurementDirective xsi:type="Gauge">
      <MeasurementURI>http://example.org/ExecutionTimeMetric
      </MeasurementURI>
   </MeasurementDirective>
  </Metric>
  <!-- ... -->
  <WSDLFile>POService.wsdl</WSDLFile>
  <SOAPBindingName>POServiceBinding</SOAPBindingName>
  <SOAPOperationName>sendPO</SOAPOperationName>
 </Operation>
</ServiceDefinition>
```

Listing 5 WSLA - Service Definition Construct

2.3.1.2 SLA parameter

SLA parameters are the main elements of the service description. They represent the relevant QoS aspects of the respective service operation. SLA parameters correspond to performance and dependability related QoS aspects which have been illustrated in Section 2.5.2 and represent measurable, observable properties of a Web service. Typical examples of SLA parameters are response time, availability or throughput of a Web service.

Listing 6 illustrates a sample SLA parameter construct. The attribute type defines the QoS attribute specified by this SLA parameter. According to the WSLA XML schema this attribute is bound to the type `xsd:string`. In Section 4.1.1.1 a modification of this definition is proposed so that SLA parameters are bound to a set of pre-defined QoS attributes. This eliminates the problem of inhomogeneous SLA parameter definitions throughout different SLAs.

```
<SLAParameter name="p1" type="ExecutionTime" unit="seconds">
   <Metric>ExecutionTimeMetric</Metric>
</SLAParameter>
```

Listing 6 WSLA - SLA Parameter Construct

2.3.1.3 Metric

A metric either contains a function or a measurement directive. The WSLA schema provides a subset of such predefined functions and measurement directives. A function enables the computation of higher-level metrics by the use of lower-level metrics. A measurement directive defines where the information about the different metrics can be obtained. A measurement directive is bound to pre-defined standard measurement directive types (`Counter`, `Gauge`, `ResponseTime`, `SumResponseTime`, `Status`, `InvocationCount`, `StatusRequest`, `Downtime`, `MaintenancePeriodQuery`).

According to the WSLA specification “metrics are the key instruments to describe exactly what SLA parameters mean by specifying how to measure or compute the parameter values” [21 p.27]. This is primary due to the fact that the definition of SLA parameters is not bound to a subset of predefined QoS attributes in the WSLA specification. As already mentioned in the last Section about SLA parameters this problem can be avoided if a predefined subset of QoS attributes is integrated in the WSLA schema.

The definition of metrics and measurement directives is not directly related to the aspect of considering QoS at the choreography description layer as it is not relevant for the mapping of SLA parameters to policy assertions. However the question arises how to integrate SLA parameters (bound to a specific QoS attribute) which does not directly relate to pre-defined measurement directives. The approach which will be further used throughout this work is the usage of the measurement directive type `Gauge` for the corresponding metrics. This measurement directive represents a common directive for all kinds of measurable SLA parameters.

Listing 7 illustrates a sample metric construct using the mentioned measurement directive type `Gauge`. `Source` references the supporting party which is defined in the parties section of the SLA. `MeasurementURI` defines the URI where metrics about SLA parameters can be retrieved.

```
<Metric name="m1" type="float">
   <Source>MeasurementService</Source>
   <MeasurementDirective xsi:type="Gauge">
      <MeasurementURI>http://example.org/ExecutionTimeMetric
      </MeasurementURI>
   </MeasurementDirective>
</Metric>
```
Listing 7 WSLA – Metric Construct

2.3.1.4 Service Level Objectives

"An SLO expresses a commitment to maintain a particular state of the service in a given time period" [13 p.17]. Hence SLO's can be regarded as obligations of the service provider which comprises guarantees for the respective SLA parameters.

Listing 8 provides a simple SLO construct for one SLA parameter. More detailed SLO constructs are provided in Section 4.2. `Obliged` refers to the party that is obliged to maintain the agreed upon state of a service. In typical scenarios this will be accomplished by the service provider. `Validity` defines the time period of the SLO. `Expression` defines the content of the SLO. An expression can either include an element `Predicate` or first order logical operators (`And, Or, Not, Implies`). These logical operators can be used to define nested expression elements. `Predicate` defines the actual assertion on specified SLA parameters. The attribute `xsi:type` refers to pre-defined compare operators (`Less, Equal, GreaterEqual, LessEqual`). `SLAParamter` holds a reference to the SLA parameter defined in the service section of the SLA. Finally, `Value` specifies the SLA parameter's value which is used along with the compare operator to define the assertion.

```
<ServiceLevelObjective name="SLOServiceExecutionTime">
   <Obliged>SupplierCPU</Obliged>
   <Validity>
      <Start>2006-11-30T14:00:00.000+01:00</Start>
      <End>2006-12-31T14:00:00.000+01:00</End>
   </Validity>
   <Expression>
      <Predicate xsi:type="wsla:Less">
         <SLAParameter>p1</SLAParameter>
            <Value>4</Value>
      </Predicate>
   </Expression>
   <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

Listing 8 WSLA - SLO Construct

2.3.2 WS-Agreement

The WS-Agreement specification defines a language and a protocol for establishing agreements between two parties. The latest draft version was published in June 2005 (Version 2005/09). WS-Agreement is being developed in the Global Grid Forum's GRAAP (Grid Resource Allocation Agreement Protocol) working group and is designed "for advertising the capabilities of service providers, creating agreements based on initial offers and for monitoring agreement compliance at run-time. The motivations for the design of WS-Agreement stem out of QoS concerns, especially in the context of load balancing heavy loads on a grid of web service enabled hosts" [7 p.2].

Similar to WSLA the WS-Agreement language specification provides an XML-based schema for defining the overall structure for an agreement document. Such an agreement consists of the following main components:

- Context
- Terms
    - Service Terms
    - Guarantee Terms
- Constraints

The context section includes the definition of the parties involved in the agreement process along with various metadata about the agreement. Among other context-related information this metadata defines the duration of the agreement by specifying time intervals during which the agreement is valid. The terms section is the main part of an agreement and comprises one or more service terms as well as zero or more guarantee terms. Service terms define the services functionalities that will be delivered under the agreement. Service terms include service description, service reference and service property terms. "A service description term (SDT) defines an inline functional full or partial description of a service, whereby the service description content itself is dependant on the particular domain" [22 p.20]. This can be a reference to an existing service, a domain specific description of a service (e.g., a job using the Job Submission Description Language), or a set of observable properties of the service [5 p.4]. Service references contain domain-specific references to existing services by providing endpoint reference. Service properties specify domain-specific aspects (e.g., QoS related aspects) that can be used to define the measurable and exposed properties associated with a service. Guarantee terms specify SLO's about the service description terms. Finally guarantee terms declare the service scope the guarantee applies to, specify SLO's about the service description terms and associate business values to the specified SLO's.

WS-Agreement in general focuses on interaction protocols and the provision and management of templates. Its overall goal is to define a common resource management for Grid environments allowing the advance reservation of those resources [14 p.25].

2.4 Policies

A policy is an assertion about a service that describes one or more characteristics that a service provider instructs a service consumer to follow. In addition to WSDL (which allows the description of functional properties) the WS-Policy framework was developed to enable the declaration of Web service requirements which have to be met by a service consumer to invoke a Web service successfully.

2.4.1 WS-Policy

The WS-Policy framework in the narrow sense comprises two specifications: WS-Policy [23] and WS-PolicyAttachment [24]. The WS-Policy specification itself provides a flexible and extensible grammar for expressing the capabilities and requirements of Web services. This grammar is domain-independent and has to be extended with a domain-specific policy.

A policy expression is an XML Infoset (well-formed XML document following the guidelines in the XML-InfoSet specification [29]) representation of a policy. The WS-Policy language basically declares four elements and one attribute to construct a policy expression.

– Policy
– All
– ExactlyOne
– PolicyReference
– wsp:Optional

The elements `Policy`, `All` and `ExactlyOne` represent policy operators which are used for combining policy assertions, whereby the `Policy` operator play two roles. On the one hand it defines a wrapper element (enclosing a policy expression), on the other hand it represent a synonym for the `All` operator.

Combining policy assertions using the `All` operator means that all the requirements represented by embraced assertions are required. Consequently, using the `ExactlyOne` operator specifies that the policy requires exactly one of the policy assertions to be fulfilled. The `PolicyReference` element can be used to reference a policy expression (either as a standalone policy or within another policy expression). Finally, the `wsp:Optional` attribute declares an optional policy assertion.

All child elements of the logical operators are policy assertions representing domain specific assertions. Policy operators can be nested. Furthermore policy assertions can include nested policy expressions. This allows to creation of flexible and complex policy expressions.

The policy operators listed above refer to the following policy constructs:

– Policy
– Policy Alternative
– Policy Assertion

These policy constructs define the data model of a policy in the following way. A policy expression is an unordered collection of zero or more policy alternatives. A policy alternative is an unordered collection of zero or more policy assertions. A policy assertion represents a domain specific requirement or capability. A domain specific requirement could be an assertion from the security domain specifying a respective security protocol for the interaction. A domain specific capability could be an assertion from the QoS domain specifying a pre-defined response time of the Web service operation.

The definition of a policy expression can be represented either in normal form or an equivalent compact form. The normal form of a policy expression directly maps to the definition of the data model by representing a policy as a collection of policy alternatives and a policy alternative as a collection of policy assertions. This is being accomplished by the use of policy operators listed above. A sample normal form policy construct which highlights the relation to the policy data model is illustrated in Figure 2.

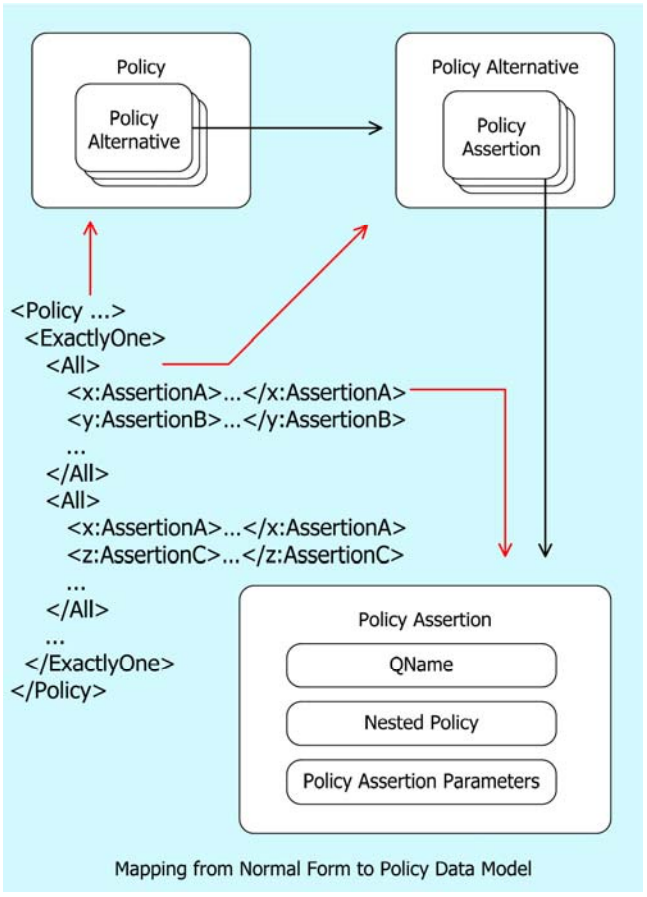

Policy
Policy Alternative
Policy Alternative
Policy Assertion
<Policy ...>
<ExactlyOne>
<All>
<x:AssertionA>...</x:AssertionA>
<y:AssertionB>...</y:AssertionB>
...
</All>
<All>
<x:AssertionA>...</x:AssertionA>
<z:AssertionC>...</z:AssertionC>
...
</All>
...
</ExactlyOne>
</Policy>
Policy Assertion
QName
Nested Policy
Policy Assertion Parameters
Mapping from Normal Form to Policy Data Model

Figure 2 Policy Data Model - Normal Form [18]

The WS-Policy specification provides a framework for the definition of domain specific policies. Two domain-specific policies have been developed so far: WS-SecurityPolicy [25] and WS-ReliableMessagingPolicy [28]. These policies are extensions of the WS-Policy framework for the security and reliable messaging related domain. In the following a policy for the QoS related domain will be proposed.

2.5 Quality of Service Definitions

Quality of Service comprises many different categories, ranging from secure and reliable messaging, as well as performance and dependability related aspects of Web services. Basically, these QoS aspects can be classified in the following way:

- QoS aspects of the Web services stack
- QoS attributes of the QoS model

The first category corresponds to secure and reliable messaging, whereas the second category refers to performance and dependability related QoS aspects. This classification is not the only differentiation possible regarding QoS in general. In [3] QoS aspects are referred to as Web service qualities and categorized according three different views (business, service – and system level). The business level corresponds to quality attributes depending on business values such as service charge, compensation value, penalty rate and reputation. The service-level corresponds to qualities depending on performance and stability whereas the system-level comprises quality attributes related to the operations of Web services (interoperability, manageability, business processing and security). Whereas this view-based categorization seems well to consider the various aspects of non-functional properties of Web services, it is not optimal for the differentiation of QoS in the context of Web service based business process development. Such a differentiation is better provided by the classification provided above. In the following, a definition of both defined QoS categories will be given.

2.5.1 QoS aspects of the Web services stack

The current Web services stack (as shown in Figure 3) defines QoS in the broader sense of security, reliable messaging and transactions. Various specifications (WS-ReliableMessaging, WS-Security, WS-AtomicTransaction, WS-BusinessActivity) have been proposed which define these QoS aspects on the protocol level (security and messaging protocols).

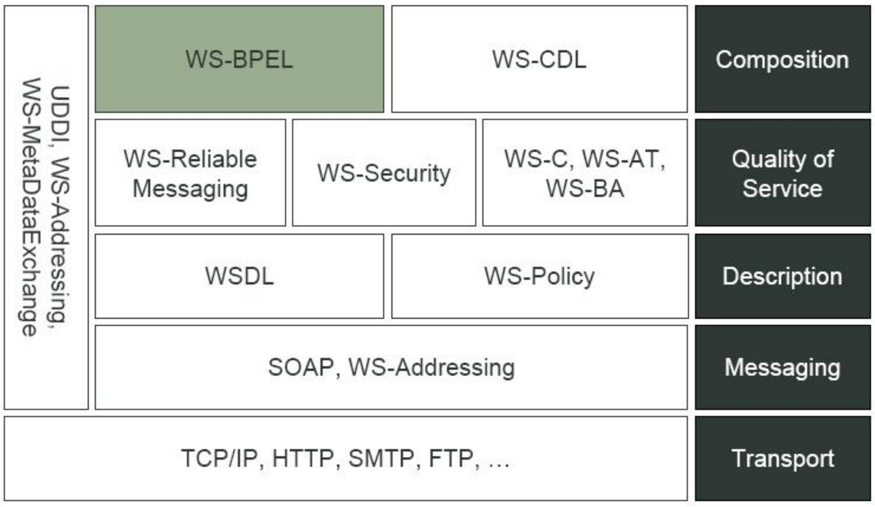


Figure 3 Web services stack[1]

In order to guarantee that the aforementioned QoS aspects are guaranteed throughout a message exchange, security and reliable messaging specific extensions for the WS-Policy framework have been developed. These WS-Policy specifications (WS-SecurityPolicy, WS-ReliableMessaging Policy) basically enable the declaration of security and reliable messaging related Web service requirements which have to be fulfilled by a service consumer in order to invoke a Web service successfully.

2.5.2 QoS attributes of the QoS model

In contrast to the QoS aspects of the Web services stack, the QoS model which is proposed in [4] focus on measurable QoS attributes of Web services. This QoS model can be classified into performance and dependability related QoS aspects.
The following definitions list the QoS attributes relating to these classifications - a more detailed listing of these attributes is provided in [4 p.3ff].

[1] Source: Univ.Prof.Dr. Schahram Dustdar. Lecture material for Service Composition.
URL: http://www.infosys.tuwien.ac.at/teaching/courses/IntAppl/3_Composition.pdf (Last accessed: May, 24 2007)

2.5.2.1 Performance

- Processing Time: The processing time (of a service S and an operation o) $t_p(S,o)$ defines the time needed to actually carry out an operation for a specific request R.
- Wrapping Time: The wrapping time $t_w(S,o)$ defines the time needed to un-wrap (wrap) the XML structure of a received request (send response).
- Execution Time: The execution time $t_e(S,o)$ defines the time a provider needs to finish the processing of a service request. It is specified by the sum of two wrapping times and the processing time: $t_e = t_p + 2 * t_w$
- Latency: The latency $t_l(S)$ of a service S defines the time of a SOAP message to reach its destination.
- Response Time: The response time t_r (S,o) defines the time needed for sending a message M from a given client to a service S until the response R for the message M returns back to the client. It is defined by the sum of two latency times and the execution time: $t_r (S,o) = t_e + 2 * t_l$
- Round Trip Time (t_{rt}): The round trip time t_{rt} defines the overall time from the time a request is issued to the moment the answer is received and processed. It is defined by time-related attributes (on both, the requester and consumer side) in the following way: $t_{rt} = (2 * t_w) + t_l + (t_p + 2 * t_w) + t_l + (2 * t_w)$
- Throughput: Defines the number of Web service requests R for an operation o that can be processed by a service S within a given period of time:

  $$t_p(S,o) = \frac{\#R}{\text{time period (sec)}}$$

- Scalability: Defines the ability of a service S not to get overloaded by a massive number of parallel requests:

  $$sc(S) = \frac{t_{rt}}{t_{rt}\ \text{(Throughput)}}$$

2.5.2.2 Dependability

- Availability: Defines a probability indicator for a service S that it is up and running:

  $$av(S) = 1 - \frac{\text{downtime}}{\text{uptime}}$$

- Accuracy: Defines the success rate of a service S:

  $$ac(S) = 1 - \frac{\#\,\text{failed request}}{\#\,\text{total requests}}$$

- Robustness: The robustness ro(s) defines a probability indicator that a service can react properly to invalid or incomplete input messages.

3 Web service based business process development

Web service based business process development involves two composition techniques of the current Web services stack, namely choreography and orchestration. As illustrated in Section 2.1 choreographies are described using the WS-CDL specification language whereas orchestrations are specified using WS-BPEL. The language constructs of WS-CDL can be mapped to WS-BPEL allowing a choreography description to be transformed into BPEL processes and corresponding WSDL descriptions. Hence, the engineering of Web service based business processes represents a top-down modeling approach of Web services in which private executable business processes are derived from a global choreography model description.
The following sections expatiates the BPEL and WSDL transformation approach mentioned above.

3.1 BPEL Transformation

The transformation of BPEL processes out of a WS-CDL choreography description poses two challenges:

- Determination of CDL elements (ordering structures, activities and definitions) which are relevant for a participant. This aspect will further be referenced as endpoint projection.
- Mapping of relevant CDL elements to corresponding BPEL elements

In the following these aspects of endpoint projection and mapping will be revealed in detail referring to an existing transformation approach. Furthermore a new transformation approach will be described whose concepts will be used for BPEL generation throughout the implementation part of this work.

3.1.1 Endpoint Projection

A WS-CDL conform choreography represents a global description of communication behavior. The projection of such a global description to endpoint processes whose interactions precisely realize the global description is called endpoint projection [8].

The main difficulty in generating BPEL processes out of a WS-CDL choreography description is the endpoint projection of the choreography flow to the respective BPEL flows. The choreography flow comprises ordering structures and activities which control the message exchange and the message processing of the parties involved.

In [12] a transformation program is referenced which is based on XSLT. This XSLT style sheet follows the (simplified) structure illustrated in Listing 9. For each role type the style sheet evaluates if the role type is relevant for the choreography flow. If the role type is relevant the choreography is traversed in a recursive way. For each node a further relevance evaluation is performed by analyzing the attributes `cdl:toRoleTypeRef` and `cdl:fromRoleTypeRef`. If the node is relevant it will be mapped to a corresponding BPEL element.

```
For each roleType
   If choreography.hasRelevantActivities(roleType)
      Traverse choreography
         For each node
            Call Template "flow" (roleType)

Template "flow" (param roleType)
   If node.hasActivities(roleType)
      Map node to BPEL element
         Call Template "flow" (roleType)
:End
```

Listing 9 1:1 Mapping

The above mentioned XSLT-based approach does not fulfill the principles of endpoint projection entirely, because more BPEL elements (structured activities) are generated than necessary. This is due to the fact that all parent nodes are considered for the mappings even if they are not directly relevant for the BPEL process (1:1 mapping). The problem is illustrated in Listing 10 showing a sample choreography flow. This choreography flow defines multiple nested `cdl:sequence` elements. Only the 3rd sequence is relevant for the role types "Customer" and "Manufacturer". The placeholder "---" represents parts of the choreography which are not relevant for those role types. The 1:1 mapping approach would generate BPEL processes with three nested `bpel:sequence` elements (where the last `bpel:sequence` contains the corresponding BPEL activities) because the first and second `bpel:sequence` element were considered to be relevant as well (as they contain relevant BPEL activities too). However, a correct endpoint projection would only generate one `bpel:sequence` element (with corresponding nested BPEL activities).

```
<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>
            </sequence>
         </sequence>
      </sequence>
   </choreography>
</package>
```

Listing 10 Choreography Flow - Example

With respect to endpoint projection a mapping approach is needed which maps only relevant parent nodes (relevance mapping). In the following such a relevance mapping algorithm will be discussed in detail.

3.1.2 Relevance Mapping Algorithm

The relevance mapping algorithm performs a traversing of the choreography analyzing relevant ordering structures (`cdl:sequence`, `cdl:parallel`, `cdl:choice`), workunits (`cdl:workunit`) and basic activities (`cdl:interaction`, `cdl:silentAction`, `cdl:assign`). The relevance of ordering structures will be determined by analyzing nested activities or variable assignments. The relevance of workunits and basic activities will be determined by analyzing role type definitions.

The core concept of the algorithm is a twofold relevance check for ordering structures which will be applied recursively. For each ordering structure that is getting processed a check for relevant descendent and child nodes is performed (Descendent & Child Relevance).

The pseudo code for the relevance mapping algorithm is provided in Listing 11. Initially the root element of the choreography flow will be processed by the function `processChoreography()`. Theoretically the choreography flow could consist of a single basic activity – in this case a mapping to a corresponding BPEL basic activity will be performed. In the common sense the root element of the choreography flow represents a ordering structure (mostly `cdl:sequence`) or a workunit. In this case the following relevance check will be performed by invoking the function `processCDLConstruct()`.

Descendant Relevance

If the ordering structure contains descendant node(s) representing a relevant basic activity or workunit (Descendent Relevance = true) the ordering structure will be considered for further analysis, otherwise the ordering structure will not be further processed.

Child Relevance

If the ordering structure contains child node(s) representing a relevant basic activity, workunit or `cdl:choice` (Child Relevance = true) the ordering structure will be mapped to a corresponding BPEL structured activity. In the case of the ordering structures `cdl:sequence` and `cdl:parallel` the mapping will only be performed if there is more than one relevant child node.

Subsequently the child nodes (ordering structures, workunits and basic activities) of the ordering structure will be enumerated. For each child node the following procedure will be applied:

- If the child node represents a ordering structure further processing will be performed for this ordering structure. In the following the term further processing corresponds to a recursive invocation of the function `processCDLConstruct()`.
- If the child node represents a workunit a further twofold relevance distinction check has to be performed.

1) The workunit is directly relevant for the respective role type (the role type is related to the variable definition(s) in the guard condition of the `cdl:workunit`). In this case the child node will be mapped to a corresponding BPEL structured activity and further processing for the workunit will be performed.
2) The workunit is indirectly relevant for the respective role type (the role type is not related to the variable definition(s) in the guard condition of the `cdl:workunit` but referenced in an interaction activity (`toRoleTypeRef`). In this case no mapping but further processing for the workunit will be performed.

– If the child node represents a basic activity the child node will be mapped to a corresponding BPEL basic activity.

If a ordering structure has relevant descendent nodes but no relevant child nodes (Descendent Relevance = true, Child Relevance = false) further processing of the child nodes (`cdl:sequence` and `cdl:parallel)` will be performed.

The following listing provides the pseudo code for the relevance mapping algorithm.

```
For each roleType {
   If roleType.isRelevant() {
      Generate bpel:partnerLinks
      Generate bpel:variables
      Generate bpel:process
      bpel:process.setRootElement()
      Call processChoreography(cdl:choreography,bpel:process)
   }
}

Function processChoreography(Node choreography, Element rootElement) {
   List = choreography.getChilds()
   For each node in List {
      Switch node.getType()
         Case type="sequence"|"parallel"|"choice"|"workunit"
            Call processCDLConstruct(node,rootElement)
         Case type="interaction"
            Map cdl:interaction to bpelConstruct
            rootElement.add(bpelConstruct)
         Case type="silentAction"
            Map cdl:silentAction to bpelConstruct
            rootElement.add(bpelConstruct)
         Case type="assign"
            Map cdl:assign to bpelConstruct
            rootElement.add(bpelConstruct)
   }
}
Function processCDLConstruct(Node cdlConstruct, Element rootElement) {
   If cdlConstruct.hasRelevantDescendants() {
    //Descendent Relevance = true
      If cdlConstruct.hasRelevantChilds() {
       //Child Relevance = true
         type = cdlConstruct.getType()
         Switch type
            Case "sequence"
```

```
               If cdlConstruct.countRelevantChilds() > 1 {
                  Map cdl:sequence to bpelConstruct
                  rootElement.add(bpelConstruct)
                  bpelConstruct.setRootElement()
               }
            Case "parallel"
               If cdlConstruct.countRelevantChilds() > 1 {
                  Map cdl:parallel to bpelConstruct
                  rootElement.add(bpelConstruct)
                  bpelConstruct.setRootElement()
               }
            Case "choice"
               Map cdl:choice to bpelConstruct
               rootElement.add(bpelConstruct)
               bpelConstruct.setRootElements()
         List = cdlConstruct.getChilds()
         For each node in List {
            type = node.getType()
            Switch type
               Case "sequence"|"parallel"|"choice"
                  Call enumerateCDLConstruct(node,rootElement)
               Case "workunit"
                  relevance = node.getRelevanceType()
                  Switch relevance
                     Case "direct"
                        Map cdl:workunit to bpelConstruct
                        rootElement.add(bpelConstruct)
                        Call enumerateCDLConstruct(node,rootElement)
                     Case "indirect"
                        Call enumerateCDLConstruct(node,rootElement)
                  }
               Case "interaction"
                  If node.isRelevant() {
                     Map cdl:interaction to bpelConstruct
                     rootElement.add(bpelConstruct)
                  }
               Case "silentAction"
                  If node.isRelevant() {
                     Map cdl:silentAction to bpelConstruct
                     rootElement.add(bpelConstruct)
                  }
               Case "assign"
                  If node.isRelevant() {
                     Map cdl:assign to bpelConstruct
                     rootElement.add(bpelConstruct)
                  }
         }
      } // End: relevantChild
      Else {
       //Descendent Relevance = true, Child Relevance = false
         List = cdlConstruct.getChilds()
            For each node in List {
               Call enumerateCDLConstruct(node,rootElement)
            }
      }
   } // End: relevantDescendant
}
```

Listing 11 Relevance Mapping Algorithm

3.1.3 Mapping Rules

Having described the general relevance mapping algorithm element mapping rules for each ordering structure, workunit and basic activity will be provided (see Table 1).

The mapping rules basically follow the mapping rules defined in [12]. There are some differences regarding the mapping of cdl:choice, `cdl:workunit` and `cdl:interaction` which will be highlighted accordingly.

WS-CDL	**BPEL**	**Mapping Conditions**
	Activities	
sequence	sequence	
parallel	flow	
choice	switch pick	nested workunit directly relevant nested workunit indirectly relevant

workunit	case
	
	while
		workunit inside choice
		(block & repeat attribute set to false)
		workunit not inside choice
		(block attribute set to false, repeat set to true)

interaction	invoke
	receive
	|| onMessage
	
	outputVariable
	|| receive
	reply
		action = request, roleType = fromRoleType
		action = request, roleType = toRoleType
		(interaction inside workunit which is inside choice)
		action = respond, roleType = fromRoleType
		(asynchronous messaging)
		action = respond, roleType = toRoleType

silentAction	sequence with nested empty	
noAction	empty	
assign	assign	
perform	---	

Table 1 WS-CDL to BPEL Mapping

The following section provides details for those activities where the mapping requires further differentiation (mapping conditions).

– `cdl:choice`

  This ordering structure requires a differentiation of the nested `cdl:workunit` constructs. If a `cdl:workunit` is directly relevant (Section 3.1.2) for the respective role type a `bpel:switch` will be generated. If a `cdl:workunit` is indirectly relevant a `bpel:pick` will be generated.

– `cdl:workunit`

  If `cdl:workunit` is a child node of `cdl:choice` a `bpel:case` will be generated. In this case the attributes `cdl:block` and `cdl:repeat` are set to `false`. Otherwise a `bpel:while` will be generated. In this case the attribute `cdl:repeat` is set to `true` and `cdl:block` is set to `false`. The case of a blocking condition (`cdl:block` set to `true`) will not be considered as it is not clearly defined how to map this scenario (wait until variable becomes available) to BPEL [9]. In [12] the authors propose to generate a `bpel:switch` if cdl:block is set to `false`. In the choreographies considered throughout this work such a scenario is always related with an enclosing `cdl:switch` construct. Hence, `cdl:workunit` represents the case condition.

– `cdl:interaction`
  Depending on this role type definition and the exchange action (request, respond) four different cases have to be distinguished for the mapping.
  1. "`request` & `fromRoleType`": generate a `bpel:invoke` activity
  2. "`request & toRoleType`": `if cdl:interaction` is defined inside a `cdl:workunit` which is defined inside a `cdl:choice` generate a `bpel:onMessage` activity; otherwise generate a `bpel:receive` activity.
  3. "`respond` & `fromRoleType`": append attribute `outputVariable` to corresponding `bpel:invoke` activity which has been already defined in case 1. This output variable contains the message response.
  4. "respond & toRoleType": generate a bpel:reply activity

  In [12] the authors define a `bpel:receive` in case 3 mentioned above. This mapping is applicable for an asynchronous messaging scenario. In the case of synchronous messaging the `cdl:invoke` activity contains input and output variables. The request message is defined in the input variable and the response message gets stored in the output variable.

In addition to the ordering structures and basic activities listed above the following choreography definitions have to be considered as well.

– `cdl:relationshipType`
  All `cdl:relationshipType` definitions have to be considered where `roleType` is either referenced in 1st or 2nd `cdl:relationshipType/roleType` definition. If `roleType` is referenced in 1st

cdl:relationshipType/roleType definition (attribute typeRef) an element bpel:partnerLink with attribute partnerRole has to be generated; otherwise an element bpel:partnerLink with attribute myRole has to be generated.

– cdl:variable

  All variable definitions have to be considered where roleType is referenced in attribute roleTypes of cdl:variable and which declare an attribute informationType. If informationType defines an XML Schema simple type an element variable with attribute type has to be generated; otherwise an element variable with attribute messageType has to be generated.

3.1.4 Attribute Mapping

Having identified the general relevance and mapping rules for the WS-CDL to BPEL mapping process complementary rules for the attribute mappings are provided. These attribute mapping rules define which attributes of the CDL elements have to be considered in order to define the required attributes of the BPEL elements. The BPEL elements sequence, flow, switch, and pick do not define any attributes; hence these elements will not be further considered.

– All attributes of bpel:partnerLink and bpel:partnerLinkType are derived from the attribute interface of the corresponding cdl:roleType/behavior definition.

– The attribute name of bpel:variable is derived from the identical attribute of cdl:variable. The attributes type and messageType are derived from the attribute type of the corresponding cdl:informationType definition.

– The attribute operation of the BPEL activities bpel:[invoke,receive,reply, onMessage] is derived from the attribute operation of cdl:interaction. The attributes inputVariable, outputVariable and Variable are derived from the attribute variable of cdl:interaction/exchange/send or receive respectively. The attributes partnerLink and portType are derived from the attribute interface of the corresponding cdl:roleType/behavior definition.

- The attribute `expression` of `bpel:assign/copy/from` is derived from the attribute `variable` of `cdl:assign/copy/source`. The attributes `variable` and `part` of `cdl:assign/copy/to` are derived from the attribute `variable` of `cdl:assign/copy/target`.
- The attributes `name` and `condition` of `bpel:[while,case]` are derived from the attributes `name` and `guard` of `cdl:workunit`.

The second step in the twofold top-down modeling process involves the generation of WSDL descriptions which will be described next.

3.2 WSDL Transformation

Each BPEL process is defined by a corresponding WSDL description. Contrary to BPEL transformation the mapping of WS-CDL to WSDL does not require the choreography flow to be analyzed in detail. WSDL descriptions define a static structure which can be extracted from a choreography without considering the choreography flow.

Table 2 summarizes the various elements of such a WSDL description illustrating the corresponding WS-CDL elements. These WS-CDL elements represent the information which has to be extracted from a choreography description in order to generate WSDL descriptions for the respective BPEL processes.

WSDL		**WS-CDL**	
Element	*Attribute*	*Element*	*Attribute*
definitions	xmlns:tns	package	xmlns:tns
	targetNamespace		targetNamespace
	name	behavior	name
message [1...n]	name	exchange	informationType
portType [1...n]	name	behavior	interface
operation [1...n]	name	interaction	operation
input | output	name	exchange	action
	message		informationType
binding [1...n]	name	behavior	name + “Binding”
	type		“tns:” + interface + “Binding”
operation [1...n]	name	interaction	operation
soap:operation	soapAction	behavior	interface namespace +
		interaction	operation

input			
soap:body	namespace	behavior	interface namespace
output			
soap:body	namespace	behavior	interface namespace
service [1...n]	name	behavior	interface + “Service”
port [1...n]	name	behavior	interface + “Port”
	binding		“tns:” + name + “Binding”

Table 2 WSDL - WS-CDL (Element & Attribute Mapping)

Listing 12 illustrates the pseudo code for the WSDL generation process. For each role type of the choreography a WSDL description of its service interface (corresponding to the respective BPEL process) will be generated if this service interface is invoked throughout the choreography flow.

```
For each roleType
   Get behavior interface,name
      Call EvaluateRoleType
      If (interfaceOfRoleTypeUsed)
         Call GenerateWSDL

Function EvaluateRoleType {
   For each channelType
      Get behavior of corresponding roleType
      Get channelType variable
      If channelType variable referenced in [1...n]interaction(s)
         Set interfaceOfRoleTypeUsed=true
}

Function GenerateWSDL {
   Generate output file [cdl:behavior interface]+ ".wsdl"
   Generate wsdl:definitions element

   For each interaction where toRoleTypeRef=roleType
      Generate wsdl:message element

   For each behavior interface of roleType
      Generate wsdl:portType element
      For each interaction where toRoleTypeRef=roleType
         Generate wsdl:operation element
      Generate wsdl:binding element
      For each interaction where toRoleTypeRef=roleType
         Generate wsdl:operation element
      Generate a wsdl:service element
         Generate wsdl:port element
}
```

Listing 12 Pseudo Code - WSDL Generation

4 QoS Integration

The integration of QoS throughout a Web service based business process development scenario requires appropriate techniques to consider QoS at the choreography and orchestration layer. This is due to the fact as neither WS-CDL nor WS-BPEL support the declaration of QoS attributes. Yet, both language specifications can be extended accordingly enabling an integration of existing standards. Considering QoS at the choreography layer can be achieved by the use of Service Level Agreements (SLAs) which focus on performance and dependability related aspects of the QoS model. The integration of QoS at the orchestration layer can be attained by the use of Web service policies. SLAs are a well-established standard and provide proper methods for defining agreements on QoS requirements between the participants involved. On this account SLAs will be used to integrate QoS inside choreography descriptions. Web service policies are state-of-the-art for the definition of non-functional requirements of Web services. In contrast to SLAs − where agreement on QoS requirements is achieved on the business level − Web service policies are more technology oriented which makes it a suitable standard for integrating QoS at the orchestration layer.

In order to map QoS aspects from the choreography to the orchestration layer it is necessary to map SLAs to corresponding Web service policies. However, the current WS-Policy specifications focus on security and reliable messaging related QoS aspects only − performance and dependability related QoS aspects have not been considered yet. Consequently, QoS aspects defined in SLAs can not be mapped to existing WS-Policy specifications (WS-RMPolicy, WS-SecurityPolicy). Such a mapping requires a WS-Policy specification for the QoS domain (WS-QoSPolicy) which is missing in the current WS-Policy framework. Therefore a mapping between these two layers implies an extension of the current WS-Policy framework by defining a policy for the QoS domain.
In the following sections the above illustrated QoS integration process will be discussed in detail.

4.1 QoS at the choreography layer

As mentioned above SLAs will be used to integrate QoS in choreography descriptions. In the following such an SLA integration approach will be highlighted. Prior to this, an evaluation of SLA specification languages will be provided.

4.1.1 Evaluation of SLA Specification Languages

The following evaluation summarizes the relevance and applicability of both specifications languages (WSLA and WS-Agreement) in respect to the QoS integration process. Further aspects of both frameworks (e.g., monitoring in the case of WSLA or provisioning of templates in case of WS-Agreement) have not been considered as these aspects are not directly related to the QoS integration effort examined throughout this work. The focus has been kept on the language specifications of both frameworks.

The WSLA language specification focuses on the definition of SLA parameters. The definition of measurable SLA parameters together with the ability to specify a rich set of SLO's using logical operators directly support the creation of SLAs for QoS attributes of Web services.
As opposed to WSLA, the WS-Agreement language specification itself has little or no support for the definition of such SLA parameters. WS-Agreement provides an umbrella structure that must be complemented by other languages to describe a service or to define guarantees [5 p.4]. To define SLAs in the context of performance and dependability related aspects, WS-Agreement has to be extended regarding the definition of service properties and SLO's. This is due to the fact that WS-Agreement considers the following topics outside the scope of the specification (the specification defines three more out-of-scope topics – only the relevant topics are referenced):

1. Defining domain-specific expressions for service descriptions
2. Defining specific condition expression language for use in specifying guarantee terms
3. Defining specific SLO terms for a specific usage domain

All topics listed above are relevant for specifying SLAs in the context of Web services. The first topic refers to the definition of service parameters and the service

reference. In WSLA these facts are considered through the specification of SLA parameters and metrics along with a binding to an existing WSDL description of the respective Web service. In WS-Agreement this would require an extension of the service description term (SDT) for the specification of WSDL service references. Furthermore the service properties section (contained in a SDT) has to be extended to allow the definition of metric-related information for the specific service parameters.
The second and third topic refers to the definition and grouping of SLO's based on defined service properties. In WSLA these facts are considered through the specification of expressions and logical operators. Expressions define assertions on SLA parameters using compare operators and can be grouped and nested using first order logic. In WS-Agreement this would require an extension to the SLO construct of the guarantee term. Neither expressions nor logical operators are defined in the specification. In [6] an extension of WS-Agreement is illustrated showing by what means the definition of SLAs can be accomplished.

The integration of QoS in Web service based business process development requires the definition of SLAs for Web services. In general, this requirement can be accomplished by both language specifications. Contrary to WS-Agreement WSLA does not have to be extended to enable the definition of SLAs which are used throughout this work. Out of this reason the integration of QoS at the choreography description layer is being accomplished with SLAs that follow the WSLA language specification.

4.1.1.1 WSLA Extension

Without any modification of the WSLA schema an automatic mapping of an SLA parameter to a policy assertion cannot be performed in a satisfying way. A mapping would only be possible if the name of an SLA parameter agrees with the policy assertions defined in the WS-QoSPolicy schema (see Section 4.2.1).

The following modification of the WSLA schema is proposed (The namespace prefix `qosp` refers to the namespace of the WS-QoSPolicy schema):

```
<xsd:complexType name="SLAParameterType">
   <xsd:attribute name="type" type="qosp:QoSAttributes"/>
</xsd:complexType>
```

Listing 13 WSLA Extension - Parameter Type

4.1.2 Integration of SLAs

Having identified WSLA for the definition of SLAs the question remains how to integrate these SLAs at the choreography description layer. As mentioned before not the SLA itself but a reference to the SLA has to be integrated in the choreography description. In accordance to other Web service related specifications the WS-CDL schema enables additionally elements to be defined in the respective CDL constructs. In the following this extension mechanism will be described in detail.

Each element of the WS-CDL schema can be extended by the definition of elements and attributes from other namespaces. All types defined in WS-CDL derive from the `cdl:tExtensibleElements` type which is illustrated in Listing 14. This type allows elements and attributes from other namespaces to be added. Furthermore, it contains the optional description element that is applied to all WS-CDL constructs.

```
<complexType name="tExtensibleElements">
   <sequence>
      <element name="description" minOccurs="0">
         <complexType mixed="true">
            <sequence minOccurs="0" maxOccurs="unbounded">
               <any processContents="lax" />
            </sequence>
            <attribute name="type" type="cdl:tDescriptionType"
             use="optional" default="documentation" />
         </complexType>
      </element>
      <element name="CDLExtension" minOccurs="0" maxOccurs="unbounded">
         <complexType>
            <sequence minOccurs="0" maxOccurs="unbounded">
               <any processContents="lax" />
            </sequence>
         </complexType>
      </element>
   </sequence>
   <anyAttribute namespace="##other" processContents="lax" />
</complexType>

<simpleType name="tDescriptionType">
   <restriction base="string">
      <enumeration value="documentation" />
      <enumeration value="reference" />
      <enumeration value="semantics" />
   </restriction>
</simpleType>
```

Listing 14 WS-CDL - Extensible Element

4.1.2.1 Definition of SLA references

To include an SLA in a choreography description an SLA reference element needs to be defined. Listing 15 illustrates the declaration of such an SLA reference.

The attribute `serviceconsumer` refers to the name of a role type which is defined in the choreography description. This role type corresponds to the SLAs service consumer. The attribute `uri` refers to the location where the SLA is stored and can be accessed.

```
<xs:element name="slaReference">
   <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="serviceconsumer" type="xs:string" use="required"/>
       <xs:attribute name="uri" type="xs:anyURI" use="required"/>
   </xs:complexType>
</xs:element>
```

Listing 15 SLA Reference – Schema Definition

4.1.2.2 Integration of SLA references

To associate an SLA with a particular service the respective SLA reference needs to be added to the behavior element of a participants' role type. The role type corresponds to the service provider and the behavior `interface` attribute corresponds to the service interface. Another possible approach would be the integration of SLA references in the channel type definition. A choreography interaction is always bound to a specific channel which defines exactly one role type representing the receive part in the interaction (service provider). However, an interaction also specifies the participating role types directly, providing information about the interaction `fromRoleType` respective `toRoleType`. Out of this reason integrating SLA references at the behavior element of a participant's role type further simplifies implementation efforts of such an SLA integration scenario.

As illustrated in Listing 14 the WS-CDL schema supports two methods for adding SLA references to CDL constructs (the namespace prefix `qosp` refers to the namespace of the WS-QoSPolicy schema):

1. CDL constructs can be extended using the `CDLExtension` element
2. CDL constructs can be semantically annotated using the `description` element

Basically, both methods can be used for an integration of SLA references. However, the use of the description element seems more appropriate as SLA references can be regarded as semantic annotations of a choreography description. The fact that only one description element can be defined for each CDL construct implies no further restrictions as multiple SLA references can be attached to one description element.

Listing 16 illustrates both extension methods. The attribute `interface` of the element `behavior` identifies a WSDL interface type which is specified in form of a qualified name (the namespace prefix `b2o` refers to an exemplary namespace used in the implementation scenario). A behavior without an `interface` attribute describes a role type that is not required to support a specific Web service interface. This cannot be the case for a role type acting as a service provider. Therefore every role type containing an SLA reference will have a corresponding Web service interface defined. The attribute `type` of the element `description` is bound to one of the following types:

– `documentation`
  defined as plain text or other non-encoded text formats
– `reference`
  defined as URI to a description of the component
– `semantics`
  defined as machine oriented semantic descriptions

```
<!-- use of CDLExtension element -->
<roleType name="ManRoleType">
   <behavior interface="b2o:manInterface" name="ManBehaviour">
      <CDLExtension>
         <qosp:slaReference
            name="SLA1"
            uri="ManufacturerCustomerSLA.xml"
             serviceconsumer="CustRoleType"
          </qosp:slaReference>
      </CDLExtension>
   </behavior>
</roleType>

<!-- use of description element (semantic annotation) -->
<roleType name="ManRoleType">
   <behavior interface="b2o:manInterface" name="ManBehaviour">
      <description type="semantics">
         <qosp:slaReference
            name="SLA1"
            uri="ManufacturerCustomerSLA.xml"
             serviceconsumer="CustRoleType"
          </qosp:slaReference>
      </description>
   </behavior>
</roleType>
```

Listing 16 SLA Integration

Throughout this work the second approach (semantic annotation) is used for the integration of SLAs in a choreography description.

4.2 QoS at the orchestration layer

Following the approach used at the choreography layer an existing Web service standard (WS-Policy) will be used for the integration of QoS at the orchestration layer.

As summarized in Section 2.4 the WS-Policy framework provides a grammar for the definition of domain-specific policies. In the following a QoS domain specific extension of the WS-Policy framework will be provided. Such an extension is the prerequisite for mapping SLAs from the choreography description layer to equivalent policies on the orchestration layer. Subsequently to the different SLA to policy mapping scenarios two policy integration approaches will be evaluated.

4.2.1 Definition of a WS-QoS Policy

The WS-QoSPolicy specification comprises domain-specific policy assertions for the QoS domain. The QoS model described in Section 2.5.2 will provide a basis for the QoS policy assertions. Not all of the attributes identified in the QoS model are definable in advance. They are either dependant on external factors or will be derived from empirical values. The WS-QoSPolicy will focus on attributes which are directly influenced by the service provider. Hence all those QoS attributes will be considered which are relevant in respect to Service Level Agreements.

Table 3 illustrates the relevance of the attributes from the QoS model.

QoS Attribute	**Relevant**	**Reason**
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 factors
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

All QoS attributes marked as relevant will be considered in the WS-QoSPolicy. A service provider can state guarantees on these attributes in an SLA.

Those attributes marked as non-relevant will not be considered based of the following reasons.

- Latency represents an external factor which cannot be influenced by a service provider as it is dependent on the type of network connection the request is sent over. In a choreography scenario Web service requests will be typically sent over the internet - participants will not be connected by an internal WAN/LAN connection but use the internet for message exchanges.
- Response Time subsumes Execution Time and Latency. A service provider can only state a guarantee on the Execution Time for a service request.
- Round Trip time defines the overall time of a service request and depends on Latency, the processing on service consumer and the processing on service provider side. Hence a service provider can not state any guarantee on this value.
- Scalability defines a probability indicator which is dependant on the Round Trip time.
- Accuracy depends on empirical values by evaluating all service invocations during a defined period of time. A service provider cannot state a guarantee on this value at the particular time an SLA will be established.
- Robustness defines a probability indicator which depends on empirical values too.

Consequently the following QoS attributes will be reflected in the WS-QoSPolicy:

- Processing time
- Wrapping time
- Execution time
- Throughput
- Availability

Typically Processing time and Wrapping time will not be explicitly referred to in an SLA. Instead a guarantee on Execution time will be defined.

The WS-QoSPolicy specifies the following assertion model and normative outline.

4.2.1.1 Assertion Model

The WS-QoSPolicy assertion model defines a loose coupling of policy assertions for all relevant QoS attributes of the QoS model (see Table 3).
The structure of SLAs imposes the following three requirements on this assertion model:

1. no root element for the policy assertions is defined
2. policy assertions are not bound to a pre-defined order
3. the same policy assertion must not be defined inside the same policy operator

The first two requirements consider the fact that multiple SLO's can be defined for the same SLA parameter (corresponding to a QoS policy assertion) and that SLO's may contain a nested structure. The definition of a root element and a predefined order for policy assertions would prevent an automatic SLA to policy mapping approach.

Listing 17 illustrates two policy assertions following this assertion model.

```
<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 17 Policy Assertions

4.2.1.2 Normative Outline

The normative outline for the QoS policy assertion is illustrated in Listing 18.

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

Listing 18 Normative Outline

The following describes the normative constraints on the outline listed above:

/qosp:[QoS]Assertion

[QoS] represents a placeholder for one of the following QoS assertions: ProcessingTime, WrappingTime, ExecutionTime, Throughput, and Availability. The exact signification of these assertions is defined in the QoS model (see Section 2.4).

/qosp:[QoS]Assertion/@value

For the `AvailabilityAssertion` the attribute `value` is of type `xs:flow`. For all other assertions the attribute `value` is of type `xs:integer`.

/qosp:[QoS]Assertion/@unit

For all assertions the attribute `unit` is of type `xs:string`.

/qosp:[QoS]Assertion/@predicate

For all assertions the attribute `predicate` is of type `tns:PredicateType`. The namespace-prefix `tns` points to the namespace of the WS-QoSPolicy. The type `PredicateType` is bound to one of the following values: `Greater, Less, Equal, GreaterEqual, LessEqual`.

4.2.1.3 XML Schema

The WS-QoSPolicy is defined by an XML schema. Listing 19 illustrates the relevant part of this schema (the complete schema is listed in the appendix of this work). Each element defines a corresponding QoS policy assertion.

```
<xs:element name="ProcessingTimeAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
              use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="WrappingTimeAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
              use="required"/>
      </xs:complexType>
</xs:element>
<xs:element name="ExecutionTimeAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
              use="required"/>
      </xs:complexType>
```

```
</xs:element>
<xs:element name="ThroughputAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
              use="required"/>
      </xs:complexType>
</xs:element>
   <xs:element name="AvailabilityAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:float" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
              use="required"/>
      </xs:complexType>
</xs:element>
<xs:simpleType name="PredicateType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Greater"/>
         <xs:enumeration value="Less"/>
         <xs:enumeration value="Equal"/>
         <xs:enumeration value="GreaterEqual"/>
         <xs:enumeration value="LessEqual"/>
      </xs:restriction>
</xs:simpleType>
<xs:simpleType name="QoSAttributes">
   <xs:restriction base="xs:string">
       <xs:enumeration value="ProcessingTime"/>
       <xs:enumeration value="WrappingTime"/>
      <xs:enumeration value="ExecutionTime"/>
      <xs:enumeration value="Throughput"/>
      <xs:enumeration value="Availability"/>
   </xs:restriction>
</xs:simpleType>
```

Listing 19 QoS Policy - XML Schema

In addition to the policy assertions the WS-QoSPolicy schema is also used to define a list of QoS attributes. These QoS attributes are referenced in the WSLA schema (see Section 4.1.1.1) for the definition of SLA parameters. As SLA parameters are restricted to the defined QoS attributes all prerequisites for an automatic mapping approach of SLAs to policies are fulfilled

This automatic mapping approach which will be discussed in detail in the following section.

4.2.2 From SLAs to policies

The extension of the WSLA schema to restrict SLA parameters to pre-defined QoS attributes and the proposal of a WS-QoSPolicy to define policy assertions based on QoS aspects form the basis for an automatic mapping of SLAs to policies. The SLA to policy mapping approach comprises the following steps:

- Each SLA will be mapped to a policy

– Each SLA parameter will be mapped to a policy assertion

For each SLA one or more Service Level Objectives (SLO's) about SLA parameters will be defined. These SLA parameters refer to the attributes of the QoS model. Depending on the usage pattern of SLO's the following different SLA scenarios can be distinguished:

– SLA scenario 1:
  One SLO is defined for every single SLA parameter
– SLA scenario 2:
  One SLO comprises multiple SLA parameters
– SLA scenario 3:
  SLA parameters are defined in multiple SLO's

In the following sections these SLA scenarios are described in detail.

4.2.2.1 SLA Scenario 1

Listing 20 illustrates a sample SLA construct which defines two SLO's (service execution time and service throughput), whereby each SLO defines exactly one SLA parameter (e.g., the SLO `SLOServiceExecutionTime` defines the SLA parameter `p1` which corresponds to the type `ExecutionTime`).

```
<SLA>
   <!-- ... -->
   <ServiceDefinition>
      <Operation>
         <SLAParameter name="p1" type="ExecutionTime" unit="seconds">
            <!-- ... -->
         </SLAParameter>
         <SLAParameter name="p2" type="Throughput" unit="requests">
            <!-- ... -->
         </SLAParameter>
         <!-- ... -->
      </Operation>
   <!-- ... -->
   <Obligations>
      <ServiceLevelObjective name="SLOServiceExecutionTime">
         <!-- ... -->
         <Expression>
            <Predicate type="wsla:Less">
               <SLAParameter>p1</SLAParameter>
               <Value>5</Value>
            </Predicate>
         </Expression>
         <!-- ... -->
      </ServiceLevelObjective>
      <ServiceLevelObjective name="SLOServiceThroughput">
         <!-- ... -->
         <Expression>
```

```
            <Predicate type="wsla:GreaterEqual">
               <SLAParameter>p2</SLAParameter>
               <Value>1</Value>
            </Predicate>
         </Expression>
         <!-- ... -->
      </ServiceLevelObjective>
   </Obligations>
</SLA>
```

Listing 20 SLA Construct – Scenario 1

Listing 21 illustrates the pseudo code for this mapping scenario. Exactly one `All` operator will be defined containing one ore more policy assertions. For each `SLO` a policy assertion will be generated. The various attributes for a policy assertion have to be enumerated according to the definition of the `SLAParameter` (defined in the Service Description Term) and corresponding `Predicate` (defined in the `SLO` – linked to a specific `SLAParameter`) attributes.

```
generateElement("wsp:Policy")
generateElement("wsp:All")
Element SLA = getRootElement()
For each SLO in SLA
   Element Expression = SLO.getNode("Expression")
   Call Function QoSAssertion(Expression, SLA)

Function QoSAssertion(Element Expression, Element SLA)
   Element Predicate = Expression.getElement("Predicate")
   String PredicateValue = Predicate.getValue()
   String PredicateType = Predicate.getAttribute("type")
   String SLAParameterName = Predicate.getValueOfElement("SLAParameter")
   Element ServiceDefinition = SLA.getElement("ServiceDefinition")
   For each SLAParameter in ServiceDefinition
      If SLAParameter.getAttribute("name") = SLAParameterName
         String SLAParameterUnit = SLAParameter.getAttribute("unit")
         String SLAParameterType = SLAParameter.getAttribute("type")
   SLAParameterType = generateElement("qosp:" & SLAParameterType)
      SLAParameterType.addAttribute("unit", SLAParameterUnit)
      SLAParameterType.addAttribute("predicate", PredicateType)
      SLAParameterType.addAttribute("value", PredicateValue)
```

Listing 21 Pseudo Code – Scenario 1

Listing 22 illustrates the equivalent policy construct for the SLA construct of Listing 20.

```
<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 22 Policy Construct – Scenario 1

4.2.2.2 SLA Scenario 2

SLA Parameters can be grouped throughout an SLO by using the logical operators `And, Or, Not, Implies`. These logical operators have to be mapped to the WS-Policy operators `All` and `ExactlyOne` respectively.

Combining policy assertions using the `All` operator means that all the behaviors represented by these assertions are required. Policy assertions combined using the `ExactlyOne` operator requires exactly one of the behaviors represented by the assertions. To allow a flexible nesting of QoS assertions the WS-QoSPolicy schema must not define a root element. The following comparison illustrates the equivalent policy operators for the different logical operators of the WSLA specification.

And	→	All
Or	→	ExactlyOne
Not	→	Reverse predicate
Implies	→	ExactlyOne & Reverse predicate

Listing 23 illustrates a sample SLA construct which defines one SLO for the service performance (according to the classification of the QoS model), whereby two different combinations for the SLA parameters `ExecutionTime` and `Throughput` are defined.

```
<SLA>
   <!-- ... -->
   <ServiceDefinition>
      <Operation>
         <SLAParameter name="p1" type="ExecutionTime" unit="seconds">
            <!-- ... -->
         </SLAParameter>
         <SLAParameter name="p2" type="Throughput" unit="requests">
            <!-- ... -->
         </SLAParameter>
         <!-- ... -->
      </Operation>
   <!-- ... -->
   <Obligations>
      <ServiceLevelObjective name="SLOServicePerformance">
         <!-- ... -->
         <Expression>
            <Or>
               <Expression>
                  <And>
                     <Expression>
                        <Predicate type="wsla:Less">
                           <SLAParameter>p1</SLAParameter>
                           <Value>5</Value>
                        </Predicate>
                     </Expression>
                     <Expression>
```

```
                        <Predicate type="wsla:GreaterEqual">
                           <SLAParameter>p2</SLAParameter>
                           <Value>1</Value>
                        </Predicate>
                     </Expression>
                  </And>
               </Expression>
               <Expression>
                  <And>
                     <Expression>
                        <Predicate type="wsla:Less">
                           <SLAParameter>p1</SLAParameter>
                           <Value>7</Value>
                        </Predicate>
                     </Expression>
                     <Expression>
                        <Predicate type="wsla:GreaterEqual">
                           <SLAParameter>p2</SLAParameter>
                           <Value>3</Value>
                        </Predicate>
                     </Expression>
                  </And>
               </Expression>
            </Or>
         </Expression>
         <!-- ... -->
      </ServiceLevelObjective>
   </Obligations>
</SLA>
```

Listing 23 SLA Construct - Scenario 2

Listing 24 illustrates the pseudo code for this mapping scenario. For each `SLO` the number of `Expression` elements will be enumerated. If an `SLO` contains exactly one `Expression` than the same procedure as in Scenario 1 will be applied (function `QoSAssertion`). Otherwise the function `EvaluateExpression` will be called. Throughout this function the nesting level of the root `Expression` element (number of nested `Expression` elements) will be enumerated. If nesting level is null than the function `QoSAssertion` will be called. Depending on the logical operator (`"Not"` or `"Implies"`) a reverse `PredicateType` will be defined. If nesting level is greater null than the corresponding logical operator will be determined. Depending on the logical operator a certain policy operator will be generated – furthermore the function `EvaluteExpression` will be called recursively for each nested `Expression` element.

```
generateElement("wsp:Policy")
generateElement("wsp:All")
Element SLA = getRootElement()
For each SLO in SLA
   Element Expression = SLO.getElement("Expression")
   If SLO.countNodes("Expression") = 1
      String PredicateType = Expression.getValueOfElement("Predicate")
      Call Function QoSAssertion(Expression, PredicateType, SLA)
   If SLO.countNodes("Expression") > 1
      Call Function EvaluateExpression(Expression, "", SLA)
```

```
Function QoSAssertion(Element Expression, String PredicateType, Element SLA)
   Element Predicate = Expression.getElement("Predicate")
   String PredicateValue = Predicate.getValue()
   String PredicateType = Predicate.getAttribute("type")
   String SLAParameterName = Predicate.getValueOfElement("SLAParameter")
   Element ServiceDefinition = SLA.getElement("ServiceDefinition")
   For each SLAParameter in ServiceDefinition
      If SLAParameter.getAttribute("name") = SLAParameterName
         String SLAParameterUnit = SLAParameter.getAttribute("unit")
         String SLAParameterType = SLAParameter.getAttribute("type")
   SLAParameterType = generateElement("qosp:" & SLAParameterType)
      SLAParameterType.addAttribute("unit", SLAParameterUnit)
      SLAParameterType.addAttribute("predicate", PredicateType)
      SLAParameterType.addAttribute("value", PredicateValue)

Function EvaluateExpression(Element Expression, String LogicalOperator,
                            Element SLA)
   If Expression.countNodes("Element") = 0 //nesting level = 0
      Element Predicate = Expression.getElement("Predicate")
      String PredicateType = Predicate.getAttribute("type")
      If LogicalOperator = "Not" | "Implies"
          If PredicateType = "Greater"
            Call QoSAssertion(Expression, "LessEqual")
         If PredicateType = "Less"
            Call QoSAssertion(Expression, "GreaterEqual")
          If PredicateType = "GreaterEqual"
            Call QoSAssertion(Expression, "Less")
          If PredicateType = "LessEqual"
            Call QoSAssertion(Expression, "Greater")
      Else
          Call QoSAssertion(Expression, PredicateType, SLA)
   If Expression.countNodes("Element") > 0 //nesting level > 0
      Element LogicalOperatorNode = Expression.getChildElement()
      If LogicalOperatorNode.getType() = "And"
         generateElement("wsp:All")
         For each Expression in LogicalOperatorNode
            Call EvaluateExpression(Expression, "And", SLA)
      If LogicalOperatorNode.getType() = "Or"
         generateElement("wsp:ExactlyOne")
         For each Expression in LogicalOperatorNode
            generateElement("wsp:All")
            Call EvaluateExpression(Expression, "Or", SLA)
      If LogicalOperatorNode.getType() = "Not"
         generateElement("wsp:All")
         For each Expression in LogicalOperatorNode
            Call EvaluateExpression(Expression, "Not", SLA)
      If LogicalOperatorNode.getType() = "Implies"
         generateElement("wsp:ExactlyOne")
         For each Expression in LogicalOperatorNode
            generateElement("wsp:All")
            Call EvaluateExpression(Expression, "Implies", SLA)
```

Listing 24 SLA Mapping – Scenario 2

Listing 25 illustrates the equivalent policy construct for the SLA construct of Listing 23.

```
<wsp:Policy>
   <wsp:All>
      <wsp:ExactlyOne>
         <wsp:All>
            <qosp:ExecutionTimeAssertion unit="seconds"
             predicate="Less" value="5" />
            <qosp:ThroughputAssertion unit="requests"
             predicate="GreaterEqual" value="1" />
         </wsp:All>
         <wsp:All>
            <qosp:ExecutionTimeAssertion unit="seconds"
             predicate="Less" value="7" />
            <qosp:ThroughputAssertion unit="requests"
             predicate="GreaterEqual" value="3" />
         </wsp:All>
      </wsp:ExactlyOne>
   </wsp:All>
</wsp:Policy>
```

Listing 25 Policy Construct – Scenario 2

4.2.2.3 SLA Scenario 3

SLA parameters may be defined in multiple SLO’s. For each SLO a time period has to be specified. Therefore it would be possible to specify multiple SLO’s (with corresponding SLA parameters) for different time periods (as illustrated in Listing 26). However, a more useful approach would be the definition of timeslots (i.e. the SLA parameter `ExecutionTime` must be of value x during peak-hours). Yet, this kind of scenario is not supported by the WSLA specification.

```
<SLA>
   <!-- ... -->
   <ServiceDefinition>
      <Operation>
         <SLAParameter name="p1" type="ExecutionTime" unit="seconds">
            <!-- ... -->
         </SLAParameter>
         <!-- ... -->
      </Operation>
   <!-- ... -->
   <Obligations>
      <ServiceLevelObjective name="SLOServiceExecutionTime">
         <!-- ... -->
         <Validity>
            <Start>2007-01-01T00:00:00.000+01:00</Start>
            <End>2008-01-01T00:00:00.000+01:00</End>
         </Validity>
         <Expression>
            <Predicate type="wsla:Less">
               <SLAParameter>p1</SLAParameter>
               <Value>5</Value>
            </Predicate>
         </Expression>
         <!-- ... -->
      </ServiceLevelObjective>
      <!-- ... -->
   </Obligations>
</SLA>
```

Listing 26 SLA Construct - Scenario 3

4.2.3 Integration of policies

The definition of a QoS policy and the definition of SLA mapping rules (which enable the generation of policies out of SLAs) are the fundamental concepts for considering QoS in a Web service based business process development scenario. Yet, the question remains how to integrate the generated QoS policies at the orchestration layer. With regard to the top-down modeling approach of Web services, two integration approaches can be differentiated. Policies can either be attached to service descriptions (WSDL) or can be integrated in BPEL processes. Figure 4 summarizes the policy mapping and integration process. In the following sections both integration approaches are described in detail. Furthermore an evaluation of both approaches with regards to the aspects of relevance and differentiation is provided.

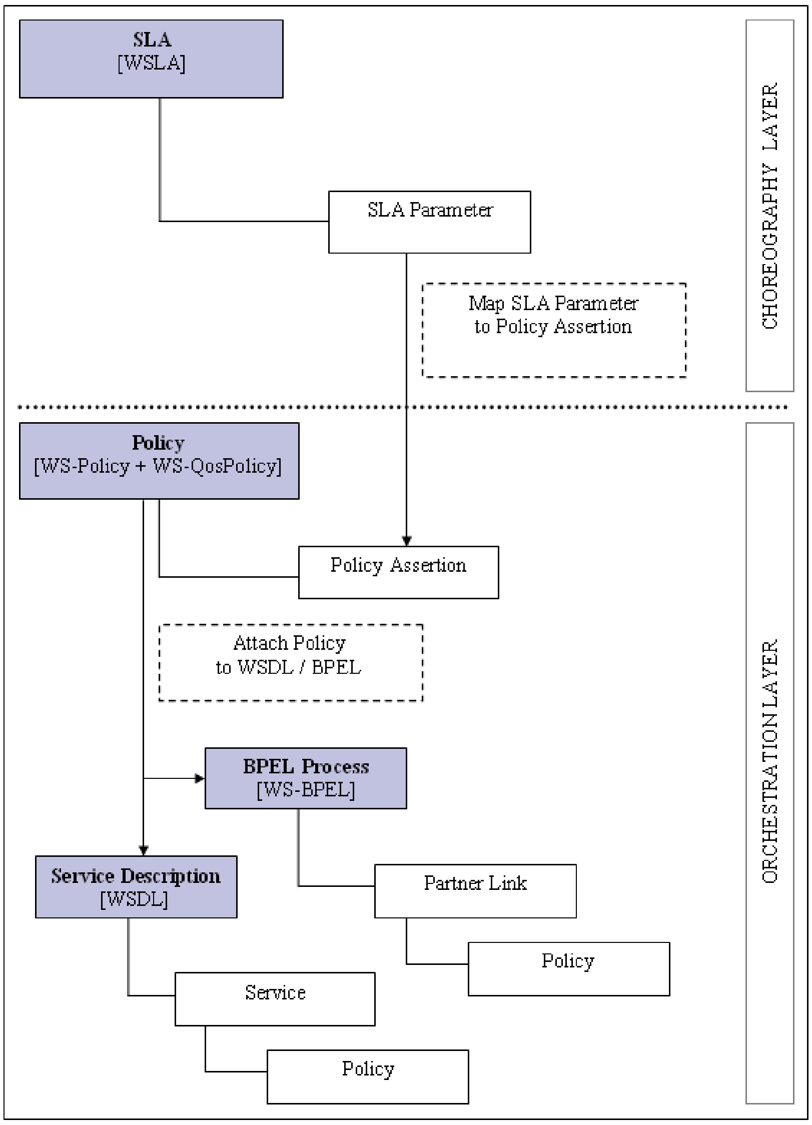


Figure 4 WS-QoSPolicy Mapping & Integration

4.2.3.1 WSDL Integration

As shown in Section 4.1 the core WS-Policy framework comprises WS-Policy and WS-PolicyAttachment. The WS-PolicyAttachment specification defines mechanisms for associating policy expressions with WSDL and UDDI descriptions [24]. UDDI is not directly related to the Web service based business process development scenario as the definition of UDDI description cannot be performed automatically; hence the focus is laid on the WSDL integration mechanism.

A policy expression can be attached to four different policy subjects (entities with which a policy can be associated) in a WSDL service description:

1. Service Policy Subject
2. Endpoint Policy Subject
3. Operation Policy Subject
4. Message Policy Subject

The scope of a policy expression is dependent on the level of the policy subject (to which a policy expression is applied to).

In the context of the Web service based business process development the attachment of policy expressions should be applied to the service policy subject. This assumption is based on the following considerations:

- The definition of an SLA is not restricted to a single operation − an SLA may also be applied to multiple operations of the same service interface. Considering this aspect only, policy expressions should be applied either to an operation policy subject (if the SLA references one operation only) or to an endpoint policy subject (if the SLA references multiple operations).
- However, the Web service based business process development scenario requires a further aspect to be taken into account. The integration of SLAs in choreography descriptions is being accomplished by attaching SLA references to the behavior of a role type which always corresponds to the service interface of the respective service provider. This aspects requires the policy expression to be attached to the layer of the service policy subject which corresponds to the `wsdl:service` element.

Listing 27 illustrates a sample WSDL construct with an attached policy according to the attachment rules mentioned above.

```
<definitions xmlns="..." xmlns:soap="..." xmlns:wsp="...">
   <!-- ... -->
   <portType name="xs:NCName">
      <!-- ... -->
   </portType>
   <binding name="xs:NCName" type="xs:anyURI">
      <!-- ... -->
   </binding>
   <service name="xs:NCName">
      <wsp:PolicyReference URI="xs:anyURI" required="true"/>
      <port name="xs:NCName" binding="xs:QName">
         <soap:address location="xs:anyURI"/>
      </port>
   </service>
   <!-- ... -->
   <wsp:Policy xmlns:qosp="..." xmlns:wsu="..." wsu:Id="xs:QName">
      <!-- ... -->
   </wsp:Policy>
</definitions>
```

Listing 27 WSDL - Example

The `wsdl:service` element actually contains a policy reference. This reference is specified by the attribute `URI` which can either point to a location inside the WSDL definition or refer to an external policy document.

In the case of a locally referenced policy the attribute `URI` is defined in the following form: `URI="#xs:NCName"` (e.g., `URI="#Policy1"`).

In the case of an external policy document the attribute "`URI` is defined in the following form:

`URI="xs:anyURI#NCName"` (e.g., `URI="http://example.org/#Policy1"`).

The policy expression itself contains an attribute `wsu:Id` which assigns an ID value as an URI (e.g., `wsu:Id="Policy1"`).

Including policies in WSDL is rather a static process. If the policy of a Web service will gets redefined, the WSDL will have to be redefined as well. Therefore policies should be attached to WSDL by referencing to an external URL where the corresponding policy is located.

4.2.3.2 BPEL Integration

The WS-PolicyAttachment specification defines two ways for associating policies with a policy subject. On the one hand policies may be attached to WSDL (as illustrated in the preceding section) on the other hand policies may be integrated in UDDI. However, in the context of Web service based business process development it is more reasonable to attach policies to the BPEL processes of the respective choreography participants (this prediction will be discussed in detail in the following section).

Similar to WS-CDL and WSDL, the WS-BPEL specification (BPEL4WS 1.1 as well as WS-BPEL 2.0) supports language extensibility by allowing elements from other namespaces to appear within WS-BPEL defined elements. This extensibility mechanism basically enables the integration of policies inside BPEL processes.

All types defined in WS-BPEL derive from the `bpws:tExtensibleElements` type which is illustrated in Listing 28. Compared to WS-CDL no root element has to be defined which contains the actual extension elements. Furthermore there is no restriction on the count of extension elements.

```
<complexType name="tExtensibleElements">
   <sequence>
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded"
       processContents="lax"/>
   </sequence>
   <anyAttribute namespace="##other" processContents="lax"/>
 </complexType>
```

Listing 28 WS-BPEL - Extensible Element [20]

Contrary to WSDL no specifications exist which illustrates such policy integration. In the following an approach will be discussed which shows by what means policies may be attached to BPEL processes. As partner links play an essential part in this approach basic information about partner links will be provided first.

Partner Links

The following samples illustrate the use of partner links in synchronous and asynchronous messaging scenarios (a general definition about partner links is provided in Section 2.2).

Synchronous messaging corresponds to the request-reply message exchange pattern. The service consumer invokes an operation (e.g., `"PORequest"`) on the service interface of the service provider. The `invoke` activity contains input and output variables – the request message is defined in the input variable and the response message gets stored in the output variable. Hence, on the side of the service consumer only one activity takes place. On the side of the service provider two activities are involved. The request message is received by the corresponding `receive` activity; the response message is send back to the service provider by initializing a `reply` activity. All activities on the service consumer and service provider side are bound to the same partner link type (and port type). In accordance to the definition above the partner link of the service consumer contains a `partnerRole` attribute, whereas the partner link of the service provider contains a `myRole` attribute.

```
<!-- Service Consumer -->
<process>
   <partnerLinks>
      <partnerLink name="POService" partnerLinkType="ns1:POServiceLT"
             partnerRole="POServiceRole"/>
   </partnerLinks>
   <!-- ... -->
   <sequence>
      <!-- ... -->
      <invoke inputVariable="PORequest" operation="PORequest"
             outputVariable="POResponse" partnerLink="POService"
             portType="ns1:POService"/>
   </sequence>
</process>

<!-- Service Provider -->
<process>
   <partnerLinks>
      <partnerLink myRole="POServiceRole" name="POService"
             partnerLinkType="ns1:POServiceLT"/>
   </partnerLinks>
   <!-- ... -->
   <sequence>
      <receive createInstance="yes" operation="PORequest"
             partnerLink="POService" portType="ns1:POService"
             variable="PORequest"/>
      <reply operation="PORequest" partnerLink="POService"
             portType="ns1:POService" variable="POResponse"/>
   </sequence>
</process>
```

Listing 29 Synchronous Messaging

The difference of an asynchronous messaging scenario to the synchronous messaging scenario illustrated above is that the service provider invokes a callback operation on the service consumer to send the response message. Subsequently the activities are bound to different port types, however only one partner link is used on both sides. This is due to the fact, that the partner link definitions contain two roles – `myRole` as well as the `partnerRole`.

```
#Service Consumer
<process>
   <partnerLinks>
      <partnerLink myRole="POCallbackServiceRole"
             name="POCallbackService"
             partnerLinkType="ns1:POCallbackServiceLT"
             partnerRole="POServiceRole"/>
   </partnerLinks>
   ...
   <sequence>
      ...
      <invoke inputVariable="PORequest" operation="PORequest"
             partnerLink="POCallbackService" portType="ns1:POService"/>
      ...
      <receive operation="POResponse" partnerLink="POCallbackService"
             portType="ns1:POCallbackService" variable="POResponse"/>
      ...
   </sequence>
</process>
#Service Provider
<process>
   <partnerLinks>
      <partnerLink myRole="POServiceRole" name="POService"
              partnerLinkType="ns1:POServiceLT"
              partnerRole="POCallbackServiceRole"/>
   </partnerLinks>
   ...
   <sequence>
      ...
      <receive createInstance="yes" operation="PORequest"
              partnerLink="POService" portType="ns1:POService"
              variable="PORequest"/>
      <invoke inputVariable="POResponse" operation="POResponse"
              partnerLink="POService" portType="ns1:POCallbackService"/>
   </sequence>
</process>
```

Listing 30 Asynchronous Messaging

Policy Subject

In WSDL a policy expression may be attached to four different policy subjects – as illustrated in the last section the specific policy subject `wsdl:service` should be used in the Web service based business process development scenario.
In BPEL two possible policy subjects can be identified in the first place - sorted by their scopes throughout a BPEL process:

– Activities
– Partner Links

The information about interactions between participants in a choreography description (involving the invocation of the same service interface) is mapped to partner links in the corresponding BPEL processes. Each BPEL activity is thereby related to a specific partner link. Considering this the use of activities as policy subjects is not a feasible approach. For each activity which references a specific partner link (whose corresponding service interface contains an SLA), a policy reference would have to be attached accordingly. This problem of multiple policy references is prevented by the use of partner links as policy subjects.
Having identified partner links as the relevant BPEL construct for the attachment of policies another important aspect regarding the policy subject has to be taken into account. On the level of BPEL processes two differentiations about policy integration can be made:

1. Integration of policies at the side of the service provider

2. Integration of policies at the side of the service consumer

Referring to this integration possibilities the following aspects regarding messaging scenarios have to be considered.
In the case of synchronous messaging no additional information exists on the side of the service provider about the service consumer which has initialized the service invocation. This is due to the fact that the partner link of the service provider does not contain any service consumer related details (illustrated in Listing 29). In the case of asynchronous messaging the service provider does have information about the service consumer - otherwise the callback operation to the service consumer could not be established. This is due to the fact that the partner link of the service provider does contain a `partnerRole` attribute with details of the service provider as illustrated in Listing 30.
It's a matter of fact that the restrictions of synchronous messaging do not apply to the service consumer. As the service consumer initiates a service invocation on the service provider its partner links always contain the relevant information about the service provider.
Considering these circumstances the following relevant policy subject for policy integration in BPEL can be identified: The `partnerLink` construct in the BPEL processes of the service consumer

Listing 31 illustrates a sample BPEL construct with an attached policy. The namespace prefix `qosp` refers to the WS-QoSPolicy schema which is further used for the definition of policy assertions. The ns-prefix `wsu` refers to the WS-SecurityUtility schema which is used for the declaration of `wsu:Id`. This attribute corresponds to the name of the SLA (from which the policy was derived) whereas the attribute `qosp:operation` declares the respective operation which is referenced in the SLA. A complete BPEL process along with an integrated policy is provided in Appendix E of this work.

```
<!-- Service Provider -->
<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 31 WS-BPEL - Example

4.2.4 Evaluation

Having described the WSDL and BPEL integration approach for attaching policy references at the orchestration layer the question remains which approach is more reasonable in a Web service based business process development scenario. In the following an evaluation of both approaches for certain key aspects will be given. These key aspects include relevance and differentiation of policy references. Concerning these key aspects the relation of BPEL and WSDL is an important aspect - therefore this relation will be described first.

On the choreography layer an SLA reference is attached to the behavior interface of a role type which corresponds to the service interface of the respective choreography participant. The operations of this service interface correspond to the different receive activities of the participant's BPEL process and are exposed through a WSDL description. With regards to Web service based business process development the BPEL processes of the choreography participants are composite Web services which are described in WSDL.

The integration of policies is being accomplished at different sides, either at the service provider or service consumer side. In the case of WSDL integration it is obvious that policy references have to be attached to the WSDL descriptions of the service provider. In the case of BPEL integration policy references should be attached to the BPEL processes of the service consumer as illustrated in the preceding section.

The following principles regarding relevance and differentiation in respect to policy integration can be disposed:

– Relevance:
  The scope of the respective policy must ensure that the policy only involves parties which are referenced in the corresponding SLA.

– Differentiation:
  The relevance aspect implies that a differentiation on the agreed service level must be ensured in the case that a party (service provider) is involved in multiple SLAs throughout the same choreography scenario.

In order to examine the accordance of both integration approaches in relation to these principles the following sample scenario is provided.
This scenario involves three service consumers which both interact with the same service provider. Two service consumers have established an SLA with the service provider. The SLAs are mapped to respective policies and integrated at the orchestration layer.

Attaching these policies to the WSDL description of the service provider violates both principles for relevance and differentiation:

– The invocation of service operations is always subject to a policy (even for service consumers without corresponding SLAs).
– The service provider can not differentiate between the relevant policies if multiple policies are attached to the same WSDL description. This is due to the fact that policies do not contain information about the relevant parties. Hence, if a service consumer invokes a service the service provider cannot determine which policy this service interaction is subject to.

These restrictions do not apply if policies are attached to the BPEL processes of the service consumers. The respective service consumer can always differentiate which policy a service invocation is subject to. This is due to the fact that service invocations are bound to partner links which in turn contain exactly one policy reference (if an SLA between the service consumer and the provider of the invoked service exists). In the case that a partner link is used for multiple interactions involving the invocation of different service operations (e.g., the service of a service provider comprises multiple operations which are invoked by a service consumer) a differentiation regarding the policy scope is guaranteed as well because the operation name is referenced in the respective policy.

5 Implementation

In addition to the theoretical part of this work an implementation has been designed as a proof of concept. In order to perform an evaluation of this implementation a use case scenario has been developed representing a characteristic Web service based business process scenario.

In the following an overview of the system architecture and used technologies will be provided. Furthermore the functionality of the implementation will be demonstrated. Subsequently the use case scenario will be illustrated. Finally the implementation will be applied to this use case and the results will be evaluated accordingly.

5.1 System Overview

The implementation was designed using Java along with XSLT stylesheets and comprises two main blocks which represent the different processing phases. These processing phases (further detailed in Section 5.2) include the editing of a choreography description to manipulate SLA references and the generation of BPEL processes and WSDL descriptions. An overview of the main blocks of the system architecture and the two different phases of the QoS integration approach is depicted in Figure 5.

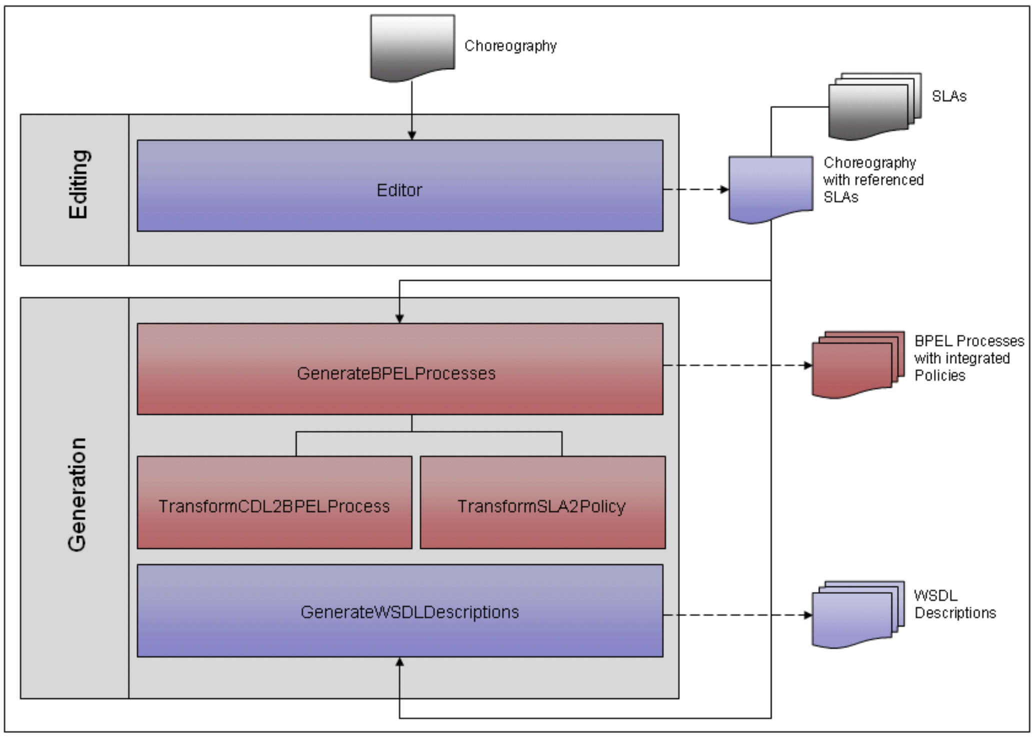


Figure 5 System Architecture

The QoS integration process requires an implementation which addresses the following challenges:

– XML processing
  refers to the navigation, modification and transformation of XML documents.
– XML presentation
  refers to the presentation of XML documents in a well arranged and dynamic form, allowing XML content to be modified accordingly.

5.1.1 XML Processing

XML processing is performed using the dom4j API. Dom4j [33] integrates with DOM (Document Object Model) and SAX (Simple API for XML). However, unlike DOM and SAX (being platform independent) dom4j is a simpler, lightweight API which has been designed especially for the Java programming language making extensive use of standard Java APIs such as the Java 2 collection API. Additional features of dom4j are full XPath [31] support and the integration of XSLT [30] using the JAXP standard APIs.

Listing 32 provides an exemplary source code listing illustrating selective aspects of XML processing in dom4j which are used in the implementation.
In this sample an XML file representing a choreography description gets parsed and is stored in dom4j's `Document` structure. Subsequently a dom4j XPath expression is constructed which returns all role type definitions of the choreography. As elements in a choreography description are defined in a specific namespace this namespace is associated with a namespace prefix which is further used throughout the XPath expression. The XPath expression is applied to the choreography document and returns a Java `List` containing the various `roleType` elements. Finally, the `List` is iterated using Java `Iterators`; each `roleType` element is added to the element `package` and a specific `roleType` element is removed from the `List`. Removing an element from the `List` will remove the element from the corresponding `Document` structure as well. This concept of a backed `List` where modifications to the List are reflected back into to `Document (Element)` allows the manipulation of XML data structures using the Java Collection API.

```
SAXReader reader = new SAXReader();
File file = new File("BuildToOrder.cdl")
Document choreography = reader.read(file);
HashMap map = new HashMap();
map.put("cdl", "http://www.w3.org/2005/10/cdl");
XPath xpath = DocumentHelper.createXPath("cdl:package/cdl:roleType");
xpath.setNamespaceContext(new SimpleNamespaceContext(map));
List roleTypesList = xpath.selectNodes(choreography);
Element package = DocumentHelper.createElement("package");
for (Iterator iter = roleTypesList.iterator(); iter.hasNext(); ) {
       Element roleType = (Element) iter.next();
       root.add(roleType);
       if (roleType.valueOf("@name").equalsIgnoreCase("TestRole"))
              roleTypesList.remove(roleType);
}
```

Listing 32 dom4j - XML Processing

As mentioned above dom4j supports XSLT using JAXP (Java API for XML Processing). JAXP contains an XSLT interface which enables transformations on an XML document without declaring details of a specific XSLT parsing implementation. Listing 33 provides a sample source code listing illustrating an XSLT transformation in dom4j.

```
TransformerFactory factory = TransformerFactory.newInstance();
Transformer transformerWSDL = factory.newTransformer(new
StreamSource("GenerateWSDLDescriptions.xslt"));
DocumentSource source = new DocumentSource(choreography);
StreamResult result = new StreamResult(new File(outputDirectory));
transformerWSDL.transform(source,result);
```

Listing 33 dom4j - XSLT Transformation

5.1.2 XML Presentation

XML presentation is performed using the Swing component `JTree`. The relevant parts (role type definitions) of the input document (choreography description) are displayed in a tree-based hierarchy (Figure 6). This is accomplished by converting an XML element (along with all its child elements) into a Swing `DefaultMutableTreeNode` which is then added to the model of the `JTree`.

Listing 34 depicts the source code [34] for transforming the element `roleTypes` into a Swing `DefaultTreeModel` which is then used to build the `JTree`.

```
XMLEditorTree xmltree;
XMLEditorTreeNode root = new XMLEditorTreeNode("", "");
DefaultTreeModel model = new DefaultTreeModel(root);
root.add(buildChildren(roleTypes));
xmltree.setModel(model);

private XMLEditorTreeNode buildChildren(Element e) {
       XMLEditorTreeNode me = new XMLEditorTreeNode(e.getName(), e);
       XMLEditorTreeNode child;
       List c = e.elements();
       for (int i=0; i<c.size(); i++) {
              Element ec = (Element)c.get(i);
              child = buildChildren(ec);
              me.add(child);
       }
       return me;
}
class XMLEditorTreeNode extends DefaultMutableTreeNode {
       public String _name;
       public Object _value;
       public XMLEditorTreeNode(String name, Object value)
       { _name = name; _value = value; }
       ...
}
class XMLEditorTree extends JTree {
       public XMLEditorTree()  {
              super();
              setRootVisible(false);
              setExpandsSelectedPaths(true);
              setShowsRootHandles(true);
              setExpandsSelectedPaths(true);
              setScrollsOnExpand(true);
       }
       ...
}
```

Listing 34 Transform XML element to JTree model [34]

The GUI and the editing part of this implementation (as illustrated in Figure 5) is based on the Swing and JDOM XML Editor presented in [34] which allows the manipulation of arbitrary XML documents.

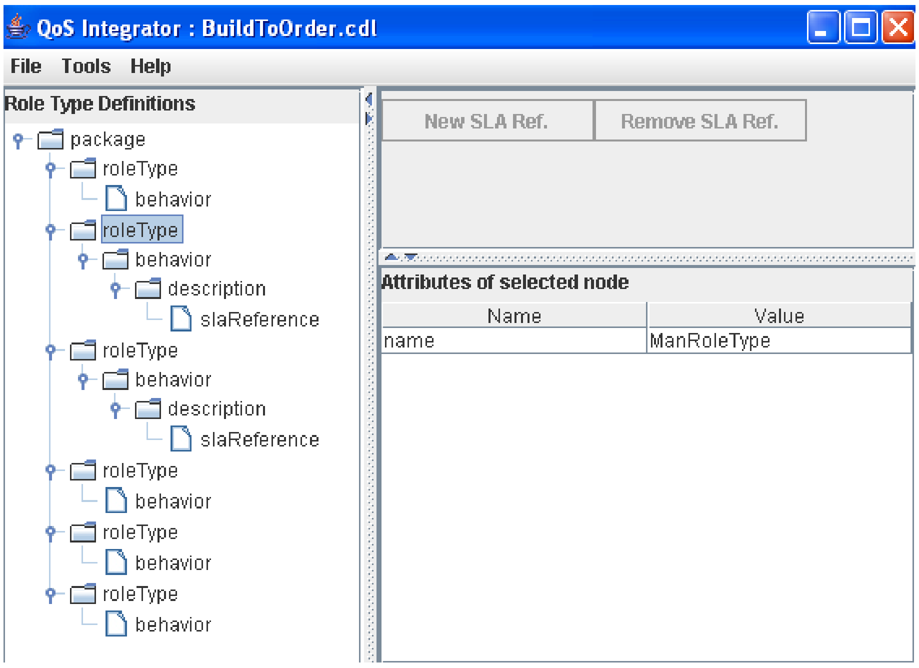
5.2 QoS Integration Process

In the following the functionality of the implementation will be presented by illustrating the QoS integration process. Required input files for the implementation are a WS-CDL conform choreography description and WSLA based Service Level Agreements between the choreography participants.

5.2.1.1 Display of Role Type Definitions

Following the choreography import the role type definitions of the choreography will be extracted and displayed in a tree-based structure. This tree defines a top-level node `package` which comprises the `roleType` nodes of the choreography description. Each `roleType` node contains a node `behavior`. If a role type definition already defines an SLA reference the `behavior` node will include a child node `description` comprising an additional node `slaReference`. The attributes of the currently selected node will be displayed in a table.

Figure 6 depicts the presentation of the role type definitions after the import of a choreography has been performed.



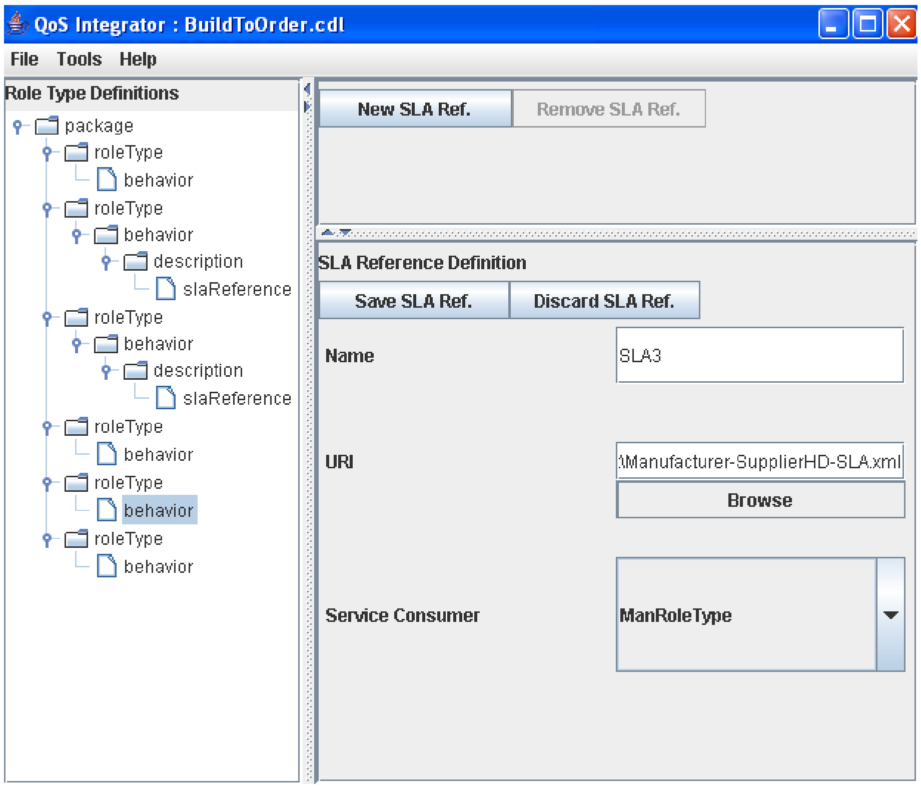
Name	Value
name	ManRoleType

Figure 6 QoS Integrator – Role Type Definitions

5.2.1.2 Manipulation of SLA References

The manipulation of SLA references comprises the addition, modification and deletion of SLA references to role type definitions. As defined in Section 3.2 SLA references have to be added to the `behavior` node of a role type definition. In order to avoid SLA references to be added at the wrong position the button for the definition of new SLA references (`New SLA Ref.`) will only be enabled if the `behavior` node of a role type definition is selected. Similar the button for the deletion of SLA references (`Remove SLA Ref.`) will only be enabled if an `slaReference` node is selected. Existing SLA references can be modified by selecting the respective `slaReference` node; the attributes displayed can then be modified accordingly.

Figure 7 depicts the definition of a new SLA reference. In order to facilitate information entering a list of all role types of the choreography is provided for the definition of the service consumer. If the corresponding SLA is stored locally the URI information can be retrieved by file browsing. Otherwise the URI has to be filled in manually. This is the case if the URI corresponds to a web address. If the SLA reference is saved (`Save SLA Ref.`) a `description` node (containing the `slaReference` node) will be added to the respective `behavior` node. However, the `description` node will only be added if no other SLA reference already exists for that role type definition. Otherwise the `slaReference` node will be added to the existing `definition` node. This implementation scenario corresponds to the fact, that multiple SLA references (encapsulated in a `description` element) can be added to a role type definition (see Section 3.2).



Name: SLA3

URI: \Manufacturer-SupplierHD-SLA.xml

Service Consumer: ManRoleType

Figure 7 QoS Integrator – SLA Reference Definition

5.2.1.3 Generation of BPEL Processes and WSDL Descriptions

The SLA reference modifications are only considered during BPEL generation if the choreography has been saved prior to this generation process. During saving the role type definitions of the choreography are replaced with the role type definitions displayed in the QoS Integrator.

The BPEL generation process subsumes the following tasks:

- Generation of BPEL processes
- Transformation of SLAs
- Integration of policies
- Generation of Partner Link Type definitions

For each (interacting) role type of the choreography a BPEL process and Partner Link Type definition is generated. Subsequently the referenced SLAs are transformed into policies. Finally these policies are integrated into the respective BPEL processes. The generated BPEL processes are named according to the corresponding participant of the respective role type.
The WSDL generation process generates a WSDL description for each role type. These WSDL descriptions define the service interfaces of the BPEL processes and are named according to the corresponding participant of the role type definition.

5.3 Build to Order Use Case

The Build to Order (BTO) scenario describes a use case in the B2B area. The use case consists of a customer, a manufacturer and supplier for CPU's, main boards and hard discs. The manufacturer offers assembled IT hardware equipment to its customers. For these purposes 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 equipments to the manufacturer. The manufacturer 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 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 throughout the participants of the BTO scenario are illustrated in the collaboration sequence diagram shown in Figure 8.

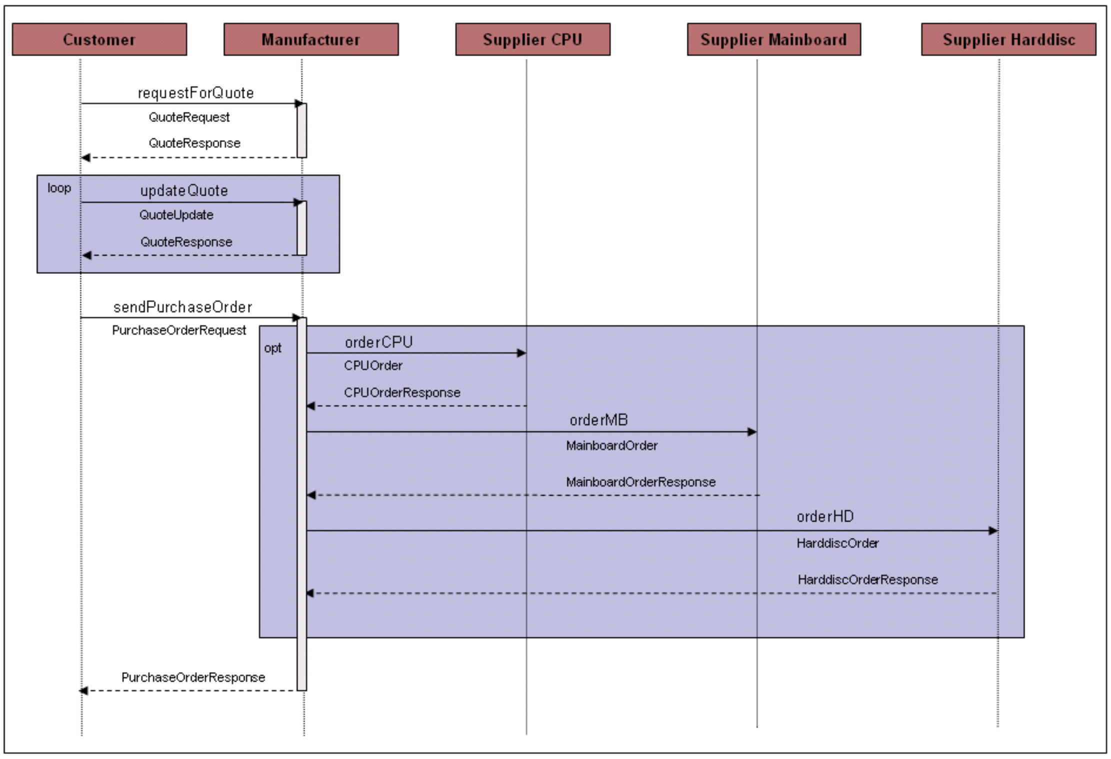


Figure 8 BTO Scenario - Collaboration Sequence Diagram

The BTO scenario comprises 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 message request `QuoteRequest`. The manufacturer receives the message request and replies to the service invocation be returning the message response `QuoteResponse`. Contrary to this, an asynchronous message scenario would require additional callback operations on the service requestor side. In this case the manufacturer would invoke an operation `requestForQuoteCallback` on the service interface of the customer to send back the `QuoteResponse`.

Table 4 summarizes the choreography participants of the BTO scenario. Each participant corresponds to one role type, which in turn defines the specified behavior interface. This behavior interface corresponds to a Web service interface defining the listed SOAP operations (invoked throughout the BTO collaboration).

Participant	Role Type	Behavior Interface	Operation
Customer	CustRoleType	custInterface	---
Manufacturer	ManRoleType	manInterface	requestForQuote updateQuote sendPurchaseOrder
SupplierCPU	SupCPURoleType	supCPUInterface	orderCPU
SupplierHD	SupHDRoleType	supHDInterface	orderHD
SupplierMB	SupMBRoleType	supMBInterface	orderMB

Table 4 BTO Scenario - Choreography Participants

The activity sequence of the BTO choreography is listed in the appendix of this work. This activity sequence shows the choreography flow which implements the BTO collaboration illustrated in Figure 8. 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. 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.

The BTO scenario distinguishes four different relationships between the choreography participants. The customer interacts with the manufacturer; the manufacturer interacts with the different suppliers. For each relationship an SLA about the invoked service operations is defined. These SLAs are summarized in Table 5.

SLA	Service Consumer	Service Provider	Service Operation	SLA Scenario
SLA1	Customer	Manufacturer	sendPurchaseOrder	2
SLA2	Manufacturer	SupplierCPU	orderCPU	1
SLA3	Manufacturer	SupplierHD	orderHD	1
SLA4	Manufacturer	SupplierMB	orderMB	1

Table 5 BTO Scenario - SLAs

The SLA between the customer and manufacturer defines an SLO for performance related SLA parameters (Execution Time, Throughput) and dependability related SLA parameters (Availability) which corresponds to the SLA scenario 2. The SLAs used between the manufacturer and the suppliers define an SLO for each SLA parameter which corresponds to the SLA scenario 1 (see Section 4.2).

An exemplary SLA (defining the SLA between the customer and manufacturer) is listed in the appendix of this work.

5.4 Evaluation

The BTO choreography and SLAs described above present the required input files which were used to evaluate the implementation.
During this evaluation SLA references were added to the role type definitions of the BTO choreography. Subsequently BPEL processes and WSDL descriptions were generated based on the modified BTO choreography. In the following an analysis of these modifications and generated files will be provided.

5.4.1 Choreography Modification

Listing 35 illustrates a small part of the BTO choreography for the manufacturer role type definition before and after the SLA reference for the customer has been added. The SLA reference conforms to the SLA integration approach which is proposed in Section 3.2. There is one insignificant difference regarding the element `description` which is bound to the namespace prefix `cdl`. The namespace prefix `cdl` points to the same URI as the default namespace which is declared at the root element `package`. This additional namespace prefix declaration (along with the namespace declaration `xmlns=""`) would not be necessary; it is added due to internal reasons which emanate from the handling process of default namespace declarations in dom4j.

```
<!-- Role Type Definition of Manufacturer -->
<package xmlns="http://www.w3.org/2005/10/cdl"
         xmlns:cdl="http://www.w3.org/2005/10/cdl"
         xmlns:qosp="http://example.org/qosp" ...>
   <!-- ... -->
   <roleType name="ManRoleType">
      <behavior interface="b2o:manInterface" name="ManBehaviour"/>
   </roleType>
   <!-- ... -->
</package>

<!-- Role Type Definition of Manufacturer with SLA Reference -->
<package xmlns="http://www.w3.org/2005/10/cdl"
         xmlns:cdl="http://www.w3.org/2005/10/cdl"
         xmlns:qosp="http://example.org/qosp" ...>
   <roleType name="ManRoleType">
      <behavior interface="b2o:manInterface" name="ManBehaviour">
         <cdl:description xmlns="" type="semantics">
            <qosp:slaReference name="SLA1"
           uri="D:/QoSIntegrator/input/Customer-Manufacturer-SLA.xml"
serviceconsumer="CustRoleType"/>
         </cdl:description>
      </behavior>
   </roleType>
</package>
```

Listing 35 Choreography Modification

5.4.2 BPEL & WSDL Generation

As mentioned above, the BPEL generation process does not only create BPEL stubs for each participant of the choreography but also includes the transformation of SLAs, the integration of policies and the generation of partner link type definitions. During the BPEL generation BPEL processes and partner link type definitions were created for all participants of the BTO choreography. Policies were integrated in the customer and manufacturer process. In the following the BPEL process of the customer will be used to analyze the implementation.

Appendix D provides the SLA between the customer and manufacturer. During the previous step an SLA reference was added to the manufacturer role type referencing this respective SLA and defining the customer as the service consumer. Thus, the QoS policy which corresponds to this SLA has to be integrated in the BPEL process of the customer. The SLA corresponds to the SLA scenario 2 and defines an SLO for the service performance and dependability. The SLO for the service performance defines two different scenarios (Execution Time less than 5 and Throughput greater equal 1 or Execution Time less than 7 and Throughput greater equal 3). The SLO for the service dependability states that Availability must not be less than 95 percent. This exemplary SLA definition was chosen to demonstrate that a complex SLA (defining nested SLO's with expressions) can be mapped to a QoS policy. The generated QoS policy was integrated in the BPEL process of the customer which is presented in Appendix E. It conforms to the SLA to policy mapping approach described in Section 4.2.2.2. The policy was added to the partner link definition which is linked to the service interface of the manufacturer. Furthermore the policy defines an attribute `operation` which corresponds to the operation defined in the SLA. This policy integration conforms to the integration approach described in Section 4.2.3.2. In the case of the manufacturer three policies (corresponding to the manufacturer – supplier SLAs) were generated and integrated in the partner link for the service interface of the respective supplier.

The generated BPEL flow reflects all BTO interactions which are related to the customer. In accordance with the relevance mapping approach defined in Section 3.1.2 only relevant choreography activities were mapped. `Sequence` elements which contain an `empty` element correspond to the silent actions which are defined for the customer in the BTO choreography.
Appendix F provides the partner link type definitions of the customer. For each partner link of the BPEL process a respective partner link type definition is provided.

In order to evaluate the WSDL generation process the WSDL description for the service interface of the manufacturer (Appendix G) is used exemplary. The generated WSDL description corresponds to the CDL to WSDL mapping approach defined in Section 3.2. All BTO choreography interactions which invoke operations on the manufacturer service interface are considered throughout the port type and binding definitions of the WSDL description. The `message` elements (defining the messages send to or from the customer) are related to the `informationType` elements of the BTO choreography. All `message` elements which are relevant for the customer were generated; the content (`message parts`) of these message elements was not specified. This would require additional processing of the information types. Information types are not specified directly in the choreography but represent references to existing XML Schema Elements or WSDL Types. During the WSDL generation process this references could be processed to define the respective `message part` elements.

6 Related work

The approach of integrating QoS in Web service based business process development scenarios has not been considered in current research so far. However certain aspects of this approach can be differentiated which are subject to related work. These aspects include top-down modeling of Web services (focusing on the mapping of WS-CDL), the extension of current Web services standards to include QoS attributes and the integration of policies in WS-BPEL.

In [12] Mendling et al. 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 of this work. Whereas the mapping of the before mentioned language constructs is referenced in detail, the second aspect for the derivation of BPEL processes is not addressed explicitly. This aspect deals with the concept of endpoint projection. In contrast to this work, no explicit projection rules are defined in order to determine which ordering structures are relevant for the respective 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 [10] Díaz 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 [32] is a toolset from pi4 Technologies and one of the first WS-CDL implementations. It is available as an Eclipse plug-in and provides a choreography designer, a choreography validation / simulation tool and a possibility to generate Java services from WS-CDL. Furthermore the support for generating BPEL processes and WSDL descriptions is currently in progress. In contrast to this thesis the integration of QoS throughout the development process has not been considered yet. It would be interesting to enrich pi4soa's choreography modeling and BPEL generation capabilities with the QoS integration approach discussed throughout this work.

The integration of QoS in Web services is subject of the approach presented in [2]. Based on standardized QoS parameters (defined in an XML schema) a framework is proposed which allows the dynamic selection of Web services with regard to QoS requirements. A client application sends a service request along with QoS requirements to a Web service broker (WSB). The WSB performs a UDDI registry lookup to receive a set of service providers and requests service descriptions from the respective service providers. The service providers return their service descriptions along with their QoS offers to the WSB which evaluates these offers against the client requirements. Finally the service provider with the most appropriate service is returned to the client. In contrast to this work the approach aims at integrating QoS directly in the Web service layer. This is accomplished by including QoS requirements and offers in service requests and service responses. However, in the context of Web service based business process development QoS has to be integrated in higher levels such as the choreography and orchestration layer.
In [17] Garcia et al. 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 architecture defines three main components, namely Brokers, Monitors and extended UDDI registries. Service providers specify QoS information on the offered services using QoS policies. Brokers process these QoS policies and publish them in extended UDDI registries. A consumer application requests service selection providing functional and QoS requirements. The Broker selects an appropriate service (fulfilling functional and QoS requirements) in the UDDI registry and reports the selected service back to the consumer. The Monitor intercept messages exchanged between the consumer application and the Web service to monitor the service execution and passes updated QoS information to the Broker. In turn the Broker updates the respective QoS information about the service in UDDI.

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. The focus is laid on the general QoS management architecture without providing details on a QoS policy specification. Furthermore no references can be found by what means QoS policies are stored on the side of the service provider. Both aspects are referred to in this work by specifying QoS policy assertions and by defining a policy integration approach.

In [1] Tai et al. investigate the composition of coordinated Web services. The WS-Policy framework is used to integrate coordination policies in BPEL. This is achieved by specifying policy assertions for the WS-C (Web services Coordination) framework. The integration of policies is performed by attaching policies to BPEL partner links or scopes. In contrast to this work the focus is laid on coordination context by specifying coordination requirements which have to be met by the involved Web services; QoS related aspects are not further considered.

7 Conclusion

The main contribution of this thesis was to show by what means QoS can be integrated in Web service based business process development scenarios. In the following a summarization of the results will be provided.

The engineering of Web service based business processes represents a top-down modeling approach in which private executable business processes are derived from a global choreography model. This top-down modeling approach transforms a WS-CDL choreography description into BPEL processes along with corresponding WSDL descriptions. Whereas the WSDL transformation process (see Section 3.2) represents a straightforward mapping approach between the languages constructs of WS-CDL and WSDL, the BPEL transformation process (see Section 3.1) is more complex. This is due to the fact that an endpoint projection has to be performed for each participant which determines the relevant constructs of a choreography description. In order to implement such an endpoint projection a relevance mapping approach was proposed. This relevance mapping guarantees that only those WS-CDL constructs will be considered for mappings which are directly relevant for the respective participant.

The integration of QoS throughout Web service based business process development comprises a twofold integration process. In accordance to the top-down modeling approach of Web services QoS will be initially integrated at the choreography layer. Subsequently a mapping of these QoS attributes will be performed followed by the integration at the orchestration layer. Neither WS-CDL nor WS-BPEL support the declaration of QoS attributes, yet both specifications can be extended to include such information accordingly.
In the case of the choreography layer such an extension was performed by the integration of SLA references inside a choreography description (see Section 4.1). Prior to this integration process an evaluation of SLA language specifications was provided identifying WSLA as an appropriate specification for the definition of SLAs. The integration of SLA references was performed by the use of semantic annotations inside the behavior definition of a participant’s role type (corresponding to the service interface of a service provider).

In the case of the orchestration layer QoS integration was performed by integrating Web service policies inside BPEL processes (see Section 4.2). These policies correspond to the QoS obligations defined in the SLAs referenced above. Due to the fact that the current WS-Policy framework provides specifications which focus on security and reliable messaging related aspects only, a policy specification for the QoS domain (WS-QoSPolicy) was provided. The WS-QoSPolicy defines policy assertions for attributes specified by the QoS model enabling an automated SLA to policy mapping approach. The actual policy integration process was looked at from two different perspectives. On the one hand the attachment of policies to WSDL descriptions was considered by using mechanisms defined in the WS-PolicyAttachment specification. On the other hand the integration of policies in BPEL processes was examined by evaluating an appropriate policy subject. It was shown that policies have to be integrated inside partner links of the service consumer's BPEL processes. Finally an evaluation was provided summarizing the advantages of the BPEL integration approach with regards to policy relevance and differentiation.

Following the theoretical part of this thesis an implementation was designed as a proof of concept. Based on a Build-To-Order use case scenario (see Section 5.3) including a WS-CDL choreography description and SLAs (conforming to the WSLA language specification) an evaluation of this implementation was performed (see Section 5.4). During this evaluation SLA references were added to the choreography description. Subsequently the referenced SLAs were mapped to corresponding policies. Finally BPEL processes (including the integrated policies) along with their corresponding WSDL descriptions were generated. Summarizing it can be stated that the evaluation confirmed the feasibility of the QoS integration process.

References

[1] S. Tai, R. Khalaf and T. Mikalsen. Composition of Coordinated Web Services. In *Proceedings of the 5th International Middleware Conference (Middleware'04), Toronto, Canada*, Oct. 2004.

[2] M. Tian, A. Gramm, T. Naumowicz, H. Ritter and J. Schiller. A Concept for QoS Integration in Web Services. In *Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03), Roma, Italy*, Dec. 2003.

[3] G. Yeom, T. Yun and D. Min. A QoS Model and Testing Mechanism for Quality-driven Web Services Selection. In *Proceedings of the 4th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2006) and the 2nd International Workshop on Collaborative Computing, Integration and Architecture (WCCIA'06), Gyeongju, Korea*, Apr. 2006.

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

[5] H. Ludwig, A. Dan, R. Kearney. Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. In *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04), New York, USA*, Nov. 2004.

[6] N. Oldham, K. Verma, A. Sheth and F. Hakimpour. Semantic WS-agreement partner selection. *In Proceedings of the 15th International Conference on World Wide Web (WWW'06), Edinburgh, Scotland*, May 2006.

[7] M. Aiello, G. Frankova and D. Malfatti: What's in an Agreement? An Analysis and an Extension of WS-Agreement. In *Proceedings of 3rd International Conference on Service-Oriented Computing (ICSOC'05), Amsterdam, The Netherlands*, Dec. 2005.

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

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

[10] G. Díaz, 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 Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW'06), Guadeloupe, French Caribbean*, Feb. 2006.

[11] J. Mendling and M. Hafner. From Inter-organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *Proceedings of On the Move – OTM – to Meaningful Internet Systems and Ubiquitous Computing (OTM'05), Agia Napa, Cyprus*, Oct. 2005.

[12] J. Mendling and M. Hafner. From WS-CDL Choreography to BPEL Process Orchestration. *Journal of Enterprise Management*, Vol.19, 2006.

[13] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, Vol.11(1), Mar. 2003.

[14] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer and A. Youssef. Web services on demand: WSLA-driven automated management. *IBM Systems Journal*, Vol.43(1), Jan. 2004.

[15] A. Dan, H. Ludwig and G. Pacifici. Web Service Differentiation with Service Level Agreements. *IBM White Paper*, May 2003.

[16] D. Garcia and M. Toledo: A Policy-based Web Service Infrastructure for Autonomic Service Integration. In *Proceedings of the 1st Latin American Autonomic Computing Symposium (LAACS'06), Campo Grande, Brazil*, July 2006.

[17] D. Garcia and M. Toledo: A Web Service Architecture Providing QoS Management. In *Proceedings of the 4th Latin American Web Congress (LA WEB'06), Puebla Cholula, Mexico*, Oct. 2006.

[18] A. Vedamuthu and D. Roth. Understanding Web Services Policy. *Microsoft Web Services Technical Article*, July 2006.

[19] W3C. *Web Services Choreography Description Language (WS-CDL)*, Nov. 2005. URL: http://www.w3.org/TR/ws-cdl-10/ (Last accessed: May 24, 2007).

[20] OASIS. Web Service Business Process Execution Language 2.0, 2006. URL: http://www.oasis-open.org/specs/index.php#wsbpelv2.0 (Last accessed: May 24, 2007).

[21] IBM. *Web Service Level Agreement (WSLA) Language Specification*, Jan. 2003. URL: http://www.research.ibm.com/wsla/ (Last accessed: May 24, 2007).

[22] Grid Resource Allocation Agreement Protocol (GRAAP). *Web Services Agreement Specification (WS-Agreement)*, Nov. 2005. (Last accessed: May 24, 2007).

[23] W3C. *Web Services Policy Framework (WS-Policy)*, Mar. 2007. URL: http://www.w3.org/TR/ws-policy/ (Last accessed: May 24, 2007).

[24] W3C. *Web Services Policy Attachment (WS-PolicyAttachment)*, Mar. 2007. URL: http://www.w3.org/TR/ws-policy-attach/ (Last accessed: May 24, 2007).

[25] OASIS. *Web Services Security Policy Language (WS-SecurityPolicy)*, Mar. 2005. URL: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-cd-02.html (Last accessed: May 24, 2007).

[28] OASIS. *Web Services ReliableMessaging Policy Assertion (WS-RMPolicy)*, Aug. 2006. URL: http://docs.oasis-open.org/ws-rx/wsrmp/200608/wsrmp-1.1-spec-cd-04.html (Last accessed: May 24, 2007).

[29] W3C. *XML Information Set*, 2004. URL: http://www.w3.org/TR/xml-infoset/ (Last accessed: May 24, 2007).

[30] W3C. *XSL Transformations (XSLT) – Version 2.0*, Jan. 2007. URL: http://www.w3.org/TR/xslt20/ (Last accessed: May 24, 2007).

[31] W3C. *XML Path Language (XPath) – Version 2.0*, Jan. 2007. URL: http://www.w3.org/TR/xpath20/ (Last accessed: May 24, 2007).

[32] pi4 Technologies Foundation. *pi4soa – Version 1.6.2,* 2007. URL: http://sourceforge.net/projects/pi4soa (Last accessed: May 24, 2007).

[33] MetaStuff Ltd. *dom4j – Version 1.6*, 2005. URL: http://www.dom4j.org (Last accessed: May 24, 2007).

[34] *Swing and JDOM XML Editor*. URL: http://hurring.com/code/java/xmleditor (Last accessed: May 24, 2007).

Appendix

Appendix A: WS-QoSPolicy XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://stud3.tuwien.ac.at/~e9751151/ws/2006/12/qos/policy"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
targetNamespace="http://stud3.tuwien.ac.at/~e9751151/ws/2006/12/qos/policy"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
   <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/09/policy"
schemaLocation="http://schemas.xmlsoap.org/ws/2004/09/policy/ws-policy.xsd"/>
   <xs:element name="ProcessingTimeAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
               use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="WrappingTimeAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
               use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="ExecutionTimeAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
               use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="ThroughputAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
               use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="AvailabilityAssertion">
      <xs:complexType>
         <xs:attribute name="value" type="xs:float" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required"/>
         <xs:attribute name="predicate" type="tns:PredicateType"
               use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:simpleType name="PredicateType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Greater"/>
         <xs:enumeration value="Less"/>
         <xs:enumeration value="Equal"/>
         <xs:enumeration value="GreaterEqual"/>
         <xs:enumeration value="LessEqual"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:annotation>
      <xs:documentation>The following QoS attributes can be used in an SLA to define
                  the corresponding SLA parameters</xs:documentation>
   </xs:annotation>
   <xs:simpleType name="QoSAttributes">
      <xs:restriction base="xs:string">
         <xs:enumeration value="ExecutionTime"/>
         <xs:enumeration value="Throughput"/>
         <xs:enumeration value="Availability"/>
      </xs:restriction>
   </xs:simpleType>
```

```xml
   <xs:annotation>
      <xs:documentation>The following element defines an SLA reference for extending
                  WS-CDL's behavior element</xs:documentation>
   </xs:annotation>
   <xs:element name="slaReference">
      <xs:complexType>
         <xs:attribute name=" name " type="xs:string" use="required"/>
         <xs:attribute name="uri" type="xs:anyURI" use="required"/>
         <xs:attribute name=" serviceconsumer " type="xs:string"
               use="required"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Appendix B: BTO Choreography - Activity Sequence

```
Initialize : Client invokes choreography by sending a QuoteRequest to the Customer

Sequence : Main sequence of choreography. Contains ProcessQuote, ProcessOrder,
    EvaluateOrder, SendResponse

    Sequence : ProcessQuote

        RequestForQuote : Customer sends QuoteRequest to Manufacturer

        SilentAction : [Manufacturer]: Process QuoteRequest

        RequestForQuote : Manufacturer sends QuoteResponse to Customer

        SilentAction : [Customer] Process QuoteResponse (1)

        Inform1 : Synchronize variable QuoteAccept between Customer and Manufacturer.
            Customer informs Manufacturer if QuoteAccept is true or false.

        QuoteBartering : QuoteBartering loop: proceed with loop as long as Customer has
            not accepted quote

            UpdateQuote : Customer sends QuoteUpdate to Manufacturer

            SilentAction : [Manufacturer] Process QuoteUpdate

            UpdateQuote : Manufacturer sends QuoteResponse to Customer

            SilentAction : [Customer] Process QuoteResponse (2)

            Inform2 : Synchronize variable QuoteAccept between Customer and
                Manufacturer. Customer informs Manufacturer if QuoteAccept is true or
                false.

        SendPurchaseOrder : Customer sends PurchaseOrder to Manufacturer

    Sequence : ProcessOrder

        SilentAction : [Manufacturer]: Invoke CheckHardwareStock Webservice. Determine
            how much of the required CPU|MB|HD is not in stock.

        Parallel : ParallelProcesses

            Sequence : ProcessCPUSequence

                Choice : ChoiceCPU

                    Choice_CPUnotInStock : CPU not in stock

                        Assign : Assign missing CPU to HardwareOrderCPU

                        SendCPUOrder : Manufacturer sends HardwareOrderCPU to
                            SupplierCPU

                        SilentAction : [SupplierCPU]: Process CPUOrder

                        SendCPUOrder : SupplierCPU sends HardwareOrderResponse to
                            Manufacturer

                    Choice_CPUinStock : CPU in stock

                        NoAction : CPUOrder not necassary

                        Assign : Set HardwareOrderCPUResponse to true

            Sequence : ProcessMBSequence

                Choice : ChoiceMB

                    Choice_MBnotInStock : MB not in stock

                        Assign : Assign missing MB to HardwareOrderMB

                        SendMBOrder : Manufacturer sends HardwareOrderMB to SupplierMB
                        SilentAction : [SupplierMB]: Process MBOrder
                        SendMBOrder   :   SupplierMB   sends   HardwareOrderResponse   to
                            Manufacturer
                    Choice_MBinStock : MB in stock
                        NoAction : MBOrder not necassary
                        Assign : Set HardwareOrderMBResponse to true
            Sequence : ProcessHDSequence
                Choice : ChoiceHD
                    Choice_HDnotInStock : HD not in stock
                        Assign : Assign missing HD to HardwareOrderHD
                        SendHDOrder : Manufacturer sends HardwareOrderHD to SupplierHD
                        SilentAction : [SupplierHD]: Process HDOrder
                        SendHDOrder   :   SupplierHD   sends   HardwareOrderResponse   to
                            Manufacturer
                    Choice_HDinStock : HD in stock
                        NoAction : HDOrder not necassary
                        Assign : Set HardwareOrderHDResponse to true
    Sequence : EvaluateOrder
            Choice : Determine if PurchaseOrder was successfully performed
                    PO_success : PurchaseOrder successfully performed
                            Assign : Set confirmation of PurchaseOrder to true
                    PO_failure : PurchaseOrder not successfully performed
                            Assign : Set confirmation of PurchaseOrder to false
    SilentAction : [Manufacturer]: Assign PurchaseOrderResponse
    SendPurchaseOrder : Manufacturer sends PurchaseOrderResponse to Customer
Initialize : Sends result to client which has invoked the choreography
```

Appendix C: BTO Customer-Manufacturer SLA

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SLA xmlns="http://www.ibm.com/wsla" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:wsla="http://www.ibm.com/wsla" name="Customer-Manufacturer-SLA">
  <Parties>
    <ServiceProvider name="Manufacturer" />
    <ServiceConsumer name="Customer" />
    <SupportingParty name="MeasurementService">
      <Sponsor>Manufacturer</Sponsor>
      <Role>MeasurementService</Role>
    </SupportingParty>
  </Parties>
  <ServiceDefinition name="manService">
    <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType"
      name="sendPurchaseOrder">
      <SLAParameter name="p1" type="ExecutionTime" unit="seconds">
        <Metric>ExecutionTimeMetric</Metric>
      </SLAParameter>
      <SLAParameter name="p2" type="Throughput" unit="requests">
        <Metric>ThroughputMetric</Metric>
      </SLAParameter>
      <SLAParameter name="p3" type="Availability" unit="uptimeratio">
        <Metric>AvailabilityMetric</Metric>
      </SLAParameter>
      <!-- Hypothetical metrics. It is assumed that metrics representing the
           respective SLA parameters are provided and can be retrieved. -->
      <Metric name="m1" type="float">
        <Source>MeasurementService</Source>
        <MeasurementDirective xsi:type="Gauge">
          <MeasurementURI>http://example.org/ExecutionTimeMetric</MeasurementURI>
        </MeasurementDirective>
      </Metric>
      <Metric name="m2" type="float">
        <Source>MeasurementService</Source>
        <MeasurementDirective xsi:type="Gauge">
          <MeasurementURI>http://example.org/ThroughputMetric</MeasurementURI>
        </MeasurementDirective>
      </Metric>
      <Metric name="m3" type="float">
        <Source>MeasurementService</Source>
        <MeasurementDirective xsi:type="Gauge">
          <MeasurementURI>http://example.org/AvailabilityMetric</MeasurementURI>
        </MeasurementDirective>
      </Metric>
      <WSDLFile>manInterfacewsdl</WSDLFile>
      <SOAPBindingName>manBehaviourBinding</SOAPBindingName>
      <SOAPOperationName>sendPurchaseOrder</SOAPOperationName>
    </Operation>
  </ServiceDefinition>
  <Obligations>
         <ServiceLevelObjective name="SLOServicePerformance">
         <Obliged>SupplierCPU</Obliged>
         <Validity>
            <Start>2007-01-01T00:00:00.000+01:00</Start>
            <End>2007-12-31T00:00:00.000+01:00</End>
         </Validity>
         <Expression>
            <Or>
               <Expression>
                  <And>
                     <Expression>
                        <Predicate xsi:type="wsla:Less">
                           <SLAParameter>p1</SLAParameter>
                           <Value>5</Value>
                        </Predicate>
                     </Expression>
                     <Expression>
                        <Predicate xsi:type="wsla:GreaterEqual">
                           <SLAParameter>p2</SLAParameter>
                           <Value>1</Value>
                        </Predicate>
                     </Expression>
                  </And>
               </Expression>
               <Expression>
                  <And>
                     <Expression>
                        <Predicate xsi:type="wsla:Less">
                           <SLAParameter>p1</SLAParameter>
                           <Value>7</Value>
                        </Predicate>
                     </Expression>
                     <Expression>
                        <Predicate xsi:type="wsla:GreaterEqual">
                           <SLAParameter>p2</SLAParameter>
                           <Value>3</Value>
                        </Predicate>
                     </Expression>
                  </And>
               </Expression>
            </Or>
         </Expression>
         <EvaluationEvent>NewValue</EvaluationEvent>
      </ServiceLevelObjective>
      <ServiceLevelObjective name="SLOServiceDepandability">
         <Obliged>SupplierCPU</Obliged>
         <Validity>
            <Start>2007-01-01T00:00:00.000+01:00</Start>
            <End>2007-12-31T00:00:00.000+01:00</End>
         </Validity>
         <Expression>
            <Not>
               <Expression>
                  <Predicate xsi:type="wsla:Less">
                     <SLAParameter>p3</SLAParameter>
                     <Value>0.95</Value>
                  </Predicate>
               </Expression>
            </Not>
         </Expression>
         <EvaluationEvent>NewValue</EvaluationEvent>
      </ServiceLevelObjective>
    <!-- Actions which should be taken if a SLO is violated (ActionGuarantees) have
      not been defined. -->
  </Obligations>
</SLA>
```

Appendix D: BTO Customer BPEL Process

```
<?xml version="1.0" encoding="UTF-8"?>
<process xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://example.org/build2order"
targetNamespace="http://example.org/build2order" name="BuildToOrderCDL_BPEL-
Process_Customer">
  <partnerLinks>
    <partnerLink name="manInterface" partnerRole="manInterfaceRole"
partnerLinkType="tns:manInterfaceLT">
      <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:qosp="http://example.org/qosp" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="Customer-
Manufacturer-SLA" qosp:operation="sendPurchaseOrder">
        <wsp:All>
          <wsp:ExactlyOne>
            <wsp:All>
              <wsp:All>
                <qosp:ExecutionTime unit="seconds" predicate="Less" value="5"/>
                <qosp:Throughput unit="requests" predicate="GreaterEqual" value="1"/>
              </wsp:All>
            </wsp:All>
            <wsp:All>
              <wsp:All>
                <qosp:ExecutionTime unit="seconds" predicate="Less" value="7"/>
                <qosp:Throughput unit="requests" predicate="GreaterEqual" value="3"/>
              </wsp:All>
            </wsp:All>
          </wsp:ExactlyOne>
          <wsp:All>
            <qosp:Availability unit="uptimeratio" predicate="GreaterEqual"
             value="0.95"/>
          </wsp:All>
        </wsp:All>
      </wsp:Policy>
    </partnerLink>
    <partnerLink name="custInterface" myRole="custInterfaceRole"
     partnerLinkType="tns:custInterfaceLT"/>
  </partnerLinks>
  <variables>
    <variable name="PurchaseOrder" messageType="tns:PurchaseOrder"/>
    <variable name="PurchaseOrderResponse" messageType="tns:PurchaseOrderResponse"/>
    <variable name="QuoteAccept" messageType="tns:QuoteAccept"/>
    <variable name="QuoteRequest" messageType="tns:QuoteRequest"/>
    <variable name="QuoteResponse" messageType="tns:QuoteResponse"/>
    <variable name="QuoteUpdate" messageType="tns:QuoteUpdate"/>
  </variables>
  <sequence>
    <receive operation="initialize" Variable="QuoteRequest"
     partnerLink="custInterface" portType="tns:custInterface"/>
    <sequence>
      <sequence>
        <invoke operation="requestForQuote" inputVariable="QuoteRequest"
         partnerLink="manInterface" portType="tns:manInterface"
         outputVariable="QuoteResponse"/>
        <sequence name="Customer_AssignQuoteAccept">
          <empty/>
        </sequence>
        <invoke operation="inform" inputVariable="QuoteAccept"
         partnerLink="manInterface" portType="tns:manInterface"/>
        <while name="QuoteBartering"
         condition="bpws:getVariable('QuoteAccept','accept','')=false()">
          <sequence>
            <invoke operation="updateQuote" inputVariable="QuoteUpdate"
             partnerLink="manInterface" portType="tns:manInterface"
             outputVariable="QuoteResponse"/>
            <sequence name="Customer_ProcessUpdate">
              <empty/>
            </sequence>
            <invoke operation="inform" inputVariable="QuoteAccept"
             partnerLink="manInterface" portType="tns:manInterface"/>
          </sequence>
        </while>
      </sequence>
      <invoke operation="sendPurchaseOrder" inputVariable="PurchaseOrder"
       partnerLink="manInterface" portType="tns:manInterface"
       outputVariable="PurchaseOrderResponse"/>
    </sequence>
    <reply operation="initialize" Variable="PurchaseOrderResponse"
     partnerLink="custInterface" portType="tns:custInterface"/>
  </sequence>
</process>
```

Appendix E: BTO Customer Partner Link Types

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:tns="http://example.org/build2order"
name="BuildToOrderCDL_PartnerLinkTypes_Customer">
 <plnk:partnerLinkType xmlns="" name="manInterfaceLT">
  <plnk:role name="manInterfaceRole">
   <plnk:portType name="tns:manInterface"/>
  </plnk:role>
 </plnk:partnerLinkType>
 <plnk:partnerLinkType xmlns="" name="custInterfaceLT">
  <plnk:role name="custInterfaceRole">
   <plnk:portType name="tns:custInterface"/>
  </plnk:role>
 </plnk:partnerLinkType>
</definitions>
```

Appendix F: BTO Manufacturer WSDL Description

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://example.org/build2order"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
targetNamespace="http://example.org/build2order" name="BuildToOrderCDL_WSDL-
Description_Manufacturer">
 <message name="QuoteRequest"/>
 <message name="QuoteResponse"/>
 <message name="QuoteAccept"/>
 <message name="QuoteUpdate"/>
 <message name="PurchaseOrder"/>
 <message name="PurchaseOrderResponse"/>
 <portType name="manInterface">
  <operation name="requestForQuote">
   <input name="request" message="tns:QuoteRequest"/>
   <output name="respond" message="tns:QuoteResponse"/>
  </operation>
  <operation name="inform">
   <input name="req2resp" message="tns:QuoteAccept"/>
   <output name="resp2req" message="tns:QuoteAccept"/>
  </operation>
  <operation name="updateQuote">
   <input name="request" message="tns:QuoteUpdate"/>
   <output name="respond" message="tns:QuoteResponse"/>
  </operation>
  <operation name="sendPurchaseOrder">
   <input name="request" message="tns:PurchaseOrder"/>
   <output name="respond" message="tns:PurchaseOrderResponse"/>
  </operation>
 </portType>
 <binding name="ManBehaviourBinding" type="tns:manInterface">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="requestForQuote">
   <soap:operation soapAction="http://example.org/build2order/requestForQuote"/>
   <input>
```

```
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://example.org/build2order"/>
   </input>
   <output>
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://example.org/build2order"/>
   </output>
  </operation>
  <operation name="inform">
   <soap:operation soapAction="http://example.org/build2order/inform"/>
   <input>
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://example.org/build2order"/>
   </input>
   <output>
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://example.org/build2order"/>
   </output>
  </operation>
  <operation name="updateQuote">
   <soap:operation soapAction="http://example.org/build2order/updateQuote"/>
   <input>
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://example.org/build2order"/>
   </input>
   <output>
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://example.org/build2order"/>
   </output>
  </operation>
  <operation name="sendPurchaseOrder">
   <soap:operation soapAction="http://example.org/build2order/sendPurchaseOrder"/>
   <input>
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://example.org/build2order"/>
   </input>
   <output>
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://example.org/build2order"/>
   </output>
  </operation>
 </binding>
 <service name="manInterfaceService">
  <port name="manInterfacePort" binding="tns:ManBehaviourBinding">
   <soap:address location="URI_to_be_specified"/>
  </port>
 </service>
</definitions>
```