BAMBERGER BEITRÄGE

ZUR WIRTSCHAFTSINFORMATIK UND ANGEWANDTEN INFORMATIK

ISSN 0937-3349

Nr. 80

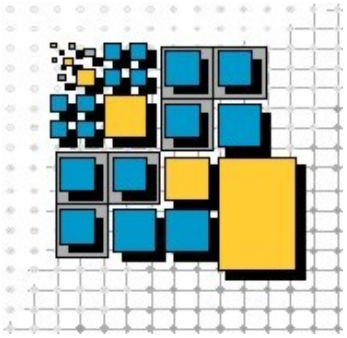
QoS-Enabled B2B Integration

Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger and Guido Wirtz

May 2009

FAKULTÄT WIRTSCHAFTSINFORMATIK UND ANGEWANDTE INFORMATIK

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG



Distributed and Mobile Systems Group

Otto-Friedrich Universität Bamberg

Feldkirchenstr. 21, 96052 Bamberg, GERMANY

Prof. Dr. rer. nat. Guido Wirtz

http://www.uni-bamberg.de/en/fakultaeten/wiai/faecher/informatik/lspi/

Due to hardware developments, strong application needs and the overwhelming influence of the net in almost all areas, distributed and mobile systems, especially software systems, have become one of the most important topics for nowadays software industry. Unfortunately, distribution adds its share to the problems of developing complex software systems. Heterogeneity in both, hardware and software, concurrency, distribution of components and the need for interoperability between different systems complicate matters. Moreover, new technical aspects like resource management, load balancing and deadlock handling put an additional burden onto the developer. Although subject to permanent changes, distributed systems have high requirements w.r.t. dependability, robustness and performance.

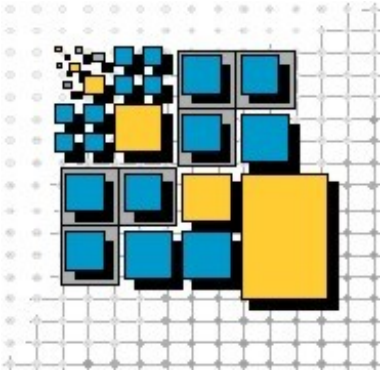
The long-term common goal of our research efforts is the development, implementation and evaluation of methods helpful for the development of robust and easy-to-use software for complex systems in general while putting a focus on the problems and issues regarding the software development for distributed as well as mobile systems on all levels. Our current research activities are focused on different aspects centered around that theme:

- *Robust and adaptive Service-oriented Architectures*: Development of design methods, languages and middleware to ease the development of SOAs with an emphasis on provable correct systems that allow for early design-evaluation due to rigorous development methods and tools. Additionally, we work on approaches to autonomic components and container-support for such components in order to ensure robustness also at runtime.
- *Agent and Multi-Agent (MAS) Technology:* Development of new approaches to use Multi-Agent-Systems and negotiation techniques, for designing, organizing and optimizing complex distributed systems, esp. service-based architectures.
- *Peer-to-Peer Systems*: Development of algorithms, techniques and middleware suitable for building applications based on unstructured as well as structured P2P systems. A specific focus is put on privacy as well as anonymity issues.
- *Context-Models and Context-Support for small mobile devices*: Investigation of techniques for providing, representing and exchanging context information in networks of small mobile devices like, e.g. PDAs or smart phones. The focus is on the development of a truly distributed context model taking care of information reliability as well as privacy issues.
- *Visual Programming- and Design-Languages*: The goal of this long-term effort is the utilization of visual metaphors and languages as well as visualization techniques to make design- and programming languages more understandable and, hence, easy-to-use.

More information about our work, i.e., projects, papers and software, is available at our homepage. If you have any questions or suggestions regarding this report or our work in general, don't hesitate to contact me at guido.wirtz@wiai.uni-bamberg.de

Guido Wirtz

Bamberg, April 2006



QoS-Enabled B2B Integration

Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger and Guido Wirtz

Lehrstuhl für Praktische Informatik, Fakultät WIAI

Abstract Business-To-Business Integration (B2Bi) is a key mechanism for enterprises to gain competitive advantage. However, developing B2Bi applications is far from trivial. Inter alia, agreement among integration partners about the business documents and the control flow of business document exchanges as well as applying suitable communication technologies for overcoming heterogeneous IT landscapes are major challenges. At the same time, choreography languages such as ebXML BPSS (ebBP), orchestration languages such as WS-BPEL and Web Services are promising to provide the foundations for seamless interactions among business partners.

Automatically translating choreography agreements of integration partners into partner-specific orchestrations is an obvious idea for ensuring conformance of orchestration models to choreography models. Moreover, the application of such model-driven development methods facilitates productivity and cost-effectiveness whereas applying a service oriented architecture (SOA) based on WS-BPEL and Web Services leverages standardization and decoupling. By now, the realization of QoS attributes has not yet received the necessary attention that makes such approaches suitable for B2Bi. In this report, we describe a proof-of-concept implementation of the translation of ebBP choreographies into WS-BPEL orchestrations that respects B2Bi-relevant QoS attributes.

Keywords B2Bi, QoS, Choreography to Orchestration Generation, Web Services, ebXML, ebBP, BPEL, SOA, WS-* standards

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

API	Application Programming Interface
BC	Business Collaboration
BT	Business Transaction
BTA	Business Transaction Activity
B2B	Business-to-Business
B2Bi	Business-to-Business integration
BPEL	Web Service Business Process Execution Language
BPMN	Business Process Modeling Notation
BPSS	Business Process Specification Schema
BSI	Business Service Interface
CA	Collaboration Activity
CP	centralized perspective
CPP	Collaboration Protocol Profile
CPA	Collaboration Protocol Agreement
DP	distributed perspective
DTD	Document Type Definition
ebBP	ebXML Business Process Technical Specification
ebXML	Electronic Business using eXtensible Markup Language
EDI	Electronic Data Interchange
EJB	Enterprise Java Beans
e-business	electronic business
e-commerce	electronic commerce
FI	Fast Infoset
IT	Information Technology
JAXB	Java Architecture for XML Binding
JAXP	Java Architecture for XML Processing
JPA	Java Persistence API
MTOM	Message Transmission Optimization Mechanism
NES	Northern European Subset
OASIS	Organization for the Advancement of Structured Information Standards
PIP	Partner Interface Process
POJO	Plain old Java objects
QoS	Quality of Service
SMEs	small and medium sized enterprises
SOA	service-oriented architecture
SOAP	SOAP (formerly Simple Object Access Protocol)
SSL	Secure Sockets Layer
TS	Technical Specification
UBL	Universal Business Language
UML	Unified Modeling Language
UN/CEFACT	United Nations Centre for Trade Facilitation and Electronic Business
URL	Uniform Resource Locator
UUID	Universal Unique Identifier
W3C	World Wide Web Consortium

WS	Web service
WSDL	Web Service Description Language
XML	eXtended Markup Language
XPath	XML Path Language
XSD	XML Schema Document
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation

Chapter 1

Introduction

1.1 Business-2-Business-Integration

Enterprises today are enforced by market pressure to integrate business processes with their partners along the supply chain. The term Business-to-Business integration (B2Bi) is frequently used to denote business process integration crossing enterprise boundaries or, more generally, boundaries of organizational units. The development of B2Bi software is far from trivial for various reasons. Personnel from different organizational units with different vocabulary and background are frequently involved in building B2Bis which requires extensive communication for defining message formats, message contents and choreography. Central technical infrastructure is frequently not available or prohibited by business politics so that truly distributed computing is needed. Thus issues like heterogeneity, communication over possibly insecure media or partial failure of autonomous systems have to be addressed. Finally, B2Bi software has to meet enterprise level issues like dynamic business relations, investment in existing IT systems as well as QoS requirements for business transactions like reliability, security and time constraints.

Service Oriented Computing (SOC) promises to cover most[1] of these integration requirements in a way that reuses existing IT assets, ensures loose coupling of interacting IT systems and provides for reusability of generated artifacts. The paradigm of Service Oriented Architecture (SOA) provides the conceptual foundations for SOC by decomposing complex business processes in manageable business tasks and subsequently assigning services that accomplish these tasks. For building B2Bi systems, enterprises not only consume their own services for accessing information and functionality but their partners' processes as well, e.g., the electronic exchange of orders or notifications about the processing status of orders.

WS-BPEL (BPEL in the following) [JEA+07] and Web services are one realization option for employing SOC in the B2Bi context where Web Services help in overcoming the technical obstacles in communicating across heterogeneous IT systems and BPEL offers a standardized way for defining the sequence of message exchanges of integration partners. Together, these

[1] QoS still is an issue

technologies promise to allow for fast and cost-effective implementation while still ensuring high quality in terms of decoupling, standardization and interoperability. Yet, integration partners also need functionality for agreeing upon the overall message flow of a B2Bi, in particular a definition of the business document and control message types to be used and the sequence of message exchanges considering each integration partner is needed. Choreography languages like ebXML BPSS (ebBP) [YWM+06] provide such functionality and thus are a valuable component in the tool set for realizing B2Bi projects.
An introduction to the technologies used in the work at hand is given in section 2.

1.2 The Necessity of QoS Features

As mentioned above, Business-2-Business-Integration stands for doing business across the boundaries of organizations. In case messages are exchanged via the Internet, reliability and security issues have to be addressed as business documents typically have legal consequences and may not be disclosed to third parties. Hence, business partners usually are heavily interested in transmitting messages in a reliable and secure manner. Reliability concerns the problem of handling communication errors and ensuring consistency between the integration partners' IT systems regarding the status of transmitted messages in such cases. Security typically is defined in terms of several more detailed QoS requirements like confidentiality, which means that no third party gains knowledge about the content of the message, integrity, which means integration partners are able to detect manipulation of messages by third parties, non-repudiation, which means integration partners cannot deny having sent a particular message, authentication, which means a third party is not able to misuse the identity of a business partner, and authorization which means that actions performed by an entity are indeed legitimate. A more comprehensive definition of QoS is given in [DLS05]: *"[...]the term QoS [...is] used to denote all non-functional aspects of a service which may be used by clients to judge service quality. This extends other more restrictive QoS definitions such as the common interpretation of QoS to mean network performance attributes."*

This definition is valid and comprehensive, but a drawback of this definition is its universality which makes it hard to manage and support. Therefore, the work at hand strives for a set of QoS requirements that at least have to be supported for enabling B2Bi which may be found by adopting the judgment of B2Bi experts. ebBP defines such a set of QoS requirements: *"The ebBP technical specification provides parameters that can be used to specify certain levels of security and reliability. This specification provides these parameters in general business terms"* ([YWM+06], sec. 3.5.7). Thus, in the context of the work at hand, QoS is defined in terms of the QoS attributes defined in ebBP.

1.3 Problem Identification and Definition

B2Bi is not a new concept in today's business world. Communication between business partners across the boundaries of the involved firms is already working today. For example a lot of

enterprises use the UN/EDIFACT standard for communication. One problem with the existing attempts to establish business-to-business integration is that until today only enterprises of considerable size are able to install a proper system for inter-business communication. So there still exists a lot of automation potential, especially for small and medium-sized enterprises.

Why is it that small and medium-sized businesses do not try to exploit these potentials? One aspect might be that small firms often do not have an IT department, and, especially in non-IT-businesses, there is often a lack of IT knowledge. Another aspect is, that considerable budgets for the implementation of a working B2Bi system are needed. In addition, communication standards in some industries are proprietary and the licenses for using them or the licenses for the implementing software are expensive.

If standards were used, one big problem would already be solved. But in reality, integration often is not based on widespread standards but on "home-grown" message formats and spontaneously defined protocols. This procedure may work if only a few partners interact with each other but the coordination costs increase exponentially when the network of partners is expanding. The usage of open and widespread standards may help to handle this problem. Standards exist for all (technical) layers of business-to-business communication: From simple data transmission via, e.g., the Transmission Control Protocol (TCP) to the definition of business documents as defined by, e.g., the Unified Business Language (UBL).

The goal of this work is to develop a proof-of-concept implementation that shows that realizing B2Bi based on open SOC standards is possible. While applying such standards facilitates flexibility, interoperability and decoupling sufficient support for B2Bi-relevant QoS is not automatically provided for. Hence, special attention is payed to the realization of these QoS features. Open source technology and freely available tools are envisaged as implementation platform in order to leverage community engagement and ensure accessibility not only for multi-national enterprises. But are the available standards already as mature as needed for productive usage? And how easy is it to implement a working system for business-to-business integration?

In conclusion and combined to one question the postulated research question is:

Is it possible to support B2Bi throughout all relevant abstraction layers with an integrated standard stack that meets all important B2Bi QoS requirements?

Chapter 2

Fundamentals

2.1 The Universal Business Language (UBL) and the Northern European Subset (NES)

The Universal Business Language (UBL) provides a unified library of standard business documents, such as purchase orders, invoices or catalogues. The standardized schemas of these business documents enable simple electronic data exchange between partners. The design of the UBL arranges for compatibility with existing business, legal, auditing and records management practices. As free and open OASIS (Organization for the Advancement of Structured Information Standards) standard, it is suitable as entry-point for electronic business, in particular for small and medium-sized companies.

The UBL defines 31 business documents and a common library for basic information (e.g. units, currencies, date formats) for a huge variety of possible unified business processes in the phases presale, ordering, delivery, invoicing and payment. Besides these data format definitions, several tools, validators and generators are already available[1].

The Northern European Subset (NES) is a subset of the UBL, developed from representatives of the Northern European countries Denmark, Sweden, Norway, Finland, the United Kingdom and Iceland. The NES defines eight "profiles" which apply to defined business scenarios. The goal of the NES is to enable companies and institutions to implement e-commerce (electronic commerce) by agreeing to a specific profile and thus ease implementation on both sides.

Every profile is modeled as a UML Activity diagram with certain decisions (e.g. reject or accept order in profile 1) and exchanged objects which are business documents (e.g. Order and Application Response in profile 1). The NES uses adapted UBL XML schema files for its business documents, so that a NES Invoice also conforms to the UBL common library. In turn, a UBL Invoice may conform to the NES common library but not necessarily has to (see figure 2.1 for an illustration of the relation between UBL and NES).

[1] `http://ubl.xml.org/products,lastvisitedMarch2008`

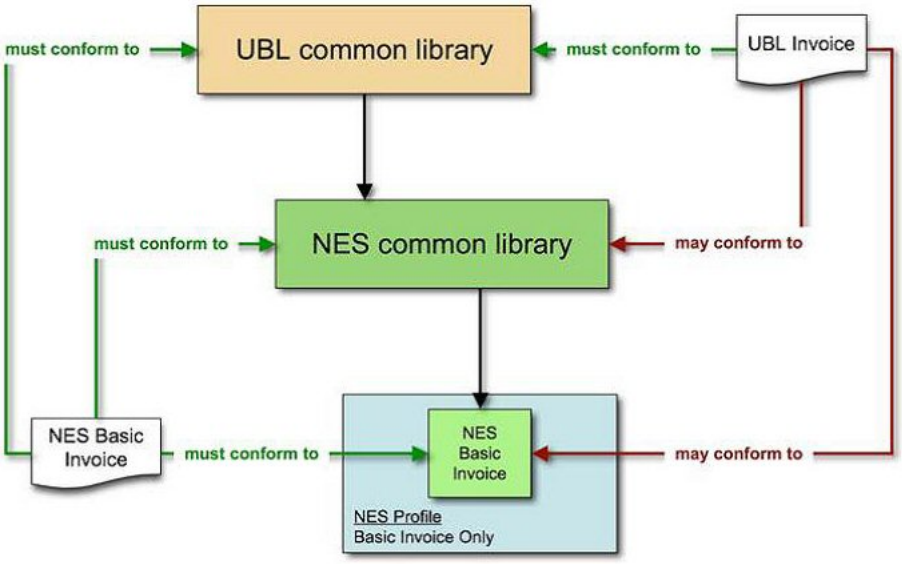


Figure 2.1: The relation of NES and UBL, taken from [Gro07a, page 5]

The profiles defined by the NES are:

- Profile 1: Catalogue only
- Profile 2: Catalogue with updates
- Profile 3: Basic Order only
- Profile 4: Basic Invoice only
- Profile 5: Basic Billing
- Profile 6: Basic Procurement
- Profile 7: Simple Procurement
- Profile 8: Basic Billing with dispute Response

A detailed introduction to the profiles is given in chapter 3.

2.2 The ebXML Business Process Specification Schema Technical Specification (ebBP)

The ebXML Business Process Specification Schema Technical Specification (ebBP) is part of the ebXML (Electronic Business using XML) standards suite, developed by OASIS and the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT). In this context, XML is used because of its ability to define structured human and machine readable documents following a defined schema such as XSD or DTD. Documents and signals as well as processes are described in XML.

For the purpose of doing e-business, ebXML defines the following parts in its technical architecture as shown in figure 2.2. For more information of the specification, please use the given references.

- Business Process Schema Specification: This specification can be used as a meta model to describe business processes and rules for doing business based on the concept of Business Transactions. ebBP is described in more detail below in this section.[YWM+06]
- Core Components: The Core Components are reusable components of business processes, which are non-specific to the involved parties and common for all businesses.[WPA+01]
- Registry Services Specification: The registry services represent a standard for a central point to share documents with business semantics like core components, messages, Collaboration-Protocol Agreements etc. and providing key functions like search and discovery. [BCC+02]
- Message Service Specification: The Message Service represents a standard for the exchange of e-business information among different organizations considering security and reliable messaging without dictating a specific transport protocol. [BBB+02]
- Collaboration-Protocol Profile and Agreement Specification (CPP/CPA): This specification is used in order to describe technical capabilities of the trading partners and to describe the agreement between two partners for data interchange; [ACC+02]

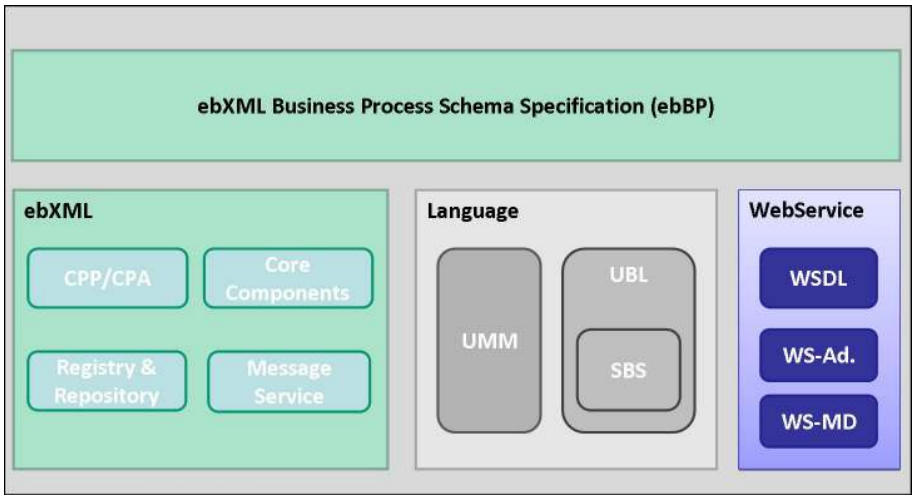


Figure 2.2: Usage context of ebBP

Figure 2.2 provides an overview of the usage context of the Business Process Schema Specification. The Business Process Schema Specification (ebBP) describes a choreography of doing e-business between trading partners. Other ebXML technical specifications like Collaboration Protocol Profile and Collaboration Protocol Agreement (CPP/CPA) [ACC+02] as well as ebXML messaging [BBB+02] complement ebBP for specifying the actual message exchange. Alternatively, if business partners do not support these ebXML standards, Web services (see section 2.3) can replace this functionality. As shown in figure 2.2, further non-ebXML standards can be used together with ebBP. For example, the business documents defined by the Unified

Business Language (UBL) can be used to describe e-business information. (see [YWM+06, page 15 and 16])

The core concept of the Business Process Specification Schema is the so-called Business Transaction (BT). Business Transactions represent reusable, atomic units of work. They specify a document flow between two independent parties. These parties are specified as requesting respectively responding roles within the context of a Business Transaction and so-called Requesting and Responding Business Activities are used for specifying the exchange of business documents in more detail. Exactly one Requesting and one Responding Business Activity always have to be defined within a Business Transaction. Depending on whether a Requesting or a Responding Business Activity is used the number of alternative, possible definitions of Document Envelopes within an Business Activity can vary from one (requesting) to an infinite (responding) number. One way document transmissions can be specified by leaving out a DocumentEnvelope within the Responding Business Activity. Thus, one way as well as bidirectional transactions can be specified.

In the following, the most important types of Business Transactions are described (see [YWM+06], section 3.4.9.1, for a complete list):

- Notification: notifications with a business context, e.g., the occurrence of an error at one party;
- Information Distribution: exchange of informations between two parties with no legal intent;
- Commercial Transaction: transactions with a business context, e.g., transmission of an invoice;

The documents to be transmitted within Responding/Requesting Activities are specified separately and are referenced by their `nameID`. The structure of Business Documents can be defined by the usage of XSD.

Business Signals are also used in the context of Activities within Business Transactions. They are used for signaling state that the Transaction, respectively the handling of the document, has reached a certain state at the other party. Business Signals as well as Business Documents are used to achieve state alignment. State Alignment means that both parties have the same knowledge about states and procedures on which they base their decisions for the next steps within the business process.

Business Collaborations offer the functionality to define choreographies reusing, among others, the defined Business Transactions in a particular context. The core elements of choreographies are Activities which are, on the one hand, reused Business Transactions in the form of so-called Business Transaction Activities (BTA) and, on the other hand, Business Collaborations in the form of so-called Collaboration Activities (CA).

ebBP arranges for specifying QoS features at the level of Business Collaborations, Business Transaction Activities, Business Transactions, Requesting/Responding Business Activitys and

transmitted Business Documents. These features, their meanings and their usage are explained in more detail within the corresponding chapters of this report.

In the context of Business Collaborations (BC), multiple roles can be specified. Within Activities, these roles are associated with the roles specified in the defined Business Transactions respectively Business Collaborations that are referenced. Furthermore, for each Business Collaboration and each Business Transaction Activity the time to perform the work can be specified. Modeling a choreography means starting at a defined point, having alternative paths for walking through the process, and ending in final states. Therefor, each Business Collaboration contains exactly one `Start` element which references the starting activity. After each activity an ebBP Decision element can be used to examine the result of the activity and to determine the next steps. A Decision can have multiple outgoing branches to following BTAs or CAs.

Fork and Join are further elements to describe the control flow within a collaboration. They are used, in order to define parallel executions with AND (Join), OR (Join, Fork) or XOR (Fork) semantics.

At the end of each path of the Business Collaboration, there has to be a Completion State to signal the end of the choreography. Different types of Completion States are distinguished and describe the result of the Business Collaboration. Therefor, ebBP offers a `Success` and a `Failure` element. [YWM+06]

Before using an ebBP choreography, the Business Service Interfaces (BSI) of each party have to be configured for doing business according to the specification.

2.3 Web Services

The World Wide Web Consortium (W3C) defines a Web service as

> a software system designed to support inter operable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [BHM+04].

In order to ensure high interoperability of Web services all public interfaces are described in the Web Service Description Language (WSDL[2]) [CCMW02]. The typical approach to define a WSDL document is to declare the used types for the Web service under consideration. Besides the standard types such as `string`, `integer` or `double` it is possible to declare application specific types using XSD-declarations. In this way it is possible to declare complete XML documents as types for Web services.

[2] due to the dependency on BPEL, WSDL 1.1 is used in this work

Using the type declarations, messages can be determined for the WSDL interface. Each message consists of one or more parts whereas one part is associated with exactly one type.

PortTypes are a set of Web service operations and messages used by this operations. WSDL has four types for transmitting messages (see [CCMW02], sec. 2.4):

- One-way: The endpoint receives a message.
- Request-response: The endpoint receives a message, and sends a correlated message.
- Solicit-response: The endpoint sends a message, and receives a correlated message.
- Notification: The endpoint sends a message.

For every of these transmission types the corresponding messages have to be defined as `input`, `output` or `fault` message.

A binding defines message format and protocol for operations and messages defined by a PortType. The binding defines the grammar of the input, output and fault messages as well as additional general information about the interface like PolicyReferences, e.g., to guarantee QoS-Features.

By defining a port, the name, the binding and the URL (Uniform Resource Locator) of a Web service is determined. To group the different ports of a Web service, the service tag is used. For an example WSDL definition see listing 2.1.

```
1  <?xml version ="1.0" encoding ="UTF-8" ?>
2
3  <definitions name="HelloWorld" targetNamespace="http://localhost/HelloWorld" xmlns:tns="
       http://localhost/HelloWorld" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema" "xmlns:soapenc="http://schemas.xmlsoap.org
       /soap/encoding/" "xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.
       xmlsoap.org/wsdl/">
4
5     <message name="helloWorldResponse">
6        <part name="Result" type="xsd:string"/>
7     </message>
8
9     <portType name="HelloWorldPortType">
10       <operation name="helloWorld">
11          <output message="tns:helloWorldResponse"/>
12       </operation>
13    </portType>
14
15    <binding name="HelloWorldBinding" type="tns:HelloWorldPortType">
16       <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
17             <operation name="helloWorld">
18          <soap:operation soapAction="urn:helloWorld#helloWorld"/>
19          <output>
20             <soap:body use="encoded" namespace="urn:helloWorld" encodingStyle="http://
                   schemas.xmlsoap.org/soap/encoding/"/>
21          </output>
22       </operation>
23    </binding>
24
25    <service name="HelloWorldService">
```

```
26       <port name="HelloWorldPort" binding="HelloWorldBinding">
27          <soap:address location="http://localhost/HelloWorld.php"/>
28       </port>
29    </service>
30 </definitions>
```

Listing 2.1: Example WSDL for a HelloWorld Web service

To find Web services of unknown service provider, the Universal Description, Discovery and Integration (UDDI) [BCvR03] provides a XML-based directory service which enables service requesters to find Web services using variable research criteria. The interaction of a Web service with other systems is handled over SOAP as depicted in figure 2.3.

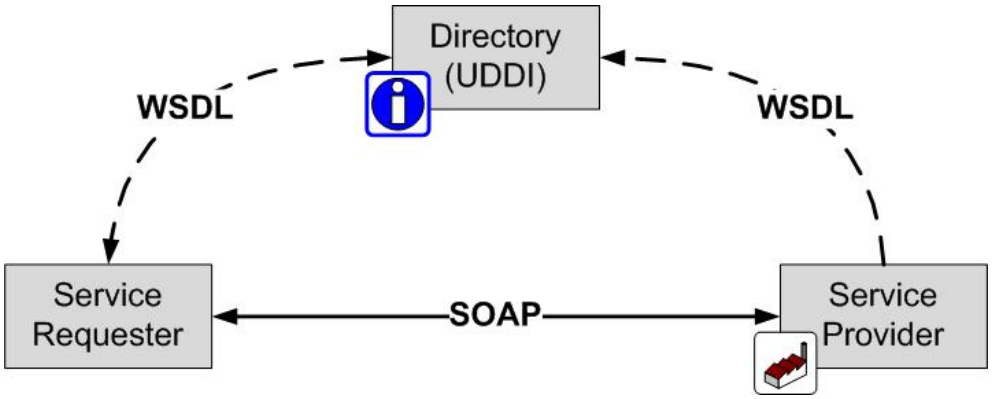


Figure 2.3: The Web Services Architecture

2.4 Web Services Business Process Execution Language

WS-BPEL is an OASIS standard which is used for modeling the behavior of business processes and based on XML specifications such as, among others, XPath 1.0 or WSDL 1.1. In this work, the latest WS-BPEL version 2.0 is used.

The OASIS Standard describes WS-BPEL as

> a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web service interfaces, and the structure of the relationship at the interface level is encapsulated in what is called a partnerLink. The WS-BPEL process defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination. WS-BPEL also introduces systematic mechanisms for dealing with business exceptions and processing faults. Moreover, WS-BPEL introduces a mechanism to define how individual or composite activities within a unit of work are to be compensated in cases where exceptions occur or a partner requests reversal. [JEA+07]

The core elements of BPEL are:

- Receive: Receipt of a message sent by a call of a service; provided by the BPEL process;
- Invoke: Invocation of a Web service with input and optional output variables;
- Reply: Response of the BPEL process to a partner calling the Web service offered by the BPEL process; always corresponds to a Receive;
- Scope: a context which offers particular conditions and functions for the enclosed activities like handlers or variables;
- Fault Handler: Handling faults within the linked scope using constructs like catch and catch all;
- Event Handler: Listener for particular messages; defined within a scope;
- OnAlarm: defines a timer within a scope; fired when a specified time out occurs;
- Assign: Operation to copy messages or parts of messages, e.g., for copying variables for a subsequent Invoke;
- Partnerlink: Representation of a WSDL interface of the BPEL process or of referenced Web services;
- CorrelationSet: A set of attributes to ensure the correlation of messages and process instances;

In addition to this constructs, there are structured activities, such as loops (if, while, repeat until etc.), scopes, sequences and so on. For a detailed introduction into WS-BPEL see [JEA+07].

An example of a BPEL process managing an order is given in figure 2.4 in the form of a Business Process Modeling Notation (BPMN) diagram.

The semantics of this BPEL process is as follows: The PartnerLink on the left side (CustomerOrderIF) specifies the input, output and fault variable for this process. The input variable (an XML file consisting of a credit card information and information about a order with different order items) is the input for the Receive of the BPEL process, the following Assign1 copies the information of the input variable of the Receive into the input variable for the first PartnerLink on the right side (CCCheckIF), which checks the credit card information, so that the PartnerLink can be called by the BPEL element Invoke1. This PartnerLink returns either true, if the credit card information is correct and the order can be paid with this credit card, or false otherwise. But if wrong information is transmitted, e.g., a non-valid credit card number, a fault will be thrown. The BPEL process defines a fault handler which catches all faults and rethrows them so the fault will be handed over to the CustomerOrderIF without further fault handling.

The output of the CCCheckIF PartnerLink is the boolean condition for the following if-fork. If CCCheckIF returns false the path on the right-hand side will be chosen which simply copies the reply of the credit card check and an additional text into the output variable (Assign3) of the BPEL process. Otherwise, if CCCheckIF returns true the path on the left-hand side after the

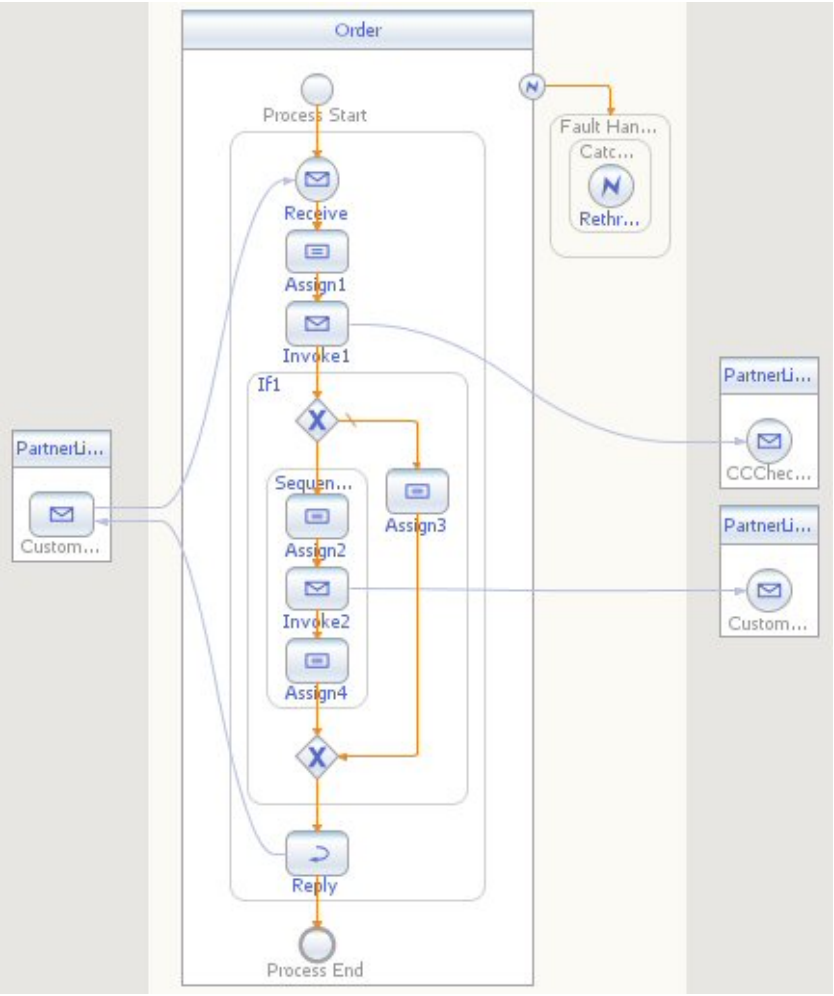


Figure 2.4: A BPEL process example, modeled in BPMN

IF-construct will be taken so that Assign2 prepares the input variable for the real order process by copying the input variable of the Receive. The following Invoke2 invokes the PartnerLink (CustomerRealOrderIF), which confirms the order (true or false) and sends an additional text which contains, e.g., the delivery time. Assign4 copies this information into the output variable of the process.

The output variable of the BPEL process is a boolean with an additional string. The boolean is either true, in case the credit card is valid and the order has been successful, or it is false, in case the credit card is not valid or the order has not been successful. In the first case, the additional information are information about the delivery time etc., in the second case information about the refusal will be given.

2.5 WS-* Standards

WS-* standards is a non-official general term for numerous specifications that extend basic Web service functionality. These Web service technologies provide a lot of new features to Web

services. Many of these WS-* extensions are used to implement QoS-Features.

These specifications are developed by different organizations so they complement or depend on each other as much as they overlap and exist in competition. The WS-* standards mostly are only specifications and the developing organizations do not provide any reference implementations or anything like that. So, frequently, the problem with these standards is to find a fully working and standards conforming implementation.

Most of this WS-* standards do their work in the background without intervention or awareness of the user. SOAP message headers are used by the majority of these specifications for conveying the information needed for implementing a particular functionality. These headers typically are created and processed by the Web service frameworks of the communication partners. So the application of most WS-* standards is completely transparent to users, however this is not always an advantage.

A description of WS-* standards used in this work is given below.

2.5.1 WS-Addressing

WS-Addressing [GHR06] is a WS-* specification which allows Web services to transmit addressing and routing information. For this reason WS-Addressing normalizes addressing information into a uniform format so that the document under transmission can be processed independently of transport or application. The two concepts for providing normalized addressing information are endpoint references and message information headers.

An endpoint reference is used to determine the address of any "endpoint". The only required tag for an endpoint reference is the `Address`-tag, which defines a simple URI of the type `xs:anyURI`. An example of an `EndpointReference` is given in Listing 2.2.

```
1 <wsa:EndpointReference>
2   <wsa:Address>xs:anyURI</wsa:Address>
3 </wsa:EndpointReference>
```

Listing 2.2: Example definition of an `EndpointReference`

The second of the two concepts mentioned above is the message information header. These headers allow uniform addressing of messages independent of underlying transport mechanisms. The uniform addressing contains besides the two required elements `To` (destination of the message) and `Action` only optional elements. These optional elements are: The `MessageID` which identifies a message by an URI and which can be referenced in the `RelatesTo`-tag. The `From`-tag and the `ReplyTo`-tag determine the endpoint reference of the sender and the reply address for the communication. By defining a `FaultTo` a fault endpoint can be defined, but in case of existence of a `FaultTo`-tag a message id has to be set. An example for a complete definition for the message information header is given in Listing 2.3.

```
1   <wsa:MessageID> xs:anyURI </wsa:MessageID>
2   <wsa:RelatesTo RelationshipType="..."?>xs:anyURI</wsa:RelatesTo>
3   <wsa:To>xs:anyURI</wsa:To>
4   <wsa:Action>xs:anyURI</wsa:Action>
5   <wsa:From>endpoint-reference</wsa:From>
```

```
6   <wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
7   <wsa:FaultTo>endpoint-reference</wsa:FaultTo>
```

Listing 2.3: Example definition of the Message Information Header

2.5.2 WS-Security

Initially developed by IBM, Verisign and Microsoft, WS-Security [OAS06] is officially called Web Services Security by the Oasis Open Group. It is a specification of how to use different security mechanisms (signature, encryption, security time stamp) for a business document which is being transmitted to make it tamper-proof. Among the most important mechanisms are signatures and encryption headers as well as binary security tokens and Kerberos certificates. These security mechanisms are transmitted using the message header. Related WS specifications are, among others, WS-Policy [VOH+07], WS-Trust [LKN+07], XML Signature [ERS+08] and XML Encryption [ERI+02].

2.5.3 WS-ReliableMessaging

This standard is designed to guarantee reliable messaging between two partners. WS-Reliable Messaging [FPD+07] avoids duplicated messages, lost messages and cares for an ordered delivery of messages, if requested. WS-ReliableMessaging requires WS-Addressing to determine the endpoint references which are the identifiers for the sender and the receiver of every message.

The typical approach of a communication between to partners (A as initiator resp. source, B as destination) is that A creates a session and B responds to this `CreateSequence`-message by acknowledging this message and assigning an identifier to the session. After that, A sends messages with this sequence-identifier and a message number in this sequence (this information has to be included in the Sequence-Block of the SOAP-header of every message, which should be delivered reliably) and A can request an acknowledgement for every single message or a sequence of messages. Using these acknowledgements, lost messages can be identified and sent again by A to B. This cycle of sending messages and acknowledging them will be repeated until all messages are delivered form A to B. To close a session after transmission of all messages to be sent, A has to send a `TerminateSequence`-message to B who acknowledges the receipt of the sequence termination. To close a session without completing it, one of the two parties can send a `CloseSequence`-message which also has to be acknowledged. For an illustration of this interrelation, see figure 2.5

The WS-ReliableMessaging standard defines four delivery assurances which can be supported by source and destination, these are (see [FPD+07], section 2.4):

- AtLeastOnce: Each message is to be delivered at least once, or else an error MUST be raised by the RM [(ReliableMessaging)] Source and/or RM Destination. The requirement on an RM Source is that it SHOULD retry transmission of every message sent by the Application Source until it receives an acknowledgement from the RM Destination. The

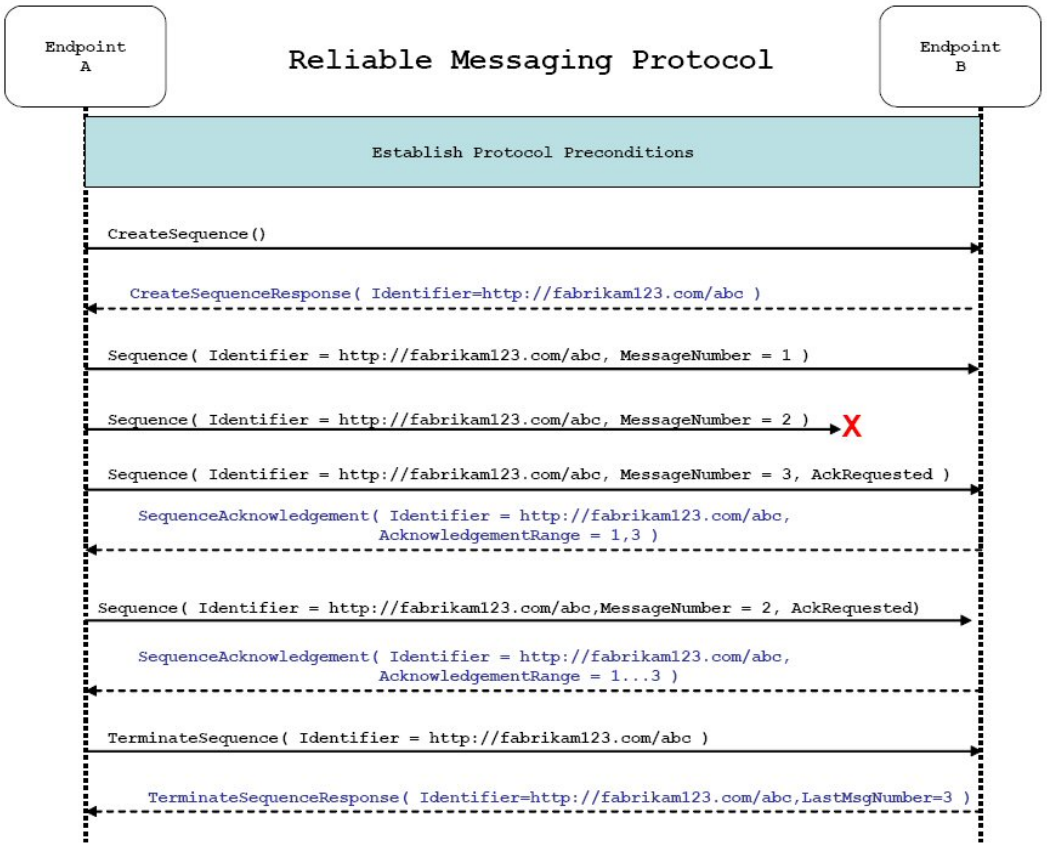


Figure 2.5: The WS-ReliableMessaging protocol, taken from [FPD+07]

requirement on the RM Destination is that it SHOULD retry the transfer to the Application Destination of any message that it accepts from the RM Source, until that message has been successfully delivered. There is no requirement for the RM Destination to apply duplicate message filtering.

- AtMostOnce: Each message is to be delivered at most once. The RM Source MAY retry transmission of unacknowledged messages, but is NOT REQUIRED to do so. The requirement on the RM Destination is that it MUST filter out duplicate messages, i.e. that it MUST NOT deliver a duplicate of a message that has already been delivered.

- ExactlyOnce: Each message is to be delivered exactly once; if a message cannot be delivered then an error MUST be raised by the RM Source and/or RM Destination. The requirement on an RM Source is that it SHOULD retry transmission of every message sent by the Application Source until it receives an acknowledgement from the RM Destination. The requirement on the RM Destination is that it SHOULD retry the transfer to the Application Destination of any message that it accepts from the RM Source until that message has been successfully delivered, and that it MUST NOT deliver a duplicate of a message that has already been delivered.

- InOrder: Messages from each individual Sequence are to be delivered in the same order they have been sent by the Application Source. The requirement on an RM Source is that it MUST ensure that the ordinal position of each message in the Sequence (as indicated

by a message Sequence number) is consistent with the order in which the messages have been sent from the Application Source. The requirement on the RM Destination is that it MUST deliver received messages for each Sequence in the order indicated by the message numbering. This DeliveryAssurance can be used in combination with any of the AtLeastOnce, AtMostOnce or ExactlyOnce assertions, and the requirements of those assertions MUST also be met. In particular if the AtLeastOnce or ExactlyOnce assertion applies and the RM Destination detects a gap in the Sequence then the RM Destination MUST NOT deliver any subsequent messages from that Sequence until the missing messages are received or until the Sequence is closed.

Any of these actions will not be noticed by the user, because the WS-ReliableMessaging instructions are put in the SOAP-Message header. Another OASIS standard for a similar functionality is WS-Reliability, but WS-ReliableMessaging is the more current standard and there currently are more implementations for WS-ReliableMessaging available. Hence, WS-Reliability will not be further considered within the work at hand.

2.5.4 WS-Policy

This WS-Specification [VOH+07] provides mechanisms that enable web services to "announce" requirements or capabilities concerning important process parameters, e.g., security features, version numbers, response timeouts etc. This is to aid in secure and reliable message exchange between services.

Web services may specify a number of policies which have to be supported by the requesting service. In general, a policy consists of at least one policy alternative, a so-called policy assertion. A policy is supported when at least one of its alternatives is supported by the requesting service.

2.5.5 XML Signature

Actually, XML-Signature [ERS+08] is not a WS-* standard, but is strongly connected with XML files which are transmitted by Web services. This standard is one of the standards which enable QoS-Features for business to business communication, so this standard will be presented here.

XML Signature is a W3C standard that allows to add a signature to an XML document. A document is signed with a signature by the sender with his secret, private key and can be validated with the corresponding public key which is contained or referenced in the XML signature. The signature can be validated successfully if and only if the XML document has not been altered in any way. Even a change in namespaces or an incrementation of any value alters the document in a way that invalidates the XML Signature. The signature element can be enveloped in the XML document to be signed or detached in an external document.

Listing 2.4 shows the structure of a XML Signature; the elements will be explained thereafter.

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <SignatureMethod />
    <CanonicalizationMethod />
    <Reference>
      <Transforms>
      <DigestMethod>
      <DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue />
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
  <Object />
</Signature>
```

Listing 2.4: The structure of an XML Signature

An XML Signature consists of four main elements which are `SignedInfo`, `SignatureValue`, `KeyInfo` and `Object`. The functionality of the single elements of a XML Signature is described below:

- `SignedInfo`: The `SignedInfo` is used to reference the signed data and specify the algorithms, that are used to create this signature. Core elements are the `SignatureMethod` (determines the algorithm to be used to sign the document) and the `CanonicalizationMethod` (determines the canonicalization method of this XML document). These two elements are used to generate the `SignatureValue`. The `Reference` element has to occur at least once and specifies the `DigestMethod`, the `DigestValue` and the object to be signed.

- `SignatureValue`: This element contains the Base64 encoded result of the signature.

- `KeyInfo`: This is an optional element, which contains information about the public key for re-creating the public key. This key is needed to validate the signature.

- `Object`: In this optional element the signed data are contained in case of an enveloped signature.

Since Java 1.6, XML Signatures can be created using the Java API. Since Java 1.6 Update 10, a bug concerning the XML namespaces of signed documents is fixed. As the implementation for the work at hand uses functionality affected by this bug the recommended Java API to use XML Signatures is Java 1.6 Update 10.

2.6 JAXB

The Java Architecture for XML Binding (JAXB)[3] is a Java programming interface which allows to generate Java classes from XSD (XML Schema Document) files. JAXB defines rules for deriving Java class structures from XSD files and vice versa in the form of bindings. These Java class hierarchies defined for particular XSD definitions can then be integrated in any Java application. XML files can be transformed into Java objects and Java objects into XML documents according to these bindings. JAXB can convert several types of XML representations into Java, in particular files or documents of the type `org.w3c.dom.Document`. These XML representations are unmarshaled (de-serialized) into the corresponding JAXB representations (of the type `javax.xml.bind.JAXBElement<T>`) so that the tags of the XML document can be accessed as Java objects.led (de-serialized) into the corresponding JAXB representations (of the type `javax.xml.bind.JAXBElement<T>`) so that the tags of the XML document can be accessed as Java objects. After processing in Java is done, the JAXB Object Tree can be serialized again by the mechanism of marshaling. A JAXBElement respectively the tree of Java objects can be marshaled into a XML representation such as files or documents. This proceeding is visualized in figure 2.6.

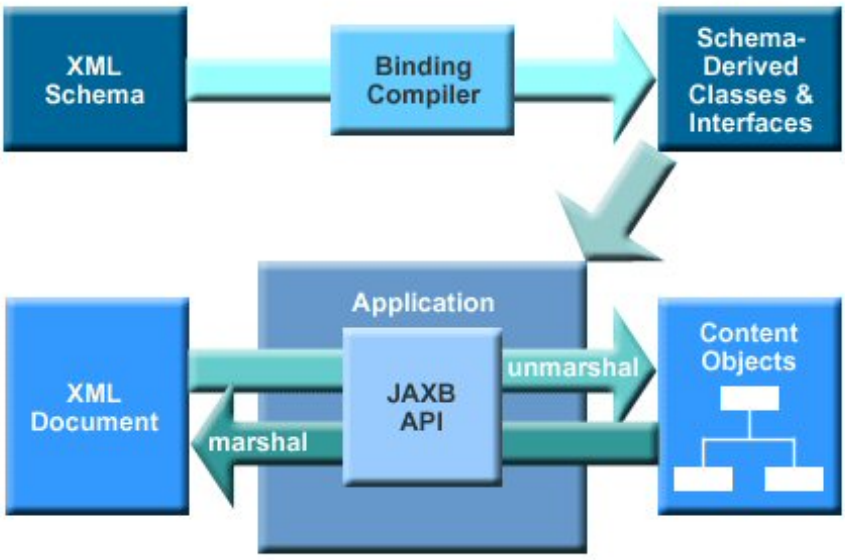


Figure 2.6: The typical approach of using JAXB, taken from [OM03]

A requirement for the usage of JAXB is the availability of all XSD files which are referenced by the XML file to be parsed. This includes all XSD files which are referenced by the original XSD file of the XML Document to be read. Using these XSD files, it is possible to create a JAXB-Binding of the XSD file under consideration which defines a class structure for representing XML documents.

The first step in creating JAXB objects in Java is to define the corresponding `JAXBContext`. With the help of this context, an `ObjectFactory` can be instantiated. This factory is able to

[3] `http://jcp.org/en/jsr/detail?id=222`

create the objects defined by the JAXB Binding and the generic wrapper element of the whole XML file (`javax.xml.bind.JAXBElement<T>`).

Chapter 3

Analysis

3.1 Modeling of NES-Profiles Using ebBP

Before organizations are able to integrate their processes, it is sensible to specify a global interaction protocol with messages and roles of the business process to be integrated. Such a specification is called choreography. The same is done in the work at hand. This chapter describes the modeling of choreographies using the ebXML Business Process Specification Schema as language and the e-business profiles proposed by the NES as input.

3.1.1 Modeling Practices Used

Before describing the modeling of the profiles, the general modeling practices, modeling constructs and their usage in context of this work are explained.

3.1.1.1 The General Approach

The Northern European Subset (NES) proposes eight profiles for doing e-business as already pointed out in chapter 2.1. These profiles describe business processes like sending catalogues, orders, billing information and credit notes by using UBL conforming documents. Put simply, a UBL conforming document is a document that provides all mandatory UBL elements. The usage of optional fields of these documents can vary in context of the different profiles. So, the state of the NES as subset of the UBL is retained.

The textual and visual description of these profiles and of the used documents are the basis of the following modeling activities.

The NES profiles are chosen as use case of this work because these profiles are real world B2Bi processes that are applied in practice in several northern European countries. Showing to be able to handle these scenarios therefore enhances the validity of the work at hand. These

profiles provide detailed schemas for business documents and the business logic that determines the control flow of their exchange. Note that developing business document schemas and determining business logic of business processes are not core subjects of investigation of the work at hand.

In a first step, the profiles are modeled as a choreography between the partners of business integration. For this purpose, the ebXML Business Process Specification Schema is chosen. The main arguments for using ebBP are the possibility to define B2Bi-relevant QoS attributes, the suitability of ebBP choreographies as basis for agreement among integration partners, and the dedication of this standard to B2Bi modeling which promises a sufficient toolset for specifying concise B2Bi process descriptions. During modeling, not only textual ebBP specifications are created but these are visualized using UML Activity Diagrams as well, especially for capturing the control flow. Modeling begins with creating these Activity Diagrams for reasons of comprehensibility and human readability. Subsequently, these diagrams are transformed into ebBP Business Collaborations. So the UML Activity Diagrams are, on the one hand, the basis for modeling but also, on the other hand, the basis for validating the ebBP Collaborations and to check their correctness. Modeling of ebBP is the essential part of this step, while UML Activity Diagrams are only used as a tool.

Modeling in the context of this work means analyzing the different profiles, filtering the relevant informations for the choreography and the integration of the business partners, and finally to complete the flow of informations in order to ensure state alignment between the business partners. State alignment is one of the most important requirements, i.e., to ensure consistent information at each party and to ensure that both parties base their decisions for next steps on the same knowledge. Additional application responses and error notifications are introduced for this purpose. All business documents used and transmitted are instances of the documents recommended by the NES.

In the following, the modeling practices used, the modeling of the profiles and the problems during the process of modeling are explained more closely.

3.1.1.2 ebBP Modeling Elements

This section shows the elements of the ebBP specification used in this project and their application within the modeling of the profiles.

Modeling business processes in this context means specifying XML files according to the XSDs of ebBP.

As shown in figure 3.1, the Business Process Specification Schema is based on Business Transactions which define flows of documents and signals in requesting and responding direction between two parties. These Business Transactions are reused as Business Transaction Activities within the flow of a Business Collaboration which can itself be reused as Collaboration Activity. Each Business Transaction Activity is performed by two roles. For this purpose, it associates the roles of its referenced Business Transaction with the roles specified within the Business Collaboration. The flow of a Business Collaboration is modeled as transitions between

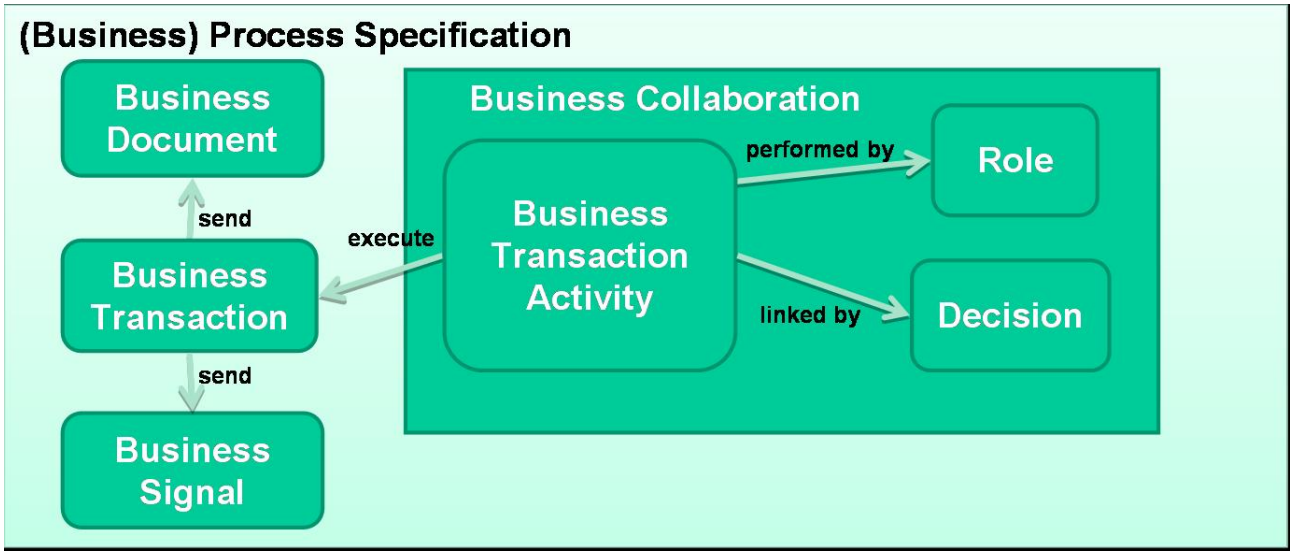


Figure 3.1: The constructs of ebBP used in this work

the mentioned activities using Decisions, Forks and Joins. In this work, only Decisions are used for connecting activities. For this reason Forks and Joins are not discussed in this chapter.

Furthermore, every Business Collaboration contains exactly one `Start` element and at least one final state which represents the end of the collaboration.

All elements are identified by a unique name id which is also used to reference the elements.

In the following paragraphs, all used ebBP elements and their usage within this use case are discussed.

Modeling of Business Signals Business Signals are used by a party to signal the partner that the handling of a received business document has reached a certain state. Business Signals can either be positive or negative. Negative Business Signals indicate that an exception has occurred. The following Business Signals are in use:

- ReceiptAcknowledgement signals a positive receipt of a business document and indicates successful validation of the business document against a specified XSD; If an error occurs, a ReceiptAcknowledgementException is sent back instead.
- AcceptanceAcknowledgement signals acceptance for business processing of the received business document. This typically indicates that the document conforms to business rules defined by the partner which might be implemented using Schematron validations. If an error occurs, an AcceptanceAcknowledgementException is sent back instead.
- GeneralException signals the occurrence of an exception, that is not covered by the two items above;

Business Signals are reused within the Requesting and Responding Business Activities of a Business Transaction. The usage of these signals depends on the Quality of Service attributes

specified for the Business Transaction and for its subelements. These attributes are discussed in more detail in chapter 4.1.2.

```
<Signal name="ReceiptAcknowledgement" nameID="ra2">
        <Specification
                location="http://docs.oasis-open.org/ebxmlbp/ebbp-signals-2.0"
                name="ReceiptAcknowledgement" nameID="rabpss2"/>
</Signal>
```

Listing 3.1: Example definition of Business Signals

Listing 3.1 shows an example of the definition of a Business Signal. Each Business Signal is identified and can be referenced by a unique `nameID`. The attributes of the `Specification` element specify the namespace of the corresponding XSD file and describe the Signal by its name and its unique identifier.

Modeling of Business Documents Like Business Signals, Business Documents are also identified with a unique `nameID` where the work at hand uses the `nameID` pattern `bd_name_of_document`. The `Specification` element specifies the namespace of the type definition file and its location. The `type` attribute specifies the type definition language, e.g., `schema` or `dtd`. Listing 3.2 gives an example of the definition of Business Documents.

```
<BusinessDocument name="catalogue" nameID="bd_catalogue">
        <Specification location="../../schemas/profile1/maindoc/UBL-Catalogue-2.0.xsd"
                name="basicNES2_0Catalogue" nameID="basicNES2_0Catalogue" type="schema"
                targetNamespace="urn:oasis:names:specification:ubl:schema:xsd:Catalogue-2"
                    />
</BusinessDocument>
```

Listing 3.2: Example definition of Business Documents

Business Documents are used by Business Transactions which encapsulate them within their Requesting or Responding Business Activities.

Modeling of Business Transactions As pointed out above, Business Transactions are representing one of the core elements of the ebBP specification. Listing 3.3 gives an example of the specification of a Business Transaction.

```
<InformationDistribution name="distributeCatalogue" nameID="bt_distributeCatalogue"
        isGuaranteedDeliveryRequired="true">
        <RequestingRole name="initiator" nameID="bt_distributeCatalogue_role_initiator">
        </RequestingRole>
        <RespondingRole name="responder" nameID="bt_distributeCatalogue_role_responder">
        </RespondingRole>
        <RequestingBusinessActivity name="sendCatalogue"
                nameID="bt_distributeCatalogue_ba_req" isAuthorizationRequired="true"
                isIntelligibleCheckRequired="true" retryCount="3"
                timeToAcknowledgeReceipt="PT1H">
                <DocumentEnvelope name="catalogue" businessDocumentRef="bd_catalogue"
                        nameID="bt_distributeCatalogue_doc_catalogue"
                        isAuthenticated="transient" isConfidential="transient"
                        isTamperDetectable="transient">
```

```
15                  </DocumentEnvelope>
16                  <ReceiptAcknowledgement name="ra" nameID="bt_distributeCatalogue_ack_ra"
17                          signalDefinitionRef="ra2">
18                  </ReceiptAcknowledgement>
19                  <ReceiptAcknowledgementException name="rae"
20                          nameID="bt_distributeCatalogue_ack_rae" signalDefinitionRef="rae2">
21                  </ReceiptAcknowledgementException>
22          </RequestingBusinessActivity>
23          <RespondingBusinessActivity name="provide acks"
24                  nameID="bt_distributeCatalogue_ba_resp">
25          </RespondingBusinessActivity>
26  </InformationDistribution>
```

Listing 3.3: Example definition of Business Transactions

A Business Transaction is an abstract type for different kinds of transactions. The concrete type to be used depends on the context of use. Each Business Transaction (BT) has exactly one Requesting Business Activity and one Responding Business Activity. Depending on the concrete type of a Business Transaction, one-way or two-way interactions can be realized by a BT. A one-way interaction is realized with an empty Responding Business Activity. Furthermore, within the Responding Business Activity of some types of Business Transactions, multiple Document Envelopes can be specified which then are to be used alternatively. Requesting and Responding Business Activities are representing Document and Signal flows in corresponding directions. The Requesting Business Activity is started by the so-called Requesting Role of a Business Transaction whereas the Responding Business Activity is started by the corresponding Responding Role. The direction of Business Document and Business Signal flows can then be determined accordingly.

The following types of BusinessTransactions are used in the work at hand:

- InformationDistribution: An `InformationDistribution` is used in order to send information with business contents which are not legally binding to the receiving party. These informations are answered (if necessary) with an `AppResponse` sent in the context of another `InformationDistribution`. In this work, this type of Business Transaction contains only a Requesting Business Activity and an empty Responding Business Activity and thus represents a one-way message transmission. An example is shown in Listing 3.3
- CommercialTransaction: A `CommercialTransaction` is used in case the business documents and their consequences are legally binding to both parties. These documents must be confirmed also by an answer with binding character. Each `CommercialTransaction` of this work has two alternative Responding Business Activities, one for positive and one for negative response using the Business Document `AppResponse`. To state a positive response, the attribute `isPositiveResponse` is set to true in the `DocumentEnvelope`. Sending an invoice is an example for how to use this kind of Business Transaction.
- Notification: `Notifications` are used to send `AppResponses` or error messages with legally binding character which can not be transmitted using `CommercialTransactions`. An example for sending a `Notification` is the occurrence of an exception at one party.

There are several possibilities to set Quality of Service attributes within a Business Transaction. This can be done at the level of Business Transaction, Requesting/Responding Business Activity

and at the level of the Document Envelope. The following listings show the different features used in the work at hand.

QoS features at the level of Business Transactions:

- `isGuaranteedDeliveryRequired` is set to true in order to transmit the documents of the Business Transaction reliably. In this work, it is always set to true because reliable transmission of Business Documents is always important, even if they do not have legally binding character.

QoS features at the level of `RequestingBusinessActivity` and `RespondingBusinessActivity` can vary if necessary, e.g., if the transmitted documents differ in QoS requirements.

- `isAuthorizationRequired` is set to true in order to guarantee that only documents from well-known partners are accepted and that the sending entity is indeed eligible for sending documents of the corresponding type.
- `isIntelligibleCheckRequired` is set to true in order to guarantee the confirmation of a message only if it is legible, i.e., only if structural and type validations succeed.
- `isNonRepudiationRequired` is set to true in order to require non-repudiation. This means the sending party cannot deny having sent a document.
- `isNonRepudiationReceiptRequired` is set to true in order to guarantee the sending of a receipt when a message has been transmitted. The receipt of a document cannot be denied.
- `retryCount` specifies the number of document transmission attempts. If an attempt to transmit a document fails another attempt to send this document is made until the maximum number of retries is reached. This work always makes three attempts.

QoS features at the level of `DocumentEnvelope` can be specified as transient, to use implementation at the network layer, or as persistent, to use implementation at the process layer, or as both or as none.

- `isAuthenticated` is set to require authentication of the document's sender.
- `isConfidential` is set to require encryption in order to prevent third parties reading the document.
- `isTamperDetectable` is set to require checksums. This guarantees manipulations of a document during transmission can be detected.

Modeling of Business Collaborations The choreography between two ore more integration partners is specified within the context of a Business Collaboration. This work only uses binary collaborations, i.e., integration between two partners. Like all elements of an ebBP document, a Business Collaboration is also identified by an unique `nameID` and is described by its name. To state whether a collaboration can only be used as Collaboration Activity within another Business Collaboration the attribute `isInnerCollaboration` could be set to `true` inside the `BusinessCollaboration` element. By default this value is set `false`.

Listing 3.4 shows an example of a Business Collaboration. At first, roles are specified by name and `nameID`. In general multiple definitions of roles with in a choreography can be specified. This work handles only binary collaborations and thus always defines only the two roles of customer and supplier as intended by the NES profiles to be modeled.

Furthermore, the expected duration of the collaboration is specified by the element `TimeToPerform`. This work distinguishes between two kinds of definition. On the one hand, the duration can be specified at design time with a fixed duration as shown in listing 3.4. On the other hand, a duration could be negotiated at runtime. Then, no duration will be specified and the type will be set to `runtime`. The usage of runtime configuration is preferable for inner collaborations. The third possibility provided by ebBP of setting the type to `configuration` is not used in this work.

Finally, it is necessary to specify the start of the choreography within the element `Start`. The `ToLink` element references the first activity which could be a Business Transaction Activity or a Collaboration Activity. References are set using `nameIDs`.

The flow of the collaboration is, as mentioned, specified by transitions between decisions and activities with final states at its end. For more details see the corresponding paragraphs.

```
<BusinessCollaboration name="NES profile 1 collaboration" nameID="cb_profile1global">
        <Role name="Customer" nameID="cb_profile1global_role_customer"></Role>
        <Role name="Supplier"   nameID="cb_profile1global_role_supplier"></Role>
        <TimeToPerform type="design" duration="P10D"></TimeToPerform>
        <Start name="start of NES profile 1 collaboration"
                nameID="start_cb_profile1global">
                <ToLink toBusinessStateRef="bta_sendCatalogue"></ToLink>
        </Start>
        ...
</BusinessCollaboration>
```

Listing 3.4: Example definition of Business Collaborations

Modeling of Business Transaction Activities Business Transaction Activities reuse the defined Business Transactions within the flow of a Business Collaboration. An example of a Business Transaction Activity is given in listing 3.5. Like for all other elements, `nameId` and `name` are used to identify the activity. The Business Transaction Activity references the Business Transaction to be reused by its `nameId` within the attribute `businessTransactionRef`.

The `TimeToPerform` element specifies the expected time to perform this Business Transaction Activity. Like the TimeToPerform of Business Collaborations, in this context, it can also be of type `design` or `runtime`. A fixed time is specified in case of type `design` and is used in outer

Business Collaborations. Within the content of an inner Business Collaboration the TimeToPerform of a Business Transaction Activity depends of the enclosing Business Collaboration. Thus, it has also to be specified at runtime. This work doesn't use the type `configuration` for defining the TimeToPerform.

The `Performs` elements determine the association of Business Collaboration roles with Business Transaction roles. This means, it determines which role of the Business Collaboration has to perform a particular role of the Business Transaction. The attributes `currentRoleRef` and `performsRoleRef` are specifying this association.

```
<BusinessTransactionActivity
        businessTransactionRef="bt_distributeCatalogue" name="send catalogue"
        nameID="bta_sendCatalogue" hasLegalIntent="false"
        isConcurrent="true">
        <TimeToPerform duration="PT1H" type="design"></TimeToPerform>
        <Performs currentRoleRef="cb_profile1global_role_supplier"
                performsRoleRef="bt_distributeCatalogue_role_initiator">
        </Performs>
        <Performs currentRoleRef="cb_profile1global_role_customer"
                performsRoleRef="bt_distributeCatalogue_role_responder">
        </Performs>
</BusinessTransactionActivity>
```

Listing 3.5: Example definition of Business Transaction Activities

Business Transaction Activities also offer the possibility to specify Quality of Service attributes. These attributes are shown in the listing below:

- `hasLegalIntent` is set to signal whether the transmitted document has legal intent and thus is legally binding for the receiver.
- `isConcurrent` is set to enable parallel execution of the Business Transaction Activity under consideration.

Modeling of Business Collaboration Activities Collaboration Activities describe the execution of inner Business Collaborations. This means, every Collaboration Activity has to reference a Business Collaboration using the `nameID` (See listing 3.6 at attribute `collaborationRef`). Furthermore, a Collaboration Activity uses the attributes `name` and `nameID` to describe its own identity. It is also necessary to associate the roles of the calling Business Collaboration with the roles they have to perform in the called Business Collaboration. For this purpose, the `Performs` element offers the attributes `currentRoleRef` and `performsRoleRef`. These attributes hold references to the specified roles using their `nameID`.

```
<CollaborationActivity name="collaboration activity external send invoice"
        nameID="ca_ext_sendInvoice" collaborationRef="cb_inner_ext_sendInvoice">
        <Performs currentRoleRef="cb_profile5global_role_supplier"
                        performsRoleRef="cb_inner_ext_sendInvoice_role_supplier">
        </Performs>
        <Performs currentRoleRef="cb_profile5global_role_customer"
                        performsRoleRef="cb_inner_ext_sendInvoice_role_customer">
        </Performs>
</CollaborationActivity>
```

Listing 3.6: Example definition of Collaboration Activities

Modeling of Decisions Another central element of modeling the flow within a Business Collaboration is the Decision. This use case uses Decisions only to link the different activities with each other. For this purpose, Decisions use the element `FromLink`, to specify the activity that should be linked, and the element `ToLink` to specify the different target elements. A Decision can contain multiple `ToLink` definitions. Which target branch is chosen, depends on the condition, that is specified within the `ToLink` element. This shows the second purpose of Decisions, i.e., to ensure the correct routing after each Business Transaction Activity depending on its outcome. This routing is important for state alignment between the business partners. This means, both parties have to take the same routing decisions.

The reference to the activity that should be linked is set within the `FromLink` element using the attribute `fromBusinessStateRef`. This attribute uses the name id of the activity.

The targets within the `ToLink` elements are also referenced by the name id of the destinations. Destinations can be final states as well as activities and are determined by attribute `toBusinessStateRef`.

For routing the flow of a Business Collaboration the ebBP Specification offers a number of possibilities of expression languages. This work only uses the languages explained more closely below. Conditions are specified within the element `ConditionExpression`. The expression to evaluate depends on the given `expressionLanguage`.

- `expressionLanguage = ConditionGuardValue`: A guard value is used for evaluation that captures BTA results predefined by ebBP. It is checked against a concrete specified value. If it evaluates to true, than the next step is determined by the `toBusinessStateRef` attribute of the enclosing `ToLink` element.
- `expressionLanguage = XPath2`: This specifies that a boolean XPATH2[1] `expression` has to be applied to the handled business document. If this expression evaluates to true, this path is chosen.

If multiple expressions evaluate to true, the first expression listed determines the flow.

Listing 3.7 gives an example that shows the application of all possible expression languages.

```
1  <Decision name="after send appResponse" nameID="dec_sendAppResponse">
2          <FromLink fromBusinessStateRef="bta_sendAppResponse"></FromLink>
3          <ToLink toBusinessStateRef="techFail">
4                  <ConditionExpression expressionLanguage="ConditionGuardValue"
5                                       expression="AnyProtocolFailure"></
                                           ConditionExpression>
6          </ToLink>
7          <ToLink toBusinessStateRef="success">
8                  <ConditionExpression expressionLanguage="XPath2"
9                                       expression="normalize-space(string(//
                                           DocumentResponse/Response/ResponseCode))='
                                           ACCEPT'">
10                 </ConditionExpression>
11         </ToLink>
12         <ToLink toBusinessStateRef="businessFail">
```

[1] http://www.w3.org/TR/xpath20/

```
13                  <ConditionExpression expressionLanguage="XPath2"
14                                       expression="normalize-space(string(//
                                             DocumentResponse/Response/ResponseCode))='
                                             REJECT'">
15                  </ConditionExpression>
16          </ToLink>
17  </Decision>
```

Listing 3.7: Example definition of Decisions

Modeling of Final States Final States are representing possible endings of a choreography. ebBP distinguishes between two kinds of Final States. In this work, every business process can have exactly one `Failure` state, which is represented by the corresponding element. In contrast to that, there can be multiple `Success` states in a Business Collaboration, also represented by the corresponding element. Each Final State is identified and referenced by a unique `nameID`. The semantics of a Final State is expressed using the `name` attribute.

This work differentiates between the following three types of Final States:

- success: signals a successful protocol termination as well as a successful business result;
- businessFail: signals a successful protocol termination with a non successful business result;
- techFail: signals a failure state of the protocol execution;

3.1.1.3 UML Constructs

UML Activity Diagrams are chosen in order to visualize the business protocol and to visualize all alternative paths through the process. Modeling UML is not necessary for creating ebBP models. But in this work, the modeling is used to visualize the Collaboration. This can be the basis for modeling in XML as well as the basis for informally validating the modeled ebBP Business Collaboration.
Only a few elements of UML Activity Diagrams are used to describe a simple idea of the flow of the particular profiles.

- **Actions** are used to show all activities with the purpose of sending or receiving business documents or with the purpose of starting other new processes of Business Collaborations.
- **Decisions** are used after each activity. Depending on the outcome of the previous activity and its context of use, a decision can have multiple outgoings. The conditions for using an outgoing path are annotated at the outgoing arrows.
- A **Starting** Node signals the start of a Business Collaboration. Each process has exactly one starting node.

- **Ending Node** (for activities) is used to signal the end of the process. Multiple ending nodes can occur in a process and could have different meanings like protocol failure, business success or business failure. The meaning of an ending node depends on the path through the decisions before and is annotated as comment or name of the ending node.

3.1.2 Critical Modeling Issues

In this section, critical modeling issues will be presented in three steps. The first step contains the problem description of a critical modeling issue, the second step describes possible solution alternatives and the third step presents the selected solution along with a justification for choosing this solution.

The aim of this section is to address each critical modeling issue separately, to analyze them in isolation and to do an impact analysis of the solution of choice on the other issues afterwards. But especially for the first two problems a coordinated solution is necessary due to the strong interdependency in the possible solution combination. The external medium problem, however, is addressed in isolation.

As can be seen in the next paragraphs, a perfect solution frequently cannot be found. This results in several trade-offs that guarantee the feasibility as well as involve several drawbacks. This is the reason why all the solution alternatives are presented and the argument for a specific alternative is shown in detail to describe the line of thought that has led to a particular solution.

3.1.2.1 The Parallel Execution Problem

The parallel execution problem occurs in the NES Profiles 5 to 8. After receiving an application response with the code `incorrect information` in the process flow, a credit note to nullify the previous invoice as well as a new, and hopefully correct, invoice are sent in parallel.

The first possible solution alternative is created using the constructs `fork` and `join`. All branches between a `fork` and a `join` construct are executed in parallel. According to the `ebBP` specification the `fork` constructs do not have `AND` semantics. These constructs can either have `OR` semantics where either zero, one or n branches will be executed, or `XOR` semantics where exactly one branch will be executed. However, it is possible to emulate an `AND fork` construct by using a `join` construct with `ALL` semantics together with an `OR` fork. In that case, every branch must be executed. Hence, the technical possibility for modeling the parallel execution with forks and joins is given. But this solution has a major drawback. If one branch fails during the execution there is no construct, attribute or anything else to describe what to do next in that case and how to handle the failure. It is not possible to have a rollback mechanism if one branch fails because no ebBP construct is available to describe the recovery solution. In the ebBP standard only the execution of a `BTA` construct can be seen as an atomic unit of work with transactional semantics but if the branches contain more than exactly one `BTA` construct, which is the case in all of the problem occurrences, the mentioned drawback still exists. The only solution then would be to introduce new ebBP constructs. But since changing the standard is not the aim of this work and is better discussed with the ebBP standard team of OASIS, this alternative is marked as the least favorable one.

The second idea is to use only an `OR fork` construct and no `join` construct. The main problem of this alternative is the necessity of the process to be in one final state in the end. This is not possible without a join construct and is also not allowed by the standard. As mentioned before, changing the standard is not considered as an option here.

The third solution is to have the tasks executed consecutively instead of in parallel. Thus a simple conversion is used to transform the parallel flow into a serialized one. Thereby the performance loss in executing the process is outperformed by the advantages of a clear and simple process model of the profiles. The sequence is defined as follows: First, the old invoice will be reseted by a credit note. Afterwards, the original invoice is sent and the process starts again with the new invoice that is now checked whether it is correct or contains any errors. The main thought behind this sequence is that the error has to be corrected before something new is issued.

To sum up, in order not to change the standard the best alternative is the third one. Reducing the complexity of a parallel process to a simple sequence as well as not changing the standard is rated higher than the performance decline that comes with that solution.

3.1.2.2 Loop Handling

The loop handling problem occurs in the NES Profiles 5 to 8. After receiving an invoice and the corresponding application response expressing that the invoice is containing errors in the process flow, the process will loop until the transaction completes successfully, in other words, until it ends in a specific state. As an example, assume an undercharged invoice is sent. This is reported back to the sender of the incorrect invoice with an application response and as a result an additional invoice is sent to request the difference between the real price and the undercharged price. However, this invoice can also be undercharged. This creates a loop that only terminates if the side issuing the invoices sends a correct invoice. There are three cases according to the NES profiles that need to be addressed, namely the `undercharge` branch, described in the example above, which sends a compensating invoice that requests the price difference between the real price and the undercharged price, the `overcharge` branch which sends a credit note that returns the amount that has been overcharged back to the customer and the `incorrect information` branch that nullifies the previous invoice with a credit note and sends a new invoice which is expected to then contain the correct information.

There are two common solutions to such a problem as described above. Either an iterative solution or a recursive solution are possible. The iterative solution uses a loop like a `while..do`- or `do..until`-construct. The recursive solution solves such problems by invoking itself over an over again until it is terminated.

First, the iterative solution using a `while..do`-loop is analyzed. To create a loop in ebBP a `decision` construct is needed which contains a condition that leads to a `BTA` construct that is executed and returns the flow to the decision. The `decision` construct is the condition of the while..do loop, the `BTA` the body. This strategy can be applied to the branches of the `decision` construct that handles the `undercharge` and `overcharge` cases which only need one compensation message to correct the error. In case of an `undercharge` the `decision` directs the flow to the `BTA` that sends an invoice that corrects the error and is checked after execution by the `decision` again. This applies analogously to the `overcharge` case. Despite the fact that this solution handles two important cases, there is a huge problem in the `incorrect information` case when both the credit note nullifying the previous invoice and the new invoice are sent. In order to analyze this in detail, the possible solutions of the parallel execution problem, namely

the execution in parallel as well as in sequence, are discussed separately to respond to the differences of each alternative.

To begin with, the problem is examined with the assumption of having the ability to execute branches in parallel. If it is determined that the invoice contains `incorrect information`, two branches will be started. Each branch compensates a part of the error found and has its own decision that then analyzes if the sent message has been correct. However, if both compensating messages have not been correct and thus have contained `incorrect information`, each decision flow then starts two flows and so forth. As it can be seen, each loop step could create an additional process. It is possible to model this flow for a specific number of concurrent processes, however, it is not possible to model this flow for an arbitrary number of concurrent processes. As it is not known ex ante how many concurrent processes will be necessary for an execution of a profile, this does not solve the problem described.

Next, the alternative that executes `BTA`s in sequence is discussed. It is assumed that after sending a credit note and its corresponding application response a decision will check whether everything has been correct and if this is not the case this would lead to a compensating `BTA`. Because this compensating `BTA` can also have an error it is also checked and compensated and so forth. This may lead to a queue[2] of BTAs to be compensated subsequently. Hence, once the first compensation is successful the next BTA in the queue has to be compensated. Such behavior could be specified using a stack-like data structure. As ebBP does not offer such data structures out of the box, this solution would require ebBP extensions which are not considered to be a desirable goal for the work at hand. As can be seen above, it is irrelevant how the parallel execution problem that arises from the iterative alternative is tackled. Both possibilities fail in solving the loop handling problem. Consequently, the iterative alternative is not a solution to the loop handling problem.

Second, the recursive solution is analyzed. Instead of having all the constructs in one process they are separated in several `BC`s. Each `BC` only handles the sending of one document, and the compensation handling will be delegated to other processes. Note, that compensation control is not given completely away by the calling process (`BC`) as the called process returns the result of the compensation. In so far, the `BC` construct is used as a recursive function and it can call itself or other `BC`s for the compensation handling.
In the best case, no recursive invocation is needed because it is only used when an error occurs. However, if an error occurs, child processes will be started to handle the error. After the child process(es) has(have) finished, the process is in a specific state (success or failure) depending on the returned state of the child processes and a decision then determines whether to proceed or to abort and inform a parent process if one exists. An advantage of this method is that the depth that the recursion can go into is hidden from the caller. A child process itself can issue, e.g., an incorrect invoice which needs to be compensated. Therefore this will be done and only the aggregated state will be returned to the caller independently of how many child process instances were needed for the actual compensation. In conclusion, this solution is a good option because it solves the problem.

Third, a combination of the recursive and the iterative solution is possible. In that hybrid solution the iterative solution is used for the `undercharge` and `overcharge` branches and the

[2] or rather a stack

recursive solution for the `incorrect information`. Although this is possible, a holistic approach is preferred for its reduced complexity.

To sum up, the iterative solution is not usable, the hybrid solution is not holistic, and thus the recursive solution is used to solve the loop handling problem. With this solution also the flexibility increases because every iterative process description can be modeled as a recursion but not vice versa.

3.1.2.3 External Medium

The external medium problem occurs within NES profiles 5 and 6. In both profiles an external medium is used by the profile's *customer* party to inform the *supplier* party that an invoice or credit note that has been sent before contains an error. An external medium is, e.g., a fax machine, telephone, email, etc.. Via this external medium both parties inform each other about a particular error type and agree upon it, namely `incorrect information`, `undercharge` or `overcharge`. The external medium is not part of the ebBP process but it influences the flow/routing of the ebBP process. The ebBP process does not control the external medium and therefore is not able to retrieve information about what failure is agreed upon. Hence, state alignment has to be achieved in a different way. The following solutions all start out with having the customer and supplier agree on the error type via the external medium.

The first solution is to let the supplier send a business document that reflects the error type, i.e., additional invoice, credit note, or compensation credit note and new invoice. This solution is probably closest to the original NES specification. As agreement via external medium is not represented in ebBP, the ebBP process specification for this solution has to arrange for the exchange of any of the business document types described above. This can be addressed by using a `XOR` fork with a join or an additional message type that allows for transmitting any of the above types. In case of the `XOR fork` alternative, several additional constructs would be needed such as a corresponding `join`. Defining the additional message type would mainly require the definition of an `xsd :choice` like structure. In both cases, some metadata for determining in which way the transmitted business document is to be interpreted would have to be defined. For example, a credit note could either replace or complement an erroneous invoice, i.e., the transmitted business document type does not unambiguously identify the error type the integration partners have agreed upon via the external medium.

The second solution arranges for an additional control message (in the form of an ebBP Notification) sent by the supplier that communicates the error type agreed upon via the external medium. Subsequently, an according invoice or credit note or both are sent by the supplier. In this case, the meaning of these business documents is clear because the error type has been transmitted before.

The third solution is identical with the second one except for the fact that the customer sends the additional control message instead of the supplier.

In summary, the main difference between the first solution and the other solutions is that the metadata necessary for explicitly capturing the control flow in ebBP is not placed into

a separate ebBP Business Transaction. Technically speaking, any solution is possible, but separating control flow information from business content is considered to be preferable to reducing the number of messages to exchange. Finally, the second solution is selected because it lends itself better to the situation that the additional control message's error type differs from the agreement via external medium than solution three. If so, solution two offers the possibility for the customer to simply reject the subsequent invoice or credit note (again via external medium). On the contrary, solution three would force the supplier to send a business document that corresponds to the error type communicated via the control message exchanged before. In order to disagree with the control message's error type the supplier possibly would have to send the wrong type of business document or a business document with meaningless content which apparently is not desirable.

3.1.3 Modeling of the NES Profiles

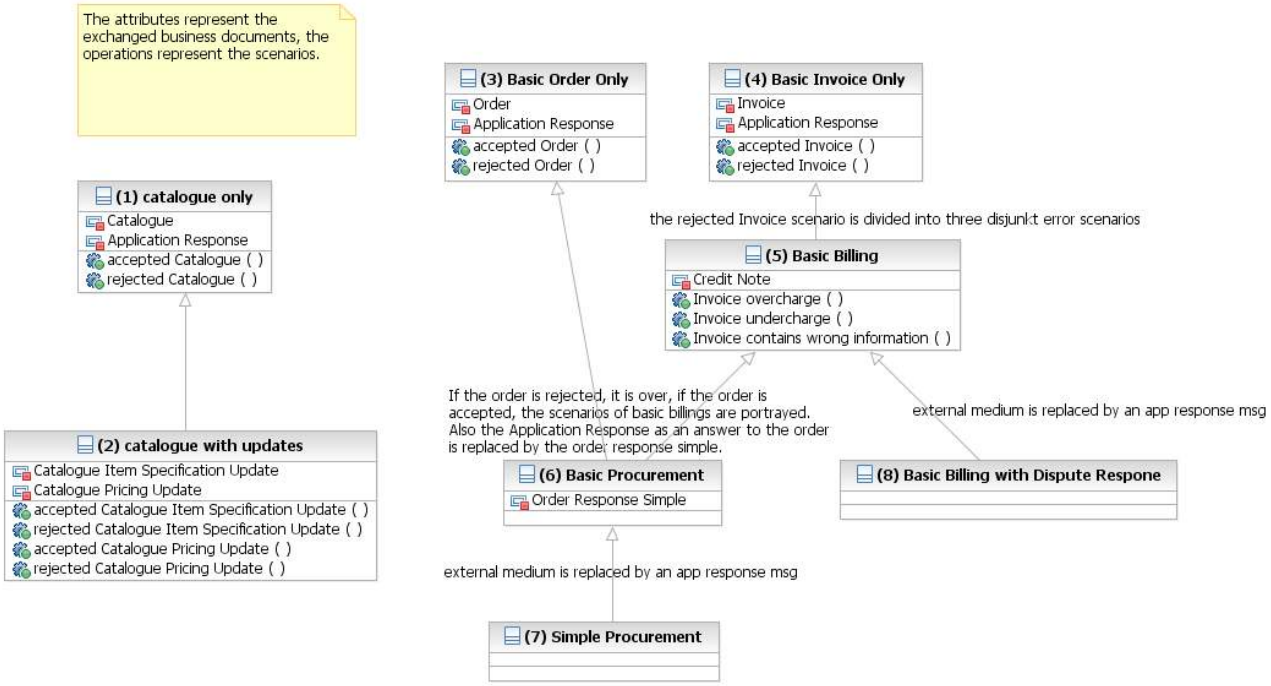


Figure 3.2: The dependencies among NES profiles visualized as UML class diagram

Figure 3.2 presents an overview over dependencies between the eight profiles of the NES. To visualize this a class diagram is used.

Profile 1 (Catalogue Only) describes a process of sending catalogues or extensions to catalogues. Profile 2(Catalogue with Updates) extends Profile 1 with pricing or item update information.

Profile 3 (Basic Order Only) represents a simple order process. Corresponding to that, Profile 4 (Basic Invoice Only) describes a basic process of sending invoices. Profile 5(Basic Billing) extends Profile 4 and offers the possibility to justify the rejection of an invoice. Reasons for rejection are transmitted via an external medium. This external medium is replaced by an application response message in Profile 8 (Basic Billing with Dispute Response). Profile 6 (Basic Procurement) combines Profiles 3 and 5 to a process of procurement. Corresponding to Profile 8, Profile 7 (Simple Procurement) automates the transmission of the reasons for rejection of invoices of Profile 6 (for more details see [Gro07b]).

3.1.3.1 Basics of Modeling the NES Profiles

In this section, some basics of modeling the NES profiles which are common to all profiles are explained and justified. Afterwards, the setting of the Quality of Service features which are common to all profiles are defined. Note that the translator to be introduced in section 4.3 can also process differing QoS settings.

Table 3.1 shows the setting of QoS features. At the level of document envelope, all attributes are set to transient. In general, these features depend on the agreement between the integration partners. For this work, security at the network layer is considered to be sufficient because all NES profiles only allow for two integration partners.

`isGuaranteedDeliveryRequired` is always set to true because in the context of doing business the guaranteed transmission of business documents is always necessary, even if they do not have legally binding content.

`isConcurrent` is always set to true because there are no reasons to prevent parallel execution of Business Transaction Activities in this work.

The other features of table 3.1 are always set to true because regarding their meaning (see chapter 3.2), doing serious e-business without them is not imaginable and can have serious consequences to integration partners.

Level	**QoS feature**	**Setting**
Business Transaction Activity	isConcurrent	true
Business Transaction	isGuaranteedDeliveryRequired	true
Requesting/Responding Business Activity	isAuthorizationRequired	true
	isIntelligibleCheckRequired	true
	retryCount	3
Document Envelope	isAuthenticated	transient
	isTamperDetectable	transient
	isConfidential	transient

Table 3.1: Setting of QoS features

Some QoS attributes can not be set in all types of Business Transactions. Table 3.2 shows the attributes set in addition to the others within `CommercialTransaction` and `Notification` at the level of the Requesting or Responding Business Activity. For transmissions with legally binding content it is necessary that the business partners can not deny the sending or receiving of messages. For this purpose, the attributes of table 3.2 are all set to true.

QoS feature	**Setting**
isNonRepudiationRequired	true
isNonRepudiationReceiptRequired	true

Table 3.2: Setting of QoS features depending on the type of Business Transaction

The definition of Business Signals is also common to all profiles. Each profile defines ex-

actly one `ReceiptAcknowledgement`, one `ReceiptAcknowledgementException`, one `AcceptanceAcknowledgement`, one `AcceptanceAcknowledgementException` and one `GeneralException`. Each Business Transaction reuses these definitions.

Each Business Transaction has two roles. This project distinguishes between the role of Initiator and the role of Responder of the Business Transaction.

Each Business Collaboration also differentiates two roles. These roles are *Supplier* and *Customer* according to the NES profiles.

Each Business Collaboration has three Final States. The Final State `techFail` states that a protocol failure occurred. The Final State `success` expresses both, a protocol and a business success. The Final State `businessFail` states a successful termination of protocol, but a non-successful termination of business.

Each Decision within this use case checks for the condition guard value `AnyProtocolFailure`. If this is true, an error has occurred at one party during the flow of the protocol. For this reason, the process can not finish successfully and the process ends with the final state `techFail`. Each decision within the models of the profiles has such a branch, and thus it is not described each time again.

3.1.3.2 Structure of the NES Profile Description

The following listing shows the structure of profile description, given by the NES. These informations are used as input to the modeling process.

- profile id
- context: the usage context of the profile;
- summary: summary of subjects, exclusions, requirements and aims of the profile;
- description: textual description of the profile;
- profile scenarios: listing of the scenarios;
- business requirements: requirements for business and documents;
- business benefits: benefits which can be gained if this profile is used;
- use case diagram: use case diagram of the profile;
- actors involved in the process;
- process parameters: business rules, exceptions and pre-conditions;
- activity diagram: profile flow as an activity diagram;
- activity description: textual description of the activities;

- scenario descriptions: description of the different scenarios;

Each of the following sections that describe the modeling of the respective NES profiles will give a short summary of the informations listed above.

Business Documents which are used within the profiles can have the same name, but they can differ in the elements provided by the XSD. The XSDs can be found the homepage of the NES[3].

3.1.3.3 Profile 1: Catalogue Only

This profile describes a process of sending a catalogue or catalogue extensions from a supplier to a customer. At the side of the customer, a decision for accepting or rejecting the catalogue is made. Depending on the outcome of this decision, the supplier sends an Application Response with positive or negative content. Therefor, the scenarios *reject catalogue* and *accept catalogue* are distinguished. Catalogue and Application Response are the business documents used in the profile. (For more details see [Nor07].)

Image 3.3 shows the ebBP model of NES profile 1, visualized as a UML activity diagram. The first step in creating an ebBP model for NES profile 1 is the definition of Business Documents. The definition references `UBL-Catalogue-2.0.xsd` and `UBL-ApplicationResponse-2.0.xsd`.

The next step is to model two Business Transactions, one for sending the catalogue and one for sending the Application Response. Both are of type `InformationDistribution`. This is because of the non-legally binding business character of a catalogue and of the confirmation of the catalogue. Each Business Transaction sends the document in the Document Envelope of its Requesting Business Activity and contains an empty Responding Business Activity.

The flow of the Business Collaboration starts with a Business Transaction Activity which is responsible for sending the Catalogue from Supplier to Customer. For this, the Supplier of the Business Collaboration is associated with the Initiator of the referenced Business Transaction. The Customer is associated analogously. The Time To Perform is defined as one hour. Because a catalogue is for information only, the attribute `hasLegalIntent` is set to false.

After sending the catalogue a decision checks for a protocol failure. If the `ConditionGuardValue` is `AnyProtocolFailure`, the flow ends with a technical failure. Protocol failure means that an error occurs during the performance of the Business Transaction Activity.

If there is no protocol failure, the Business Transaction handling the Application Response is used to send the acceptance or rejection of the catalogue back to the Supplier. For this purpose, a Business Transaction Activity that associates the Customer with the Initiator role is used. The TimeToPerform value is set to one hour.

A final decision checks for protocol failures. If a failure occurs, the choreography ends again with technical failure. If no error occurs, the decision uses an XPATH2 expression to check the Application Response for acceptance or rejection of the catalogue. Depending on this decision,

[3]http://www.nesubl.eu/documents/nesvalidationtools.4.6f60681109102909b80002641.html

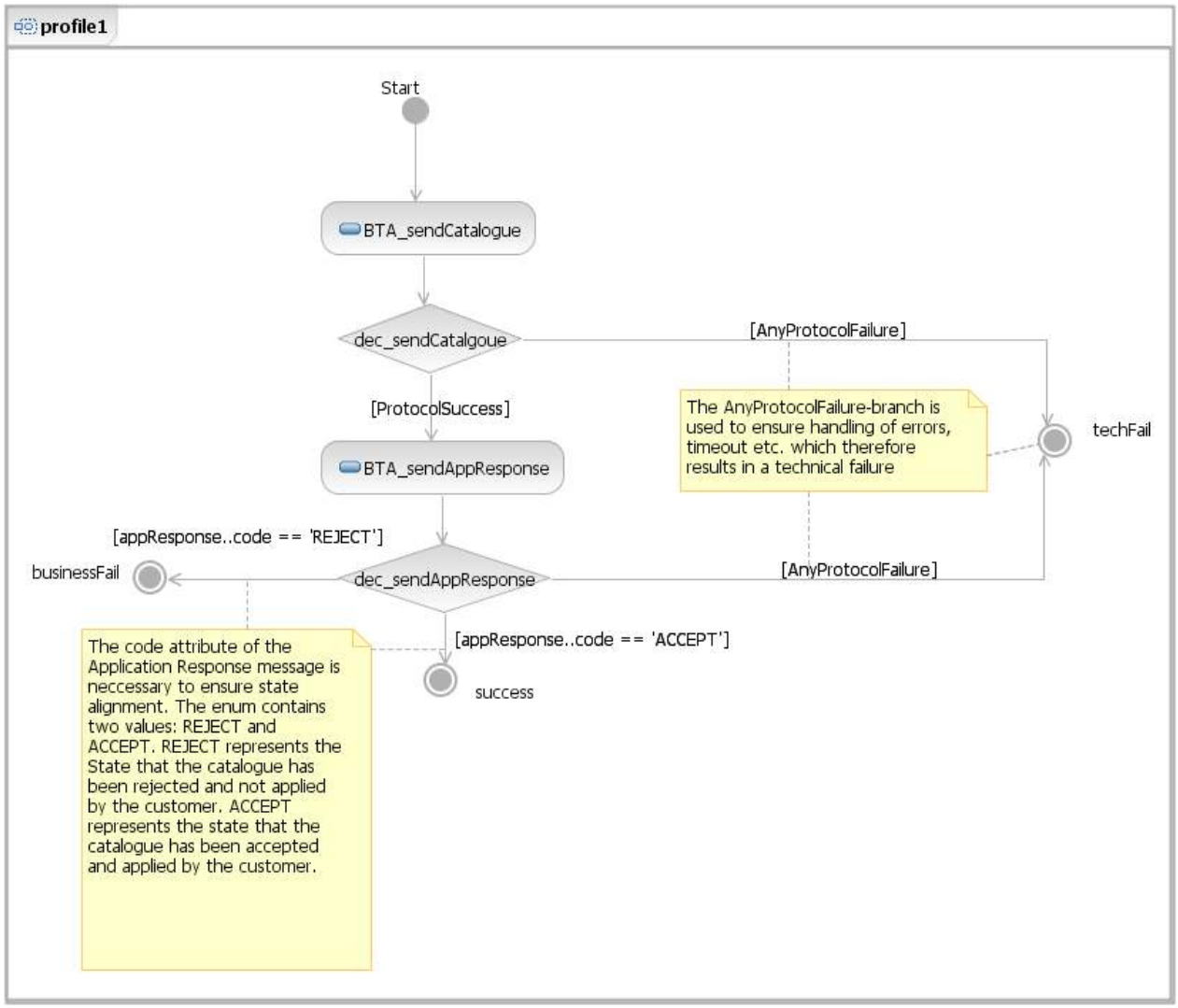


Figure 3.3: The choreography of NES profile 1 as UML activity diagram

the flow ends with successful termination (business success) if the catalogue is accepted. If not, the flow ends with `businessFail`. The last decision of this profile represents the implementation of the two scenarios of Profile 1. Listing 3.8 gives an example of an XPATH2 condition expression used in Profile 1.

```
<ConditionExpression
        expressionLanguage="XPath2"
        expression="normalize-space(string(//DocumentResponse/Response/ResponseCode))='
            ACCEPT'">
</ConditionExpression>
```

Listing 3.8: Example XPATH2 definition within a decision

The fork of the activity diagram, as described in the NES profile, is not considered because the application of the catalogue is related to the backend of the Customer and doesn't matter for business integration. Also the handling of a catalogue rejection or acceptance at the side of the Supplier is not relevant for business integration and concerns only the Supplier's backend system.

3.1.3.4 Profile 2: Catalogue with Updates

Profile 2 represents an extension to Profile 1. It can not only send catalogues and catalogue extensions, but it distinguishes between sending catalogues, catalogue item updates and pricing updates. For this purpose, the profile uses the documents *Catalogue*, *Catalogue Item Specification Update*, *Catalogue Pricing Update* and *Application Response*. For each of the first three document types the profile defines two scenarios, one for accepting and one for rejecting a particular document. It is not determined which document is sent. After transmission of the document, the course of the profile is equal to profile 1. The document is accepted or rejected at Customer's side and after sending the decision as an Application Response the document is applied (for more details see [Nor07]).

The description of the modeling concentrates on differences to Profile 1. The problem of this profile is how to handle the undetermined document type that is to be sent from Supplier to Customer. Using the XOR Fork element of ebBP would mean modeling a whole flow for each possible document. Furthermore, for state alignment between both partners and for routing within the process description, it is necessary to know which document is sent or received next. Therefore, an XSD is designed which can contain each of the three possible catalogue elements. This is realized using an `xsd:choice` element (see document `UBL-Custom-CombiCatalogue.xsd` of the project sources). Listing 3.9 shows the realization of the choice. It says that a `CombiCatalogue` can contain either a `Catalogue` or a `CatalogueItemSpecificationUpdate` or a `CataloguePricingUpdate`. Its content can then be handled by the backend systems.

```
<xsd:complexType name="CombiCatalogueType">
   <xsd:choice>
               <xsd:element ref="cat:Catalogue"></xsd:element>
               <xsd:element ref="cisu:CatalogueItemSpecificationUpdate">
               </xsd:element>
               <xsd:element ref="cpu:CataloguePricingUpdate">
               </xsd:element>
   </xsd:choice>
</xsd:complexType>
```

Listing 3.9: Realization of CombiCatalogue

The usage of the CombiCatalogue document type replaces the undetermined sending of one of the three possible catalogue extensions. Thus, the number of six scenarios reduces to a number of two which are realized by the final decision of the ebBP model which decides whether the `CombiCatalogue` is accepted or not.

Because of reducing the number of scenarios profile 2 can be modeled equally to profile 1. This means that the `CombiCatalogue` replaces the `Catalogue` in the Business Document definition of Profile 1. Except for names, the rest of the ebBP file equals the model of profile 1.

3.1.3.5 Profile 3: Basic Order Only

Profile 3 describes a simple order process. This means a Customer sends an order to a Supplier who decides to accept or reject the order. Depending on this decision, he delivers the goods and requests payment or tries to resolve the rejection of the order externally. Figure 3.4 shows

the NES profile's activity digram, illustrating this process. This profile uses the document type Order(Basic) and distinguishes between the two scenarios of accepting and rejecting the order. (For more details see [Nor07].)

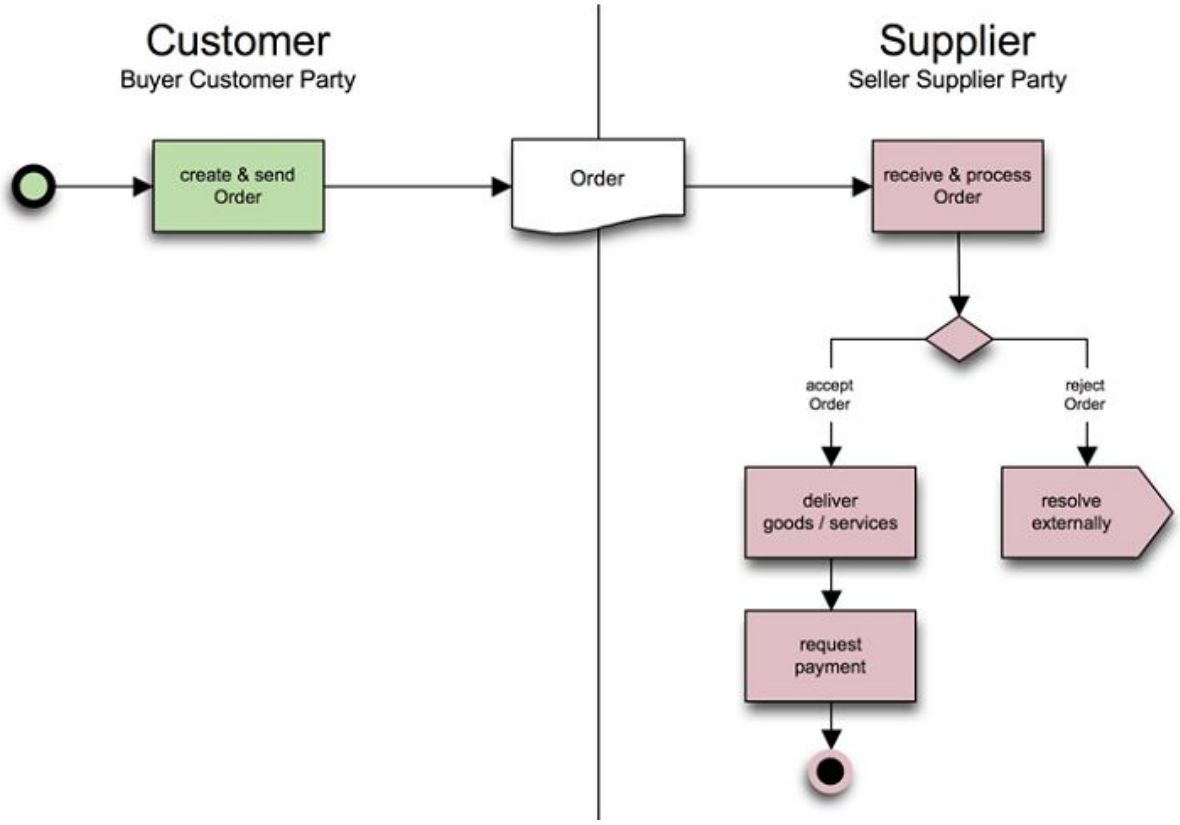


Figure 3.4: The NES UML activity diagram of profile 3

For state alignment between the two partners as well as routing in the choreography and later in orchestrations it is necessary to send the state of the order back to Customer. For this purpose, an additional Application Response is introduced. It is sent from Supplier to Customer after the Supplier has decided to accept or reject the order.

The ebBP model of Profile 3 uses the document type definitions `UBL-ApplicationResponse-2.0.xsd` and `UBL-Order-2.0.xsd`. Because of the legal intent of an order, a `Commercial-Transaction` is chosen as Business Transaction type and the Order is sent within its Requesting Business Activity. In its Responding Business Activity two Document envelopes are defined, one for positive and one for negative response to the order. In both cases the Application Response is referenced and the ebBP `isPositiveResponse` attribute is used distinguish between positive and negative responses. Note that, in this work, the isPositiveResponse attribute is transmitted as part of the message container at runtime as well, although this does not exactly match the ebBP semantics.

Figure 3.5 shows the flow of the Business Collaboration of Profile 3. It has only one Business Transaction Activity which is responsible for the execution of the Business Transaction described above.

The attribute `hasLegalIntent` is set to true because of the legally binding character of an

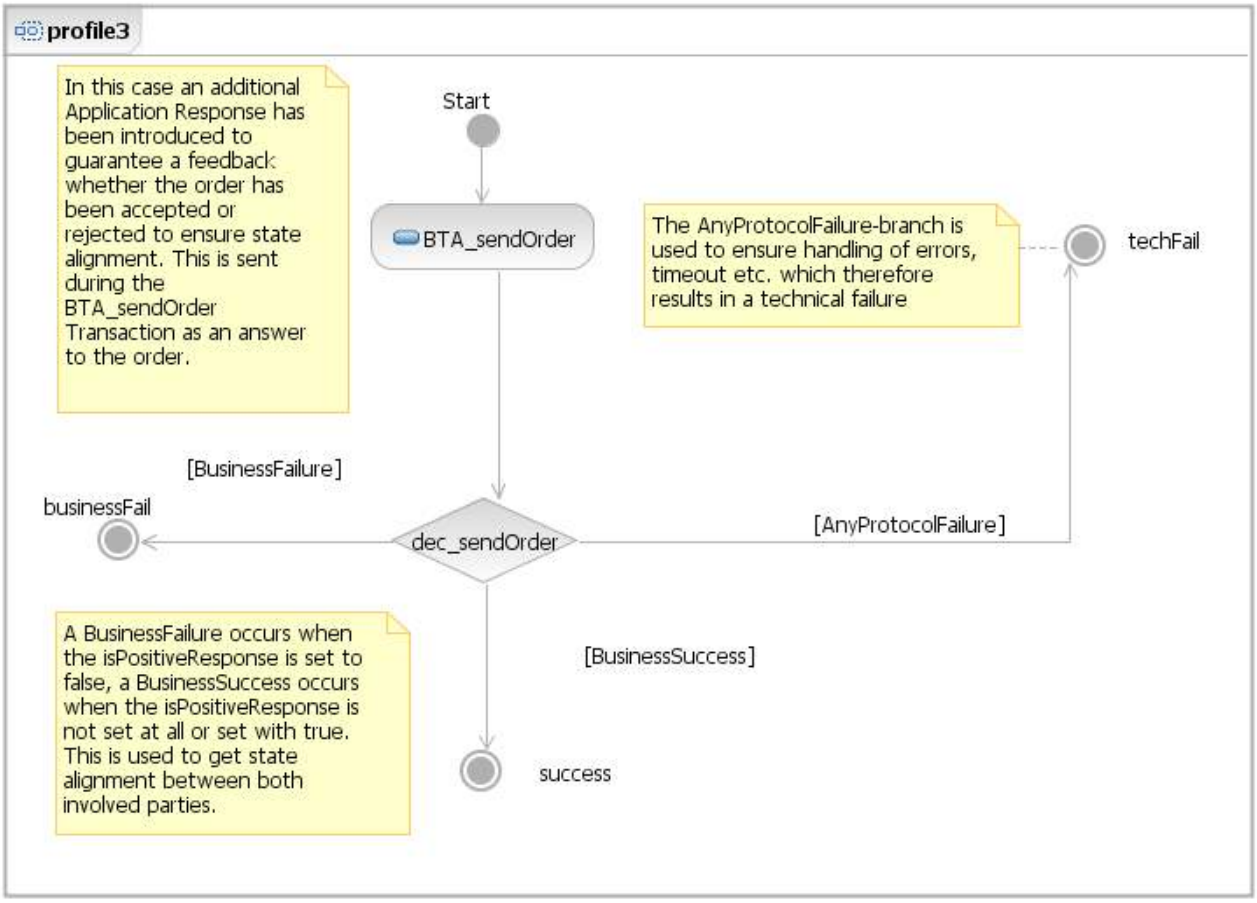


Figure 3.5: The choreography of NES profile 3 as UML activity diagram

order.

The Supplier of the Business Collaboration is associated with the Responder of the Business Transaction and the Customer is associated with the Initiator. The estimated TimeToPerform value is set to one hour.

As already pointed out, the referenced Business Transaction sends an Application Response back to the requester. Therefore, after the BTA, a Decision references the AppResponse in order to distinguish between the possible results of the BTA. If there is no protocol failure and thus the process does not end in state `techfail` the Decision checks the predefined condition guard values `BusinessFailure` and `BusinessSuccess`. In this work, these guard values are determined by checking whether the Application Response is sent within a Document Envelope that has `isPositiveResponse=true` or `isPositiveResponse=false`. Depending on this Decision, the control flow steps into Final State `success` or `businessFail`. This Decision implements the alternative scenarios of this profile.

Delivery of goods, payment requests and external handling of rejected orders as described in the NES profile are not relevant for integrating the processes of the business partners and thus are not considered in this model.

3.1.3.6 Profile 4: Basic Invoice Only

This profile is similar to Profile 3. In this process, the Supplier sends an Invoice to Customer who can accept the invoice and pay or reject the invoice and use an external medium to signal the wrong invoice. This process has two scenarios as well, the rejection and the acceptance of the invoice (for more details see [Nor07]).

For state alignment between the processes of the business partners, an additional Application Response to message the state of the acceptance of the invoice is introduced.

The Business Documents of the ebBP model are referencing `UBL-Invoice-2.0.xsd` and `UBL-ApplicationResponse-2.0.xsd`. A Commercial Transaction is used to transmit the invoice in its Requesting Business Activity and to transmit the answer as an Application Response in its Responding Business Activity. The Responding Business Activity uses two alternative Document envelopes which differ in use of the attribute `isPositiveResponse` to state whether an Invoice is accepted. The other reason for using a Commercial Transaction is the legally binding character of an invoice.

Because of this, the Business Transaction Activity which reuses the Commercial Transaction has set the attribute `hasLegalIntent` to true. This activity associates the Supplier of the Business Collaboration with the Initiator of the Business Transaction and the Customer with the Responder. After sending the Invoice and answering the state at the Customer by use of the Application Response, a Decision implements the different flows of control of the scenarios of the profile. If no protocol failure occurs, the decision examines the state of the attribute `isPositiveResponse` by using the condition guard values `BusinessSuccess` and `BusinessFailure`. The outcome of this Decision determines the corresponding Final State.

A visualization of the choreography of this profile is similar to the activity diagram shown in figure 3.5. Of course, the names are different in so far as “order” has to be replaced with “invoice”.

3.1.3.7 Profile 5: Basic Billing

Profile 5 is an extension of Profile 4. It extends Profile 4 by splitting up the scenario of a rejected invoice. For this purpose, the process specifies the scenarios of *overcharged invoice, undercharged invoice* and *invoice with incorrect informations*. The scenario of an accepted invoice rests untouched.

If an invoice is rejected, an external medium is used by the Customer to inform the supplier. If the invoice is undercharged, the Supplier sends a new invoice about the remaining amount. If an invoice is overcharged, the amount will be balanced by sending an credit note to the customer. In case of an invoice with incorrect informations, the Supplier sends a credit note zero balancing the incorrect invoice. Furthermore, the Supplier sends a correct invoice in parallel. This is an iterative process until all amounts are balanced because credit notes or invoices for balancing can be wrong too. This means that the number of steps is not specified before starting the process. (For more details see [Nor07].)

This profile uses the documents Invoice (basic), Credit Note (basic) and, because of the state alignment problem already described for Profile 4, an Application Response.

To handle this iterative process a recursive implementation approach is chosen. Recursive means (binary) ebBP Inner Business Collaborations for sending credit notes and for sending invoices are defined.

The parallel execution of sending credit note and sending invoice in case of the incorrect information scenario is serialized. For a detailed justification see sections 3.1.2.1 and 3.1.2.2.

The modeling of the inner Business Collaboration for sending invoice and credit note and their justification are explained in section 3.1.3.11. These sections also explain the chosen Business Transactions.

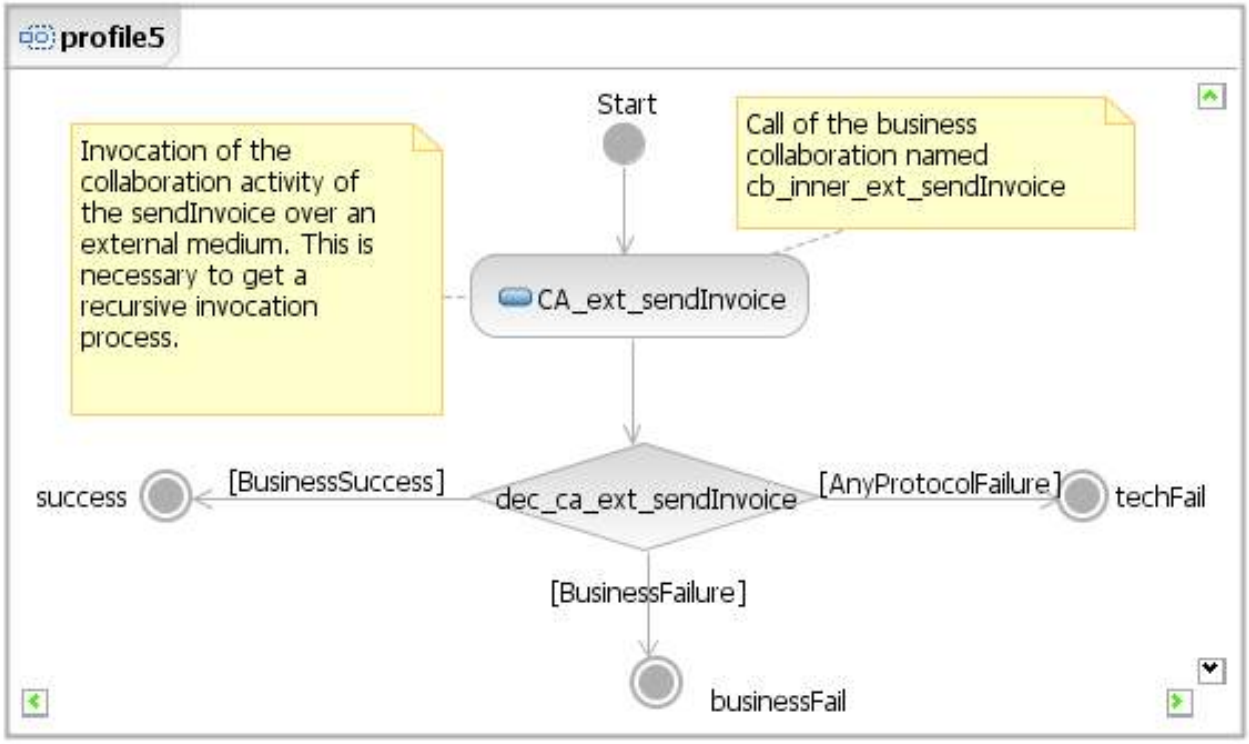


Figure 3.6: The choreography of NES profile 5 as UML activity diagram

Figure 3.6 shows the general flow of the choreography of Profile 5. For sending the initial invoice, the corresponding inner Business Collaboration is used via the Collaboration Activity `CA_ext_sendInvoice`. This activity is responsible for associating the roles of the calling Business Collaboration with the roles of the called one. Supplier and Customer of the outer Business Collaboration are associated with the corresponding roles of the inner one and the recursive process starts. The Decision after the Collaboration Activity depends on the outcome of the recursive `CA_ext_sendInvoice` process, which determines the Final State of the whole process. The Final States are `success`,`businessFail` and `techFail`.

3.1.3.8 Profile 6: Basic Procurement

This profile combines profiles three and five to a process of procurement. At first, the Customer sends an order to the Supplier who checks the order and accepts or rejects it. She informs the Customer of her decision using an Application Response. Depending on this decision, the Customer waits for the delivery of the ordered goods or cancels the order. In case of an accepted order, the Supplier goes to the next step of sending an invoice and starts the iterative process as described in section 3.1.3.7. This means the invoice can be accepted by the Customer, who then arranges the payment, or the Customer rejects the invoice and uses an external medium to justify his rejection. Three cases can occur in case of a rejected order: undercharge, overcharge and incorrect information. The handling of these cases is explained in section 3.1.3.7.

This profile distinguishes between the following scenarios:

- accepted Order, accepted Invoice
- rejected Order
- accepted Order, Invoice overcharge
- accepted Order, Invoice undercharge
- accepted Order, Invoice contains wrong information

(For more details see [Nor07].)

Modeling this profile, at first, requires to model the order process. For this purpose, a Business Transaction of the type `CommercialTransaction` is chosen to send a Business Document that conforms to the XSD `UBL-Order-2.0.xsd` within the Requesting Business Activity. The Responding Business Activity distinguishes between two Document envelopes with different settings of the attribute `isPositiveResponse`. Both envelopes encapsulate an Application Response that conforms to `UBL-ApplicationResponse-2.0.xsd`. Reasons for the choice of a Commercial Transaction are the legal intent of the order and the possibility to alternate the `isPositiveResponse` attribute within the document envelopes within the Responding Business Activity.

Based on this attribute, the following Decision determines the routing using the condition guard values `BusinessSuccess` and `BusinessFailure`. In case of a business failure the process ends in final state `businessFail`. This is the implementation of scenario `rejected Order`.

The exit `BusinessSuccess` represents the remaining scenarios with an accepted order. In case of an accepted order, which means business success to the Decision, the handling of the invoice starts with the Collaboration Activity `CA_ext_sendInvoice`. This activity calls an inner Business Collaboration for sending invoices and handling incorrect invoices with regard to the external medium used for justifying the rejection. The iterative process of balancing the invoice is solved using recursion. A closer description of this process is made in section 3.1.3.11.

The final Decision depends on the outcome of the called inner Business Collaboration. The decision is made between condition guard values `AnyProtocolFailure`, `BusinessSuccess` and `BusinessFailure`. The exits of this Decision connect to the corresponding Final States.

The fork of the NES diagram, which starts a parallel execution of sending an Application Response and starting the invoice process in case of an accepted order by the Supplier is serialized in the ebBP model. An Application Response is sent before the subprocess of sending the invoice is started. This ensures state alignment among the partners. Each side knows of the Decision and can route accordingly to the invoice process if the order is accepted.

The flow is visualized in figure 3.7. It is not required to model backend activities of the parties

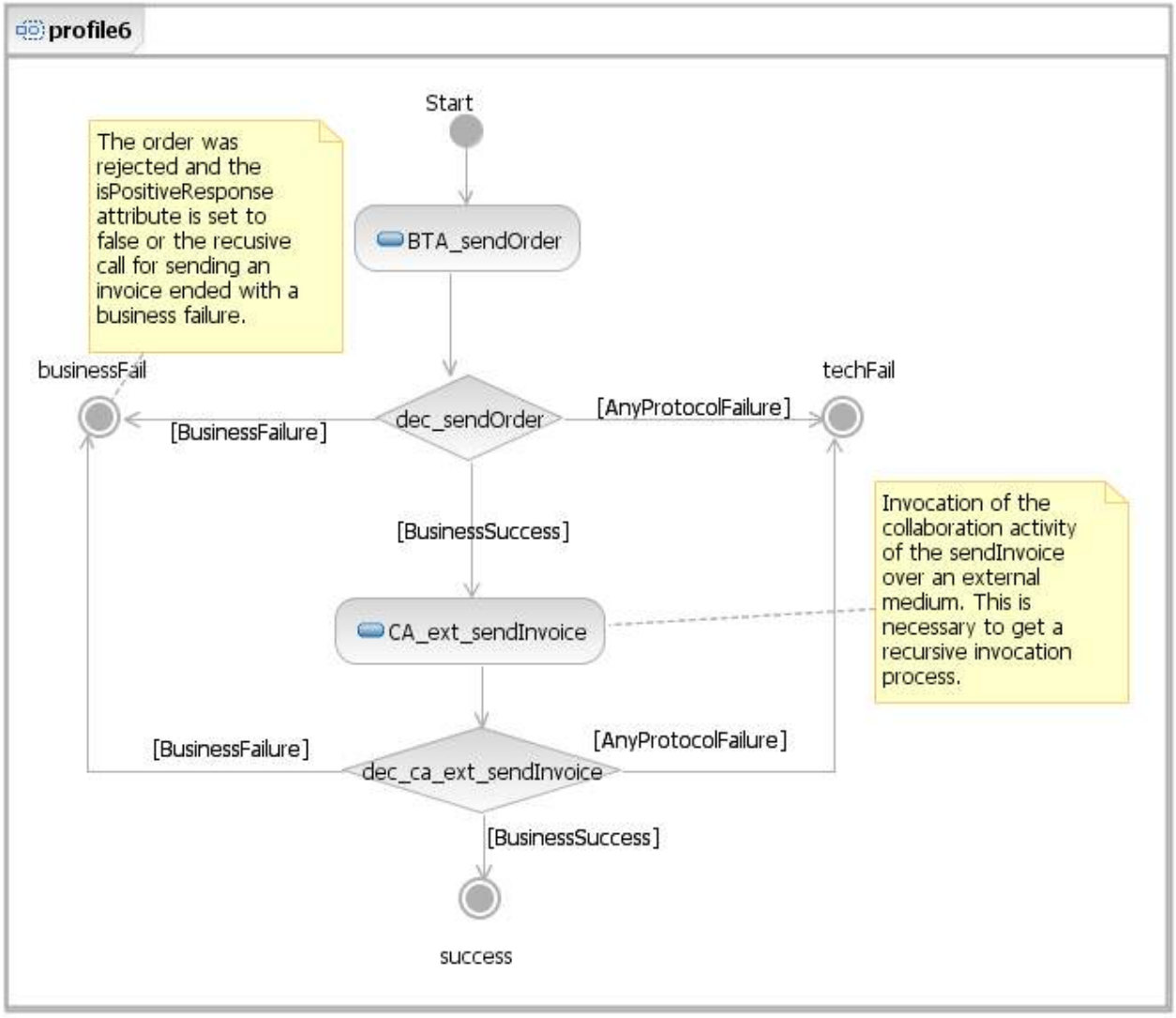


Figure 3.7: The choreography of NES profile 6 as UML activity diagram

for the purpose of creating an ebBP model. Therefore, waiting for delivery of goods or canceling the order at the customer's side are not considered.

3.1.3.9 Profile 7: Simple Procurement

Profile 7 is an extension of Profile 6 as much as Profile 8 is an extension of profile 5. Thus, the external medium to justify the rejection of the invoice is replaced by an additional Application Response. The handling of the incorrect invoice as well as the handling of the order is the same as in Profile 5 and described in section 3.1.3.7. The rest of the profile, including the different scenarios, is the same as in Profile 6 (for more details see [Nor07]).

Because of the replacement of the external medium, the Collaboration Activity `Ca_sendInvoice` now calls an inner Business Collaboration that uses an Application Response for notifying the Customer of the incorrect invoice. This Business Collaboration and its requirements are described in section 3.1.3.12.

The flow of the model is similar to the flow of profile 6, visualized in figure 3.7.

3.1.3.10 Profile 8: Basic Billing with Dispute Response

This profile is an extension to Profile 5. It differs from Profile 5 in informing the Supplier of an error in the invoice by means of a dedicated message. Instead of using an external medium, in this process, the customer sends an Application Response with the description of the incorrect invoice. The handling of the incorrect invoice is similar to the recursive process used by Profile 5 (or more details see [Nor07]).

The changed type of information transmission leads to a changed modeling within the inner Business Collaborations. These processes are explained in section 3.1.3.12. The outer Business Collaboration equals the flow of the collaboration of Profile 5. There is only the difference of calling the inner Business Collaboration for sending invoices without usage of an external medium.

This is done by the Collaboration Activity `Ca_sendInvoice` instead of `CA_ext_sendInvoice`.

For visualization please use the graph of Profile 5 (figure 3.6).

3.1.3.11 Business Collaboration sendInvoice and sendCreditNote with External Medium

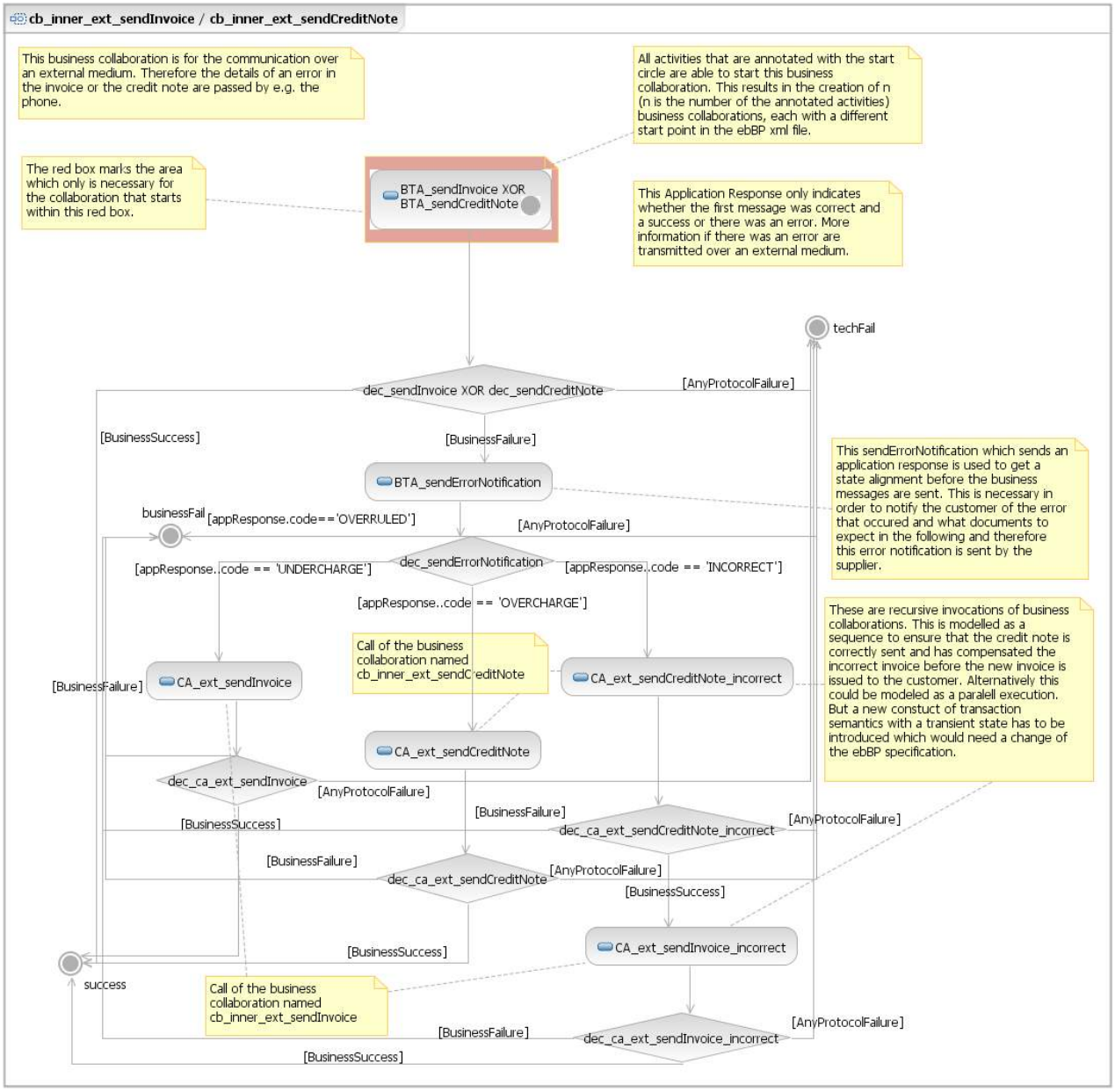


Figure 3.8: The Business Collaboration sendInvoice/sendCreditCote with external medium as a UML activity diagram

In UML diagram 3.8 there are two BCs modeled, namely the BC cb_ext_inner_sendInvoice and the BC cb_ext_inner_sendCreditNote. They only difference becomes manifest in the starting BTA and the following decision, the rest is the same. Thus, they were integrated into one diagram.

Both BCs are used in the NES Profiles 5 and 6 and make use of an external medium to agree or disagree on a specific error in case an error occurs.

To begin with, the BC which sends the invoice is focused. As can be seen in NES profiles 5 and 6, the supplier sends an invoice to the customer informing him about the costs of a previous order. The customer then checks and validates the invoice. Either the invoice is correct, and then its payment is arranged for, or it contains an error which results in informing the supplier about the error, in that case, over the external medium.

To model this, a Commercial Transaction is used to signal the legal intent of the invoice which is of type UBL-Invoice-2.0.xsd and sent from the supplier to the customer at the beginning. In addition, it requires a response message that confirms the success or failure of the transaction. Therefor, an application response of type UBL-ApplicationResponse-2.0.xsd is introduced to ensure that the customer can signal the supplier a success as well as an error. This is a necessary step to ensure state alignment. As analyzed and explained in section 3.1.2.3 the external medium should be used to inform the other partner about this error. However, the information that an error has occurred is sent via the above application response containing only a field that states that the previous invoice is not correct *somehow*. On both sides the application response is checked in the decision named dec_sendInvoice. If the isPositiveResponse attribute of application response is true, which signals a business success, the BC ends in the final state named success. In the other case, it then needs to be decided which error has occurred and how to handle it.

Next, the handling of incorrect invoices is described. When receiving the notification of an error the supplier determines which kind of error has happened. There are three different types, namely undercharge, overcharge and incorrect information. Every kind needs a different handling as stated in the activity diagram of the NES Profiles.

It is assumed that both the supplier and the customer have now agreed upon an error type or disagreed completely over the external medium. As the external medium is not part of the process an additional error notification message is introduced to synchronize the states of both sides as pointed out in section 3.1.2.3. This error notification is sent by the BTA with the name BTA_sendErrorNotification. Due to its nature as a notification the business transaction type notification is selected as well as the common data type UBL-ApplicationResponse-2.0.xsd to carry the error notification information. This information can either be one of the three error types described above in case the supplier and the customer agreed upon one of them or the type OVERRULED which states that, e.g., the customer thinks the invoice is not correct but the supplier is a firm believer of the correctness of the invoice. As disagreement is an option and cannot completely be avoided the OVERRULED scenario has been introduced to the ebBP process, although this is not modeled in the respective NES profiles.

The code contained by the error notification message is checked on both sides by the decision dec_sendErrorNotification. In case of the OVERRULED type the BC ends in the final state

named `businessFail`. The other cases will be described in the following by first analyzing the NES Profiles and then showing the modeling results in the ebBP process.

First, the `undercharge` case is explained. Since the customer has been charged less the supplier sends an additional invoice requesting the residue from the customer. When it is received from the customer, the check and validation process is executed upon this invoice as well as the original invoice. If then all received invoices for this choreography instance are correct the payment will be arranged for. However, if the second invoice contains an error the customer then notifies the supplier via the external medium as described above for the first invoice. This can be subsumed as a loop as analyzed in section 3.1.2.2. In reference to the loop handling problem in section 3.1.2.2 the error compensation is delegated to another `BC` instance. In this case the `CA` named `CA_ext_sendInvoice` is called by the `BC` and is assigned the task of handling the sending of the additional invoice etc.. Only the outcome of the `CA` is checked via the `decision dec_ca_ext_sendInvoice`. If the called `BC` has ended in a business failure state the caller process also ends in the `final state businessFail`. Analogous to the business failure the `final state success` is only reached if the called `BC` has ended in a business success state.

Second, the `overcharge` case is explained. Since the customer has been charged too much the supplier sends a credit note to balance the previous invoice. The invoice and the credit note are checked and validated at the customer side to ensure its correctness. If and only if the credit note is correct the payment will be arranged. If this is not the case the customer informs the supplier about the incorrect credit note via the external medium which results in the agreement on the error and so on and so forth. This is analogous to the error handling of the invoice with the only difference that the erroneous object is a credit note.

In the ebBP process the handling of this case is quite similar to the `undercharge` case. Instead of sending the calls to the `BC` named `cb_inner_ext_sendInvoice` the `cb_inner_ext_sendCreditNote` is invoked to send the credit note to balance the previous invoice. The other constructs behave the same way, just the naming of the following `decision` is changed to `dec_ca_ext_sendCreditNote`.

Third, the `incorrect information` case is explained. It can be seen as a combination of the undercharge and overcharge case because in order to correct the erroneous invoice a credit note as well as an invoice are sent. The credit note nullifies the previous invoice and is sent from the supplier to the customer who checks and validates the message. If and only if the credit note zero balances the erroneous invoice this part ends, in the other case the supplier will be informed of the error in the credit note and the handling of this error will be initiated. In parallel the new and hopefully correct invoice is sent to the customer starting the process with checks and validation again. As described by the parallel execution problem (see 3.1.2.1) there is the need to serialize the parallel execution. The approach is to correct the error before beginning to process the new invoice. So first the `CA CA_ext_sendCreditNote_incorrect` starts the `BC cb_inner_ext_sendCreditNote` to correct the previous invoice. After checking the outcome of the call with the `decision dec_ca_ext_sendCreditNote_incorrect` the process either ends in the `final state businessFail` if the result of the call is the business failure state or proceeds to call the `BC cb_inner_ext_sendInvoice`. A similar decision is placed right after the `CA CA_ext_sendInvoice_incorrect`, however, in the business success case the `final state`

`success` is reached.

As can be observed, the `CA`s in the `incorrect information` case have a postfix `_incorrect` to signal that this is not the same call as the ones in the other cases and the messages sent have other semantics.

Finally, the equivalent process that starts with sending a credit note is briefly described. Instead of sending an invoice a credit note of the type `UBL-CreditNote-2.0.xsd` is transmitted. However, the rest stays the same due to the fact that routing decisions are based upon the error code of the whole active process. The `undercharge` etc. refer to the process as a whole. At the beginning only an invoice is existent which is referred to but several other messages can be added to the instance document history, e.g., when an undercharged invoice has to be corrected by two additional invoices until the sum of the invoices is the correct balance.

3.1.3.12 Business Collaboration sendInvoice and sendCreditNote without External Medium

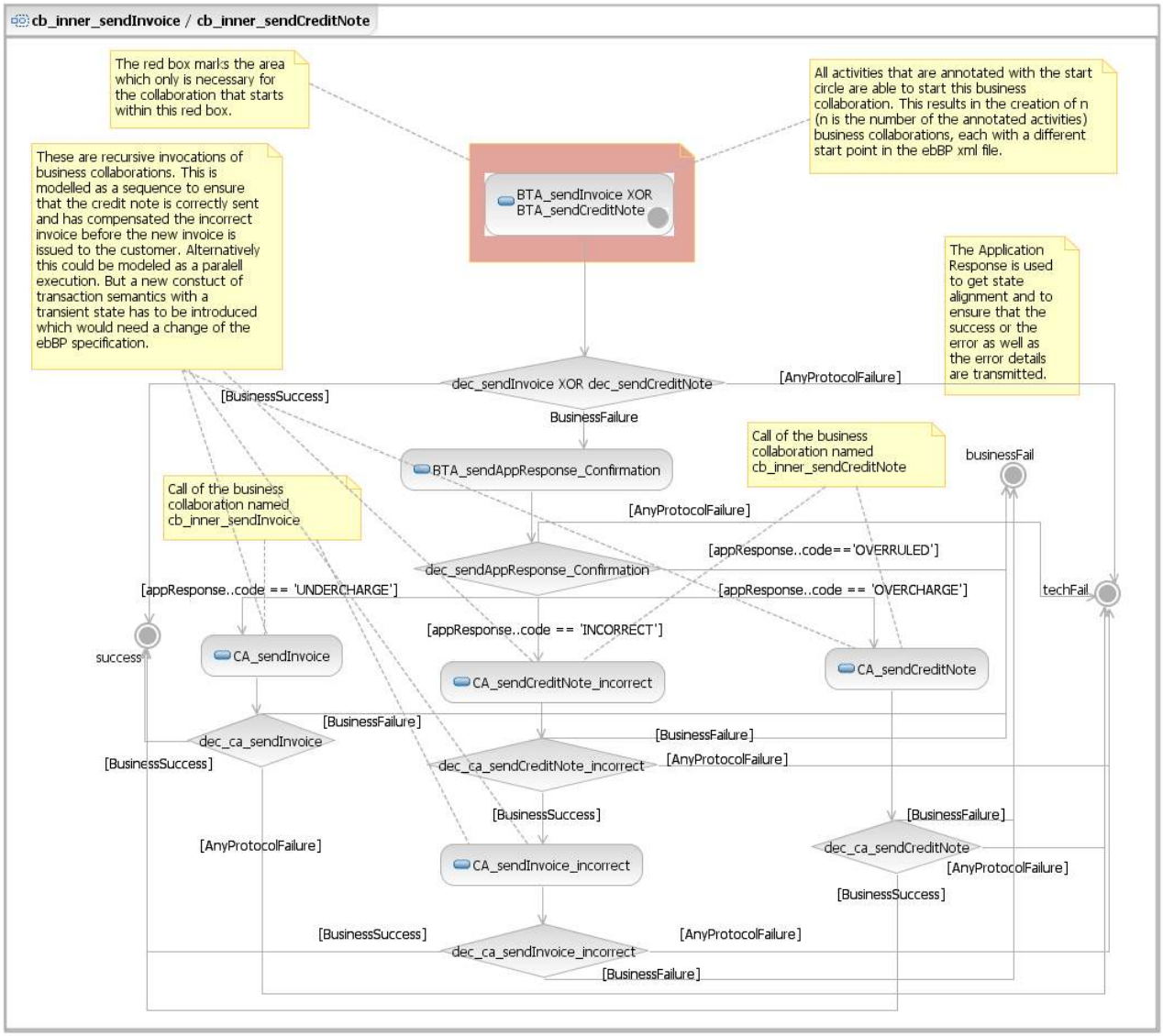


Figure 3.9: The Business Collaboration sendInvoice/sendCreditNote without external medium as a UML activity diagram

As can be observed easily, figures 3.8 and 3.9 only differ in few parts. Thus, only these parts will be described in detail, the other parts are analogously described in section 3.1.3.11.

In the UML diagram of figure 3.9 there are two BCs modeled, namely the BC cb_inner_sendInvoice and the BC cb_inner_sendCreditNote. They only differ in the starting BTA and the following decision, the rest is the same. Thus, they were integrated into one diagram.

Both BCs are used in NES Profiles 7 and 8 and instead of using an external medium as it is done in NES Profiles 5 and 6 an electronic message is exchanged.

So the commercial transaction sending the invoice or credit note contains an application response that contains a detailed failure description to inform the supplier about the error. The supplier then checks that description internally, and, if the supplier agrees on the failure and its type, the failure is echoed back to the customer to ensure state alignment between both parties. However, in case the supplier disagrees, e.g., the customer states the invoice is overcharged, but the supplier knows for sure it is not, the OVERRULED code is used to inform the customer that this transaction ended in a business failure. So instead of sending an error notification as described in section 3.1.3.11 an application response confirmation is sent which simply is an application response as described above. Note that a Business Transaction of type Notification is used to send the application response, although it is denoted BTA_sendAppResponse_Confirmation.

3.2 Evaluation of QoS Features

The analysis of QoS requires a definition of QoS. In many QoS related publications a definition is missing. This leads to an ambiguous understanding of QoS as can be seen by the definition used in [OMG08]: *"QoS can be defined as a set of perceivable characteristics expressed in user-friendly language with quantifiable parameters [...]."* Although *quantifiable parameters* are postulated, the same specification includes *security* in its *QoS Catalog*, but security qualities like confidentiality or encryption are hard to quantify. In [DLS05], a more general definition of QoS is used which is also adopted for the paper at hand: *"[...]the term QoS [...is] used to denote all non-functional aspects of a service which may be used by clients to judge service quality. This extends other more restrictive QoS definitions such as the common interpretation of QoS to mean network performance attributes."* In order to operationalize this notion of QoS for the work at hand, the set of QoS attributes being at least necessary for B2Bi has to be identified. Obviously, security and reliability are necessary attributes as business documents may be highly confidential and mission-critical, but a more thorough set of required QoS attributes identified by B2Bi experts can be found in the ebBP specification: *"The ebBP technical specification provides parameters that can be used to specify certain levels of security and reliability. This specification provides these parameters in general business terms"* ([YWM+06], sec. 3.5.7). Therefore, support for the QoS attributes identified in ebBP should be sufficient for B2Bi. These can be classified in attributes that relate to business documents and attachments (DocAtt), to so-called business activities used for specifying the transmission of a business document (BA), to so-called business transactions that associate one or two BAs for achieving state changes in the integration partners' systems (BT), to so-called business transaction activities which specify the actual execution of a BT (BTA), and to business collaborations (Coll) that specify the control flow between BTAs and Colls. Table 3.3 lists the QoS attributes identified in ebBP and the according level of specification. Most of these QoS attributes are self-explanatory and don't accept any parameters. Exceptions are the *DocAtt* attributes which are all related to security and the *timeToPerform* attribute. For the *DocAtt* attributes, ebBP provides the realization options *transient*, *persistent* and *transient-and-persistent* which means that the respective attribute has to be realized using security mechanisms at the transport level (*transient*), at the application level (*persistent*) or both. Finally, *timeToPerform* offers the parameters *design*, *configuration* and *runtime* for specifying whether timers shall be set statically (*design*), be agreed upon by partners before executing collaborations (*configuration*) or during collaborations (*runtime*). A discussion on how these QoS attributes have been realized here follows in section 4.1.2.

[4]also applicable for the ebBP control flow construct *Fork* which has not been used in this approach

QoS Attribute	Level of Specification
isAuthenticated	DocAtt
isConfidential	DocAtt
isTamperDetectable	DocAtt
isIntelligibleCheckRequired	BA
isNonRepudiationRequired	BA
isNonRepudiationReceiptRequired	BA
timeToAcknowledgeReceipt	BA
timeToAcknowledgeAcceptance	BA
isAuthorizationRequired	BA
retryCount	BA
isGuaranteedDeliveryRequired	BT
hasLegalIntent	BTA
isConcurrent	BTA
timeToPerform[4]	Coll/BTA

Table 3.3: ebBP QoS attributes and levels of specification

3.3 Platform Selection - GlassFish vs. Tomcat

From the various possible BPEL-Engines and WS-Stack implementations available today two open source implementations are chosen which will be analyzed below. The two candidates are: First, the GlassFish application server in combination with the Metro WS-Stack and the BPEL engine of openESB.
Second, the Apache ODE BPEL engine running on top of the Axis2 WS-Stack in the Apache Tomcat container. Table 3.4 shows a short overview over the two alternatives.

Platform	Platform A	Platform B
Application Server/Container	GlassFish	Tomcat
WS-Stack	Metro	Axis2
BPEL-Engine	openESB	Apache ODE

Table 3.4: Overview of the two platforms to analyze

3.3.1 GlassFish

Solution component	Name	Bundled as	URL
Application Server	GlassFish	GlassFish V2 UR2 Build 04 Release	`https://glassfish.dev.java.net`
WS-Stack	Metro	bundled with GlassFish	`https://metro.dev.java.net`
BPEL-Engine	openESB	engine part of openESB	`https://open-esb.dev.java.net`

Table 3.5: Platform A: Evaluated configuration

Test configuration:

3.3.2 Tomcat

Test configuration: This configuration was extended with various modules to enable QoS-Features - the used modules are listed in Table 3.8. Installation manuals can be found on the modules' homepages.

Solution component	Name	Bundled as	URL
Container	Tomcat	Apache Tomcat 6.0.18	`http://tomcat.apache.org`
WS-Stack	Axis2	bundled with Apache ODE	`http://ws.apache.org/axis2`
BPEL-Engine	Apache ODE	Apache ODE 1.2	`http://ode.apache.org`

Table 3.6: Platform B: Evaluated configuration

3.3.3 Derivation of a Feature Test Plan

3.3.3.1 Relevant Criteria for the Platform Selection

- QoS-Support (main requirement for this work (cf. section 1.2))
- Standard conformance
- Extensibility points
- IDE-Integration
- Usability criteria: installation difficulty, debugging possibilities, maintainability
- Performance

3.3.3.2 Feature Test Plan

IDE-Integration Today the usage of an integrated development environment (IDE) is essential to handle complex projects developed in teams. Therefore, the integration of the platform with full-fledged IDEs such as Eclipse, NetBeans or JDeveloper can hardly be dispensed with. Relevant aspects are: Integration of the Server (Starting, Stopping, Deployment), compatibility and executability of generated code, and support for server extensions.

Usability Another point that is partly linked with IDE-Integration is usability. Which tools exist to manage the configuration? How good and extensive are these tools? How easy is the installation and initial configuration of the platforms? How does the deployment work? This list could arbitrarily be extended.

Standard Conformance Standard Conformance is another aspect to decide which platform is better suited for this project. This is particularly important because support for B2Bi in this work shall be based on open standards as far as possible for interoperability and accessibility reasons. The focus of the conformance checks is put on the BPEL standard: Are all relevant constructs supported? What are the limitations of the different BPEL engines? Due to the necessity of QoS aspects to secure the Web service communication it is also important that as much WS-* standards as possible are supported.

Extensibility points If the platform does not provide enough support for needed features, e.g., if a working implementation of a certain standard is required, but not provided by the platform, it must be possible to extend the platform with new functionality. This can be done by writing extensions, modules or plug-ins.

Performance Performance tests actually require test cases that cover various sizes and various configurations. As such cases have not been available at the time of testing, very simple processes like receiving a message, assigning the values to a variable, using this variable when invoking another Web service and returning the result to the caller of the BPEL process have been used.

These BPEL processes have been load tested using the Web service testing tool SoapUI[5]. For example, the tool shows the average response time in a run of 100 Web service calls.

Functional QoS Tests WS-* standards, e.g., WS-ReliableMessaging or WS-Security, promise to enable QoS features in an interoperable way. According to the QoS requirements identified in section 3.2 the following WS-* standards have been identified to be of key importance for this project:

- WS-Security
- WS-ReliableMessaging
- WS-AtomicTransaction and WS-Coordination

Furthermore MTOM and FastInfoSet are tested as a way of increasing the performance by reducing the message size. To encrypt the communication on the network layer, SSL is tested.

Each of these standards is tested on both platforms. In every test case, first, the communication between a BPEL process and a Web service with one enabled QoS feature is tested, e.g., a BPEL process calling a Web service using reliable messaging. If this test is passed successfully the communication between two BPEL processes is tested, e.g., a BPEL process invokes another BPEL process using reliable messaging.

The tests of WS-Security are divided in sub-tests because there exists a number of alternatives for how to precisely realize encryption. For this project, “user name authentication with symmetric key” and “Transport security (SSL/TLS)” are tested.

Reliable messaging is tested with and without the option “exact order” which guarantees the delivery of messages in the order they have been sent.

[5]Free version available at `http://www.soapui.org/`; See web page for more information.

3.3.4 (Feature) Tests and Results

3.3.4.1 IDE-Integration

GlassFish + openESB (Platform A): When downloading the openESB suite with GlassFish it is also bundled with NetBeans. This is a first indicator that openESB as well as the included BPEL engine might be well integrated with NetBeans. Indeed it is very easy to use the combination of Platform A and NetBeans: The server can be administrated[6] automatically using NetBeans, e.g., the deployment of EJB modules can be managed directly via the NetBeans UI, while the usage of the GlassFish Admin Console is not needed very often. Moreover, BPEL projects can be built and deployed directly within NetBeans. The BPEL editor of NetBeans supports logging by using a BPEL extension which is supported by openESB. A particularly useful feature of NetBeans and Platform A is that running BPEL processes easily can be debugged. To do so, the process can be visualized using BPMN, breakpoints can be set and variables can be checked at runtime etc.

A handicap of Platform A is that a BPEL process has to be encapsulated within a JBI CompositeApplication. But this CompositeApplication can be generated very easily with NetBeans. Integration with other IDEs such as Eclipse was not investigated.

Tomcat + Apache ODE (Platform B): The components of Platform B are not bundled with a special IDE. Nevertheless Apache Tomcat can be integrated in NetBeans and Eclipse (plug-in necessary). But only simple administration tasks such as starting and stopping the server can be executed directly from within the IDEs. For most tasks the admin console has to be used. Sample processes were created with the BPEL editors of NetBeans and Eclipse and both could be deployed. Regrettably, the logging extension and debugging functions of NetBeans are not supported for Platform B.

Conclusion: Due the lack of logging and debugging possibilities defined in the BPEL standard the possibility to debug a running BPEL process directly within NetBeans is a strong reason to use Platform A. Another point is that GlassFish itself is perfectly integrated within NetBeans. In conclusion we can point out: The IDE-Integration is clearly better with Platform A.

3.3.4.2 Usability

Most of the usability aspects which should be tested already have been described in the previous section. Due to the tight NetBeans integration of Platform A, it has strong advantages in usability aspects. An advantage of Platform B is that no composite application is needed to deploy a BPEL process. A BPEL process simply can be copied into a special folder and then will be deployed automatically. The installation of the platform is rather easy with both platforms.

[6] At the time of writing, but this is unlikely to change

GlassFish can be installed with a (Windows) install routine. Libraries required for core QoS features are built-in and thus no extensions have to be installed. Apache ODE contains a preconfigured version of Apache Axis2, but it is not bundled with Tomcat. So Tomcat has to be installed first, and then ODE can be deployed. Unfortunately, the preconfigured Axis2 version does not include the necessary modules that support WS-* standards. The installation of these modules causes some problems because some dependencies are not clearly defined, e.g., a jar is missing and has to be copied from an "unbundled" Axis2 version.

Table 3.7 concludes the usability aspects.

	Platform A	Platform B
Deployment	Easiest way is to use NetBeans; Deployment by admin console is also possible; automated deployment with ant is complex; CompositeApplication is needed	Short deployment descriptor an copy-paste in the "processes" folder; BPEL process can be deployed without Composite Application
Admin-Interface	modern look-and-feel; very "powerful", functions partly hard to find	Only a few functions accessible by the interface, most configuration has to be done in config files
Difficulty of installation	Very easy (windows installer)	Installation of Tomcat, Axis and ODE easy - but problems with installing Axis2 modules (unclear dependencies)
Logging and Debugging	Log4j can be used; Logging definitions can directly be inserted in NetBeans-BPEL-Editor	Log4j is ODE's default logger; Logging statements directly in BPEL are not supported

Table 3.7: Comparison of some usability aspects

3.3.4.3 Standard Conformance

Instead of testing all defined BPEL elements in the standard it is decided to test the constructs needed for this project (see section 4.1.1). The producers of the two BPEL engines also provide some information about BPEL compliance[7].

Detailed functional tests of several more complex BPEL constructs are provided for Platform A in chapter 4.1.1.2.

Both platforms promise support for the most commonly used WS-* standards. The following table 3.8, which shows the supported standards, is a summary of various web pages of the manufacturers and independent sites[8].

[7]Apache ODE: `http://ode.apache.org/ws-bpel-20-specification-compliance.html`; openESB: `http://developers.sun.com/docs/javacaps/designing/bpelsecug.cnfg_bpel-se-language-const_r.html`

[8]Information about the different Apache Tomcat modules can be found on the project homepages: Apache Muse: `http://ws.apache.org/muse/`; Apache Rampart: `http://ws.apache.org/axis2/index.html`;

Standard	Platform A	Platform B
WS-Addressing	X	X (Core Module)
WS-Eventing		
WS-Notification		X (Apache Muse)
WS-ReliableMessaging	X	X (project Sandesha2 - support for Axis 2.1.3)
WS-Policy	X	X (Apache Rampart)
WS-Security Policy	X	X (Apache Rampart)
WS-Security	X	X (Apache Rampart)
WS-Trust	X	X (Apache Rampart)
WS-SecureConversation	X	X (Apache Rampart)
WS-Atomic Transaction	X	- Should be realized by Module "Kandula2" - but project is stuck
WS-Coordination	X	- Part of "Kandula2"
WS-Metadata Exchange	X	X (Apache Muse)
WSDL 1.1 Support	X	X
WSDL 2.0 Support		

Table 3.8: Comparison of supported WS-* standards. (X := respective standard supported)

3.3.4.4 Performance Tests and Results

Our performance tests indicate that the BPEL engine of Platform A is faster than the engine of Platform B. In the majority of our test cases Platform A is faster in responding to a message by a factor of 10. In one test case with a number of invokes in a while loop (10 loops) the BPEL engine of Platform B crashes.

Clearly, the results of such tests are not very useful to predict the performance in a "real" production scenario. Performance testing is not that easy when realistic test cases are missing. The real "performance" in our final tests of the whole system at the end of the project show clearly that the performance tests we did during the platform selection were far from realistic. For example, building and deploying a simple BPEL process can be done in 10 seconds, but building and deploying a BPEL process that implements a NES profile takes about 2 minutes (on the machines used during the project). The time differences in deploying and executing the simple test services as opposed to the NES profile services might indicate that Platform B outperforms Platform A for the NES profile services as well. Though, the opposite case is

Apache Sandesha2: `http://ws.apache.org/sandesha/sandesha2/`; Apache Kandula2: `http://ws.apache.org/kandula/2/`

at least as likely. Unfortunately, the performance of both platforms could not be evaluated a second time at the end of the project due to time constraints.

3.3.4.5 Functional QoS Feature Tests and Results

The goal of the following functional QoS-Tests is to test the ability of the platforms to have BPEL processes communicate with given QoS features. For example the QoS feature "isGuaranteedDeliveryRequired" should be realized by WS-ReliableMessaging. Apache Axis2 and GlassFish both claim that they support this standard. In our context it is necessary that the protocols support interaction between a Web service and a BPEL process as well as between two BPEL-processes.

As described above, the QoS attributes predefined in ebBP are to be realized using different standards. Therefore, the following standards are tested: WS-ReliableMessaging, WS-AtomicTransaction/WS-Coordination and WS-Security (User name authentication with symmetric key, Transport Layer Security). Furthermore MTOM and FastInfoSet have been tested as a way of performance enhancement. Also, simple SSL encryption has been tested.

The results are shown in table 3.9.

3.3.5 Evaluation of the Feature Tests and Decision

The easier deployment process and the extensibility of Axis2 are the only arguments for Platform B. Since most of the required QoS features are supported GlassFish without writing add-ons and regarding the time constraints of the project, the extensibility cannot be decisive. The better IDE-Integration and in particular the possibility of logging and debugging together with NetBeans is a good reason to choose GlassFish. The QoS-Support is a K.O.-criteria for our project because implementing multiple WS-* standards is far from realistic for the project. Due to the poor QoS-Support results of Platform B, the decision is rather easy: Only GlassFish together with the openESB-BPEL-Engine provides reasonable support for our project.

Chosen platform: GlassFish with openESB

Test	Platform A	Platform B
Test 1 - WS-Security		Module ”Rampart” is only useful for communication between Web services - a “secure” invoke was not possible in every case: BPEL invoke to “Secure” WS/BPEL was not possible
Test 1.1: User name authentication with symmetric key	passed	failed
Test 1.2: Transport security (SSL)	passed	failed
Test 2 - WS-ReliableMessaging		An “reliable invoke” was not possible - same problem as with test 1
Test 2.1: Reliable Messaging	passed	failed
Test 2.2: Reliable Messaging with exact order	passed	failed
Test 3 - WS-AtomicTransaction/ WS-Coordination	WS-to-BPEL: passed; BPEL-to-BPEL: not working	Implementation of module “Kandula2” seemingly has been stopped - so there is no existing implementation for WS-AT/WS-Coor for Platform B
Test 4 - MTOM and FastInfoset		untested
Test 4.1: MTOM	passed	-
Test 4.2: FastInfoset	passed, but no measurable impacts	-
Test 5 - SSL	passed	passed

Table 3.9: Results of the functional WS-* tests

Chapter 4

Design and Implementation

This chapter starts out with a discussion on how to map ebBP constructs to BPEL elements and on how to implement the QoS features defined in ebBP. Subsequently, Web service interfaces as a core tool for decoupling the components of this work are described. This comprises the design and correlation of messages, the description of categories of different Web services and the effect of applying WS-* technologies on the design of Web service interfaces. Finally, a description of the architecture of this work shows how the various components fit together and how these components are implemented.

4.1 Realization Strategies

In this section, a rough discussion on how ebBP elements as well as ebBP QoS properties can be realized is given. Therefore, the usage of BPEL elements, WS-* standards and project specific Web services realizing QoS properties are described.

4.1.1 ebBP to BPEL Mapping Constructs

This section shows the concepts of BPEL used to implement the ebBP constructs. For the following sections, basic knowledge of BPEL and ebBP is required, although this section focuses conceptual questions. Implementation details (Variable definitions, Correlation Set, fully defined Invokes, Receives, Replies and so on) are neglected, instead only the structure of mapping concepts is shown in the following sections. The mapping constructs are built and visualized in the BPEL design perspective of the NetBeans IDE. These constructs are to be used as samples during the phase of implementation as sample. They are not used as templates for some kind "templates engine". Furthermore, it is necessary to test the functionality of the constructs on the selected platform.

4.1.1.1 Design of the Mapping Constructs

For a better understanding, some mapping constructs are explained by the help of screen shots of the NetBeans IDE BPEL designer. Because of the conceptual character of the constructs and the missing implementation details, the screen shot sometimes show errors which are to be ignored.

Because of the complexity of this project's BPEL processes, they are split up into several constructs. Each mapping construct is comparable to a module which is used when a placeholder indicates its usage. The Empty element of the BPEL Standard is used as placeholder. All non recursive mapping constructs have been validated. This means, they have been realized and positively been tested in this work in context of generated BPEL processes, i.e., according BPEL processes have been generated, validated against its XSD, deployed on openESB's BPEL engine and its execution has been triggered using dummy backend implementations. In this sense, the constructs of Recursive Call and Collaboration Activity cannot be considered to have been validated, but note, that generation, validation and deployment works for all profile processes that employ these constructs. Alone, the execution via dummy backends has not been performed.

Business Collaboration The mapping construct for Business Collaborations builds a new BPEL process, one for each party. This construct is used only for non-internal Business Collaborations of the ebBP model. Inner Business Collaborations are discussed within the paragraph for Collaboration Activities.

The mapping starts with an outer BPEL Scope which contains all other constructs of the process. This Scope has an BPEL OnEvent and a BPEL FaultHandler, both responsible for the fault handling of the processes.

Depending on the party's role as initiator or responder, the corresponding UUID construct is used first. After that, the backend of the initiator has to be asked for the first business document using an Invoke with a corresponding Assign of the input variable of the Invoke. The following Receive waits for the business document to be sent by the initiator's backend and assigns it to a global variable that always stores the latest business document.

After receiving and assigning this message, the process steps into a Scope, enclosing the whole collaboration. For this Scope, an OnAlarm symbols the Time To Perform of the Business Collaboration of the ebBP model. Because this timer has to start with sending the first business document of the backend, it is necessary to place the mentioned Receive before the actual Business Collaboration's Scope. Otherwise, if the Receive is placed within the Scope, the timer starts before receiving the document. This Receive is the Receive element taken from the first Requesting Business Activity construct. The timer is realized as an OnAlarm Event Handler, and throws an exception, if a timeout occurs. The duration is set by the Translator if the Time To Perform type is set to `design` or `configuration`. If `runtime` is specified, the construct for timeout negotiation is necessary in front of that Scope.

The content of the Scope is a Sequence consisting of a Business Transaction Activity (BTA)

construct with a following Decision mapping construct. The Decision construct can link to other BTA-Decision sequences. This flow is visualized in figure 4.1. Not visualized are the outer Event Handler and Fault Handler of the Business Collaboration Scope. The basic structure, which is nearly common to an outer and inner Business Collaboration is shown in listing 4.1. Timeout Negotiation and other elements are not regarded within that listing, because its purpose is to give a description of the idea of the structure.

```
<process>
 <scope name="Scope_cb_profile3global">
  <faultHandlers>...</faultHandlers>
  <eventHandlers>...</eventHandlers>
  <sequence name="Sequence_cb_profile3global">
   <empty name="Empty_UUID_Handling"/>
   <assign name="Prepare_Invoke_ReadyToReceive_BD"/>
   <invoke name="Invoke_ReadyToReceive_BusinessDocument"/>
   <receive name="Receive_Business_Document"/>
   <assign name="Assign_After_Receive_BD"/>
   <scope name="Scope_cb_profile3global_timeout">
    <eventHandlers>
     <onAlarm>
      <for><!-- DURATION --></for>
      <scope>
       <sequence>
        <throw name="Throw_BusinessCollaboration_Timeout"/>
       </sequence>
      </scope>
     </onAlarm>
    </eventHandlers>
    <sequence name="Sequence_cb_profile3global">
     <sequence name="Sequence_bta_plus_decision">
      <empty name="Empty_BTA"/>
      <empty name="Empty_Decision"/>
     </sequence>
    </sequence>
   </scope>
  </sequence>
 </scope>
</process>
```

Listing 4.1: Basic structure of the mapping for BCs and inner BCs

The Fault Handler of the Business Collaboration Scope (shown in listing 4.2) contains a Catch All construct for catching all exceptions within the process. This includes protocol failures and timeouts and maps the ebBP Final State `techFail`. All errors which occur in the process are thrown to this outer Fault Handler which is responsible for controlled termination of the process. If an error is caught, it is necessary to inform the backend, the partner process, if reachable, and any child process, if available. This is done within the context of a Flow which ends with `SystemFailure`. If the partner process is not reachable, an empty Fault Handler catches the exception and ends this path of the Flow.

```
<faultHandlers>
 <catchAll>
  <sequence>
   <flow>
    <!-- Information of the Backend -->
    <sequence>
     <assign name="Assign_SystemFailureMessage"/>
     <invoke name="Invoke_SignalSystemFailureMessage_Backend"/>
    </sequence>
```

```
10     <!-- Information of the Partner Process -->
11     <scope>
12      <faultHandlers>
13       <catchAll>
14        <empty name="Empty_DoNothing"/>
15       </catchAll>
16      </faultHandlers>
17      <assign name="Assign_SystemFailureMessage"/>
18      <invoke name="Invoke_SignalSystemFailureMessage_PartnerProcess"/>
19     </scope>
20     <!-- Information of the Child Process -->
21     <if name="If_Active_Child_Process">
22      <assign name="Assign_SystemFailureMessage"/>
23      <invoke name="Invoke_Terminated_Parent_Process"/>
24     </if>
25    </flow>
26    <exit name="Exit_SystemFailure"/>
27   </sequence>
28  </catchAll>
29 </faultHandlers>
```

Listing 4.2: The mapping construct of the outer Fault Handler as simplified BPEL XML code

Corresponding to that Fault Handler, each process has an OnEvent Handler waiting for the message of the partner process that signals the `SystemFailure`. When this event occurs, a Flow starts to propagate the error to the backend and to available child processes before it ends with `SystemFailure` by itself.

For later use, each BPEL process has to define some variables, which provide global informations, like the content of the latest business document, the latest business signal or other necessary informations. Therefor, all Invokes have an Assign for preparing the input variables according to these informations. Each Receive element and some Invoke elements are followed by an Assign element used for setting these variables by extracting the informations from the output variables.

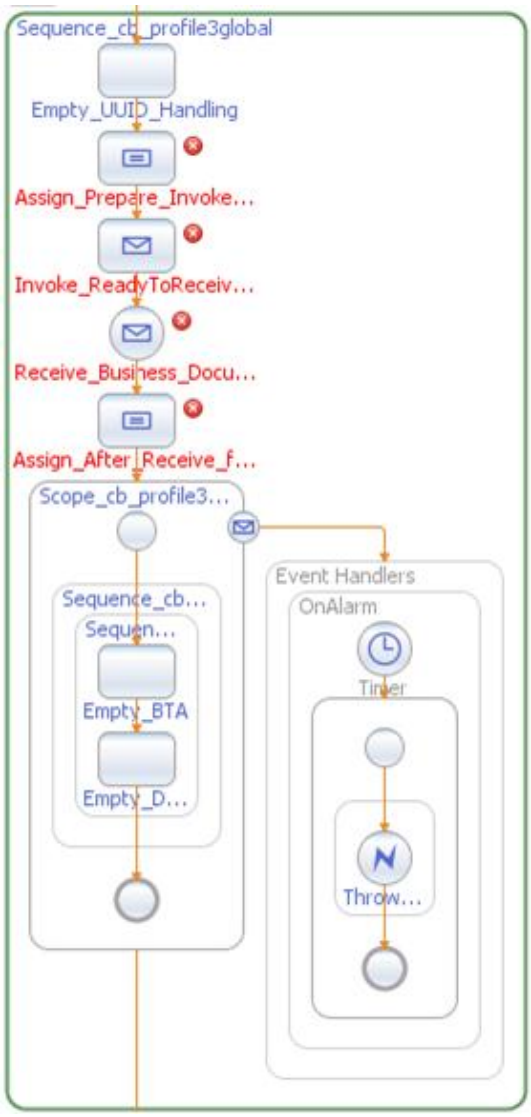


Figure 4.1: Mapping construct for Business Collaboration

Collaboration Activity This mapping construct is similar to the construct for Business Collaborations and is used for mapping inner Business Collaborations of the ebBP model. This means, each inner Business Collaboration is mapped to a new process started by a parent process (the mapping of the calling Business Collaboration) to which it has to signal its final states. The basic structure of the mapping for an inner Business Collaboration is similar to the structure of an outer one, as shown in listing 4.1.

This construct also starts with a Receive. But, because it is the mapping of a recursive invocation, the sender of the message is another process, namely the parent process, instead of the backend system. The Receive is followed by Assigns to set global variables as described in the paragraph of the mapping of Business Collaborations. Among others, these are recursion informations like the `recursionCount` passed in by the parent process. The `recursionCount` is concatenation of the pattern `(Type)Nr.` . Each parent process concatenates the recursion count with the type of the child process in parentheses, followed by a number and a dot.

Furthermore, like in the outer Business Collaboration mapping, the outermost Sequence of the inner Business Collaboration contains the construct for timeout negotiation and the constructs for receiving the first business document (also described in the paragraph for Business Collaborations). The rest of the main Scope is not changed.

Changes are only made to the outer Event and Fault Handlers. The OnMessage Event Handler is waiting for an error message of an terminating parent process or an terminating child process. The Event Handler contains a Flow, informing the child process and the parent processes of termination messages. A message to the child process is transmitted, if a boolean variable states that child processes are active. In case the child process is not reachable, the Invoke used for informing the child process is enclosed by a Scope with an empty Fault Handler in order to provide for a controlled termination of the Flow.

The branch of the event Flow that is responsible for informing the parent process has an Assign for filling the message followed by an Invoke for message transmission. To ensure controlled termination of the Flow in case of a non-reachable parent process, the Invoke must be enclosed by a Scope with an empty Fault Handler.

After termination of the Flow, the Event Handler terminates the process.

Similarly, the Fault Handler has a Catch All element with a Flow element containing branches to inform the parent process, active child processes and the partner process. Information in each case means assigning a specified message and sending it to the corresponding interfaces using the Invoke element. Informing the child process is enclosed within an If-clause for checking if a child process is active.

The idea behind Fault Handler and Event Handler is that the partner process is informed about the error at the recursion level on which the error occurs. Further, each partner's side propagates the error in both directions, to higher and lower recursion levels. Thus, there is only one error message between the processes of the integration parties at the level of the occurrence of the error.

UUID Distribution The distribution of a UUID in the BPEL process does not correspond to an element of the ebBP model, but for identification and later reproducibility each process needs a UUID. It is important that both partner's orchestrations use the same UUID. For this purpose, the initiator process of the collaboration receives a message of the backend which initiates the UUID distribution. In a next step, the global variables of the process like `messageID, profileID, currentRole, partnerRole, recursionCount, activeChildProcesses` are set. This is based on information sent by the backend.

The following Assign prepares the call of the Web service that is responsible for creating an UUID. After this service has been called, an Assign sets the returned UUID as a global variable. The following steps are to prepare messages for the partner process and the backend system and using corresponding Invokes for UUID distribution. After the Invoke towards the backend, the Correlation Set is initialized. Figure 4.2 visualizes the construct for initiator and responder.

At the responding side, the process waits to receive a UUID distribution message from the partner process. After receiving the UUID the responder uses several Assigns for setting global variables and preparing a message for UUID transmission to the backend. Together with the Invoke of the UUID towards the Backend the Correlation Set of the responder process is initialized.

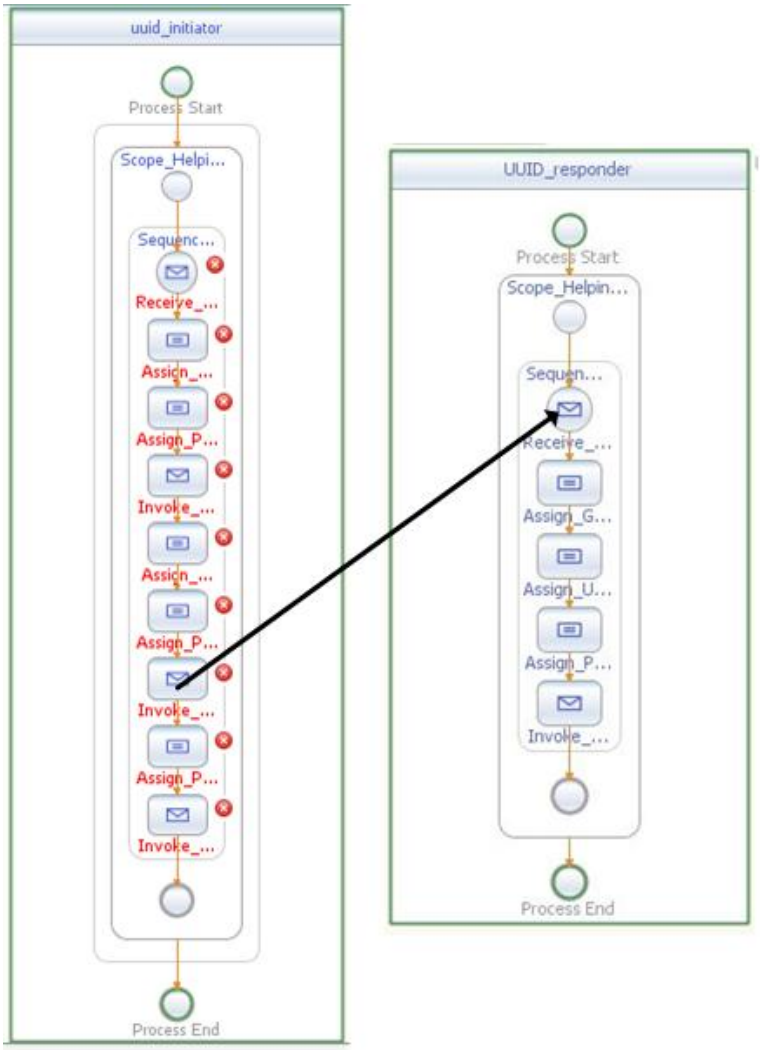


Figure 4.2: Mapping construct for UUID distribution

Acceptance Acknowledgement This mapping construct is used in order to map the Acceptance Acknowledgement elements of ebBP models. In this work, the effect of this element is a Schematron validation executed by a corresponding Web service. This check has to be performed only by the party that receives the business document while the sending partner has to wait for this business signal. Due to this fact, the description of this construct has to be done for the requesting role as well as for the responding role. The requesting role describes the party that sends the business document while the responding role describes the receiving one. Figure 4.3 shows the mapping construct for both sides (the left one for the requesting role, the right one for the responding role).

To begin with, the requesting role is considered. After having sent a business document the sender has to wait for the business signal `AcceptanceAcknowledgement`. This is mapped by a BPEL Receive element which waits for the call of the corresponding process. After receiving the signal, it is assigned to a global variable that stores the latest incoming business signal. This Sequence is enclosed by a Scope with an OnEvent Handler and an OnAlarm Handler. The OnEvent Handler is listening for a message containing the Business Signal `AcceptanceAcknowledgementException`. If this message arrives, the message details are assigned to global variables for reuse by other elements. At the end, an exception is thrown that will be caught by the CatchAll element of the outer Fault Handler of the Business Collaboration.

The waiting time for business signals is limited by an OnAlarm Handler which maps the Time To Acknowledge Acceptance element of the ebBP model specified within the Requesting or Responding Business Activity. If a timeout occurs an exception is thrown and caught by the CatchAll element of the outer Fault Handler of the Business Collaboration mapping. This Fault Handler is responsible for informing the partner process of the occurrence of an error. For this reason, an OnAlarm Handler for the mapping of the Acceptance Acknowledgement at the responding side is not necessary.

The right half of figure 4.3 illustrates the handling of the Schematron validation at the responding process. If an Acceptance Acknowledgement is specified in the ebBP model the responding party has to execute Schematron validation upon the incoming Business Document using the corresponding Web service. For this purpose, an Invoke element enclosed by two Assigns that are responsible for assigning input and output variables are used. A following If-clause checks the success of the validation based on the output variable of the Schematron Web service call. If the validation has been successful a message is filled with the business signal `AcceptanceAcknowledgement` and is used as input variable for the Invoke towards the partner process. The else case works similarly. The `AcceptanceAcknowledgementException` is sent to the partner process before an exception is thrown that is to be caught by the outer Fault Handler of the Business Collaboration mapping.

OnAlarm Handlers used for mapping the Time To Acknowledge Acceptance are only specified for the requesting party. The requesting role will throw an exception to the outer Fault Handler of the Business Collaboration if a timeout expires. So, the partner process will be informed of that failure. Another reason for this choice is that the timer of the responding party will start later than the timer of the requesting party. Thus, the two timers would not run synchronously. Usually, the responding role is informed of a timeout occurrence via outer Fault/Event Handler before its own timeout would expire. As WS-ReliableMessaging is envisaged for message

transmission it is also safe to assume that the responding side would either receive the error message or would not be able to deliver the Acceptance Acknowledgement message. Due to this, the timeout at the responding role is omitted.

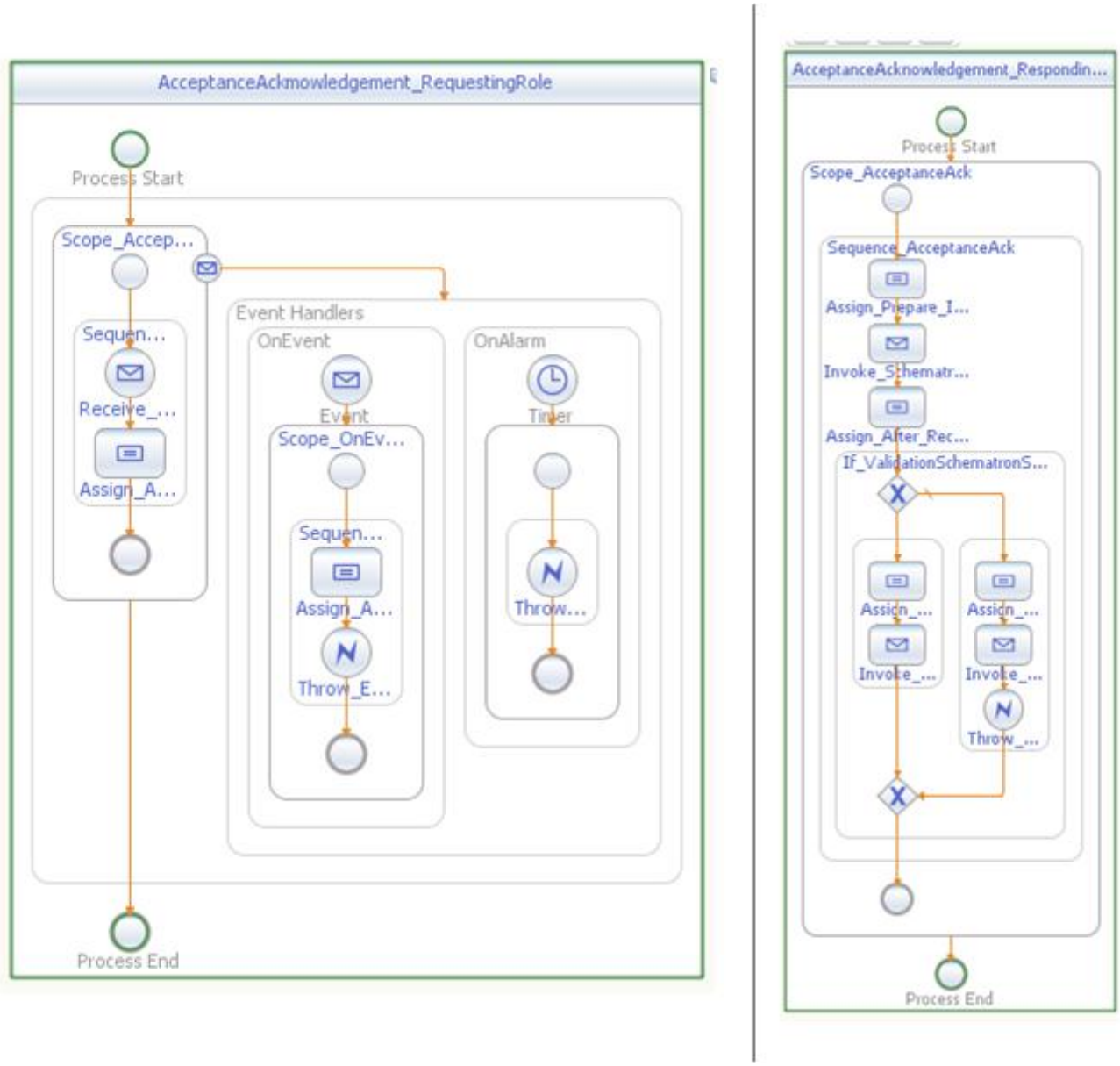


Figure 4.3: Mapping construct for AcceptanceAcknowledgement

Archive This mapping construct represents the realization of a call to the so-called Archive Web service that is responsible for archiving messages. The realization of multiple QoS attributes requires this archival service. For details see section 4.1.2. It can be used at the side of the requester as well as at the side of the responder of a Requesting or Responding Business Activity.

A BPEL Assign prepares the message to be archived. This Assign is needed for setting the fields of the message Meta Block (a data structure introduced in this work for conveying meta data) and copying the message to the body. An Invoke element sends the prepared message to the responsible Web service. The Archive service signals via the output variable of the Invoke whether the archiving has been successful. A following If-construct checks the success and, if a negative answer is sent back, an exception is thrown and handled by the outer Fault Handler of the Business Collaboration mapping construct. Figure 4.4 shows the visualization of this mapping construct.

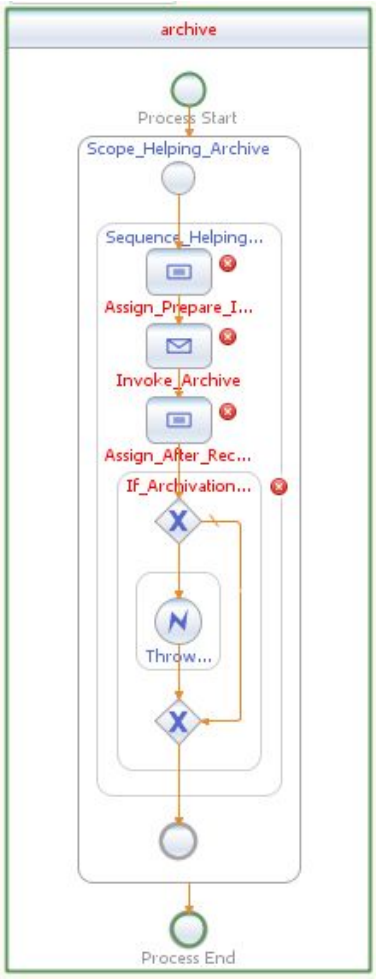


Figure 4.4: Mapping construct for a call of Archive service

Authorization Check This mapping construct concerns the responding party. It is used if the corresponding QoS attributes are set inside the ebBP model. For more details about which features lead to the use of this construct, see section 4.1.2.

Within a BPEL Sequence, the message for the Invoke of the corresponding Web service is prepared by the BPEL Assign element. The Invoke's output message which states whether authorization can be granted is prepared for being checked by an If construct by an Assign. If there is a negative answer an exception is thrown which is handled by the outer Fault Handler of the Business Collaboration construct. If the answer is positive the flow of the process can proceed. Figure 4.5 shows a screenshot of the NetBeans IDE visualizing this construct.

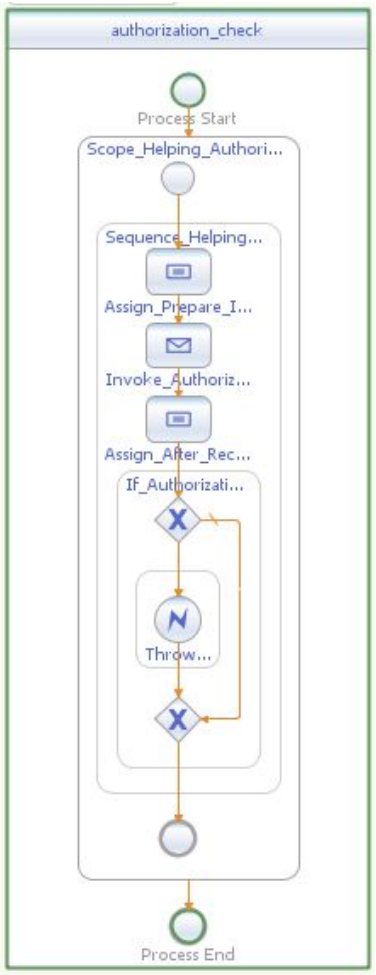


Figure 4.5: Mapping construct for a call of the Authorization Check service

Business Transaction Activity Figure 4.6 shows the mapping of Business Transaction Activities. If the Time To Perform is defined as `runtime`, a runtime negotiation construct in front of the scope of the BTA mapping is necessary. The negotiated timeout or the specified timeout of the ebBP model is set to an OnAlarm Event Handler that throws an exception if a timeout occurs. The timer has to start after receiving the business document. Therefore, it is necessary that the Invoke of the Ready to Receive message, the receipt of the business document and the Assigns of the corresponding messages have to be extracted from the construct for the Requesting Business Activity and be placed outside of the scope. But this is only the case if the mapped Business Transaction Activity is not the starting activity of the Business Collaboration. If it is the first one the mentioned constructs are moved outside of the inner Scope of the Business Collaboration construct.

The mentioned Ready to Receive message is necessary to signal the backend system that the process has reached the state for handling the next message. The rest of the Business Transaction Activity mapping is a sequence of constructs for the Requesting and the Responding Business Activity.

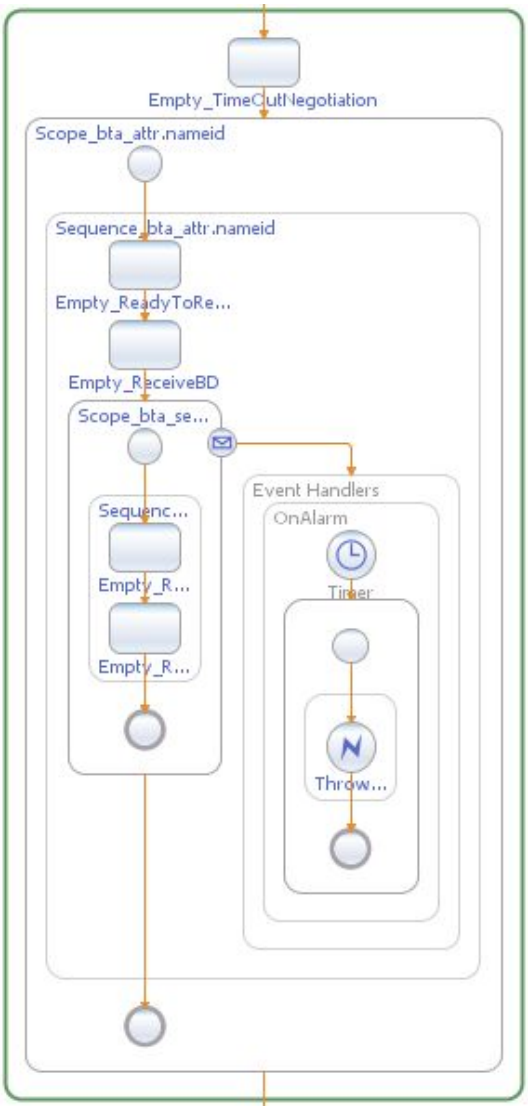


Figure 4.6: Mapping construct for Business Transaction Activity

Decision This mapping construct is the BPEL mapping for the Decisions of the ebBP models. The branches for protocol failures are, as mentioned yet, mapped by the outer Fault Handler of the enclosing Business Collaboration construct.

The other branches are handled sequentially in the order they are specified in the ebBP model. For each branch, at first a boolean variable is set to true if the condition for this branch matches.

This can be done by evaluating the `isPositiveResponse` field of the Meta Block in case the condition is specified as a Conditional Guard Value, or by using the Web service for evaluating expressions in case the condition is an `XPATH2` expression. Accordingly, the boolean variable is set by an Assign within an If/Else construct.

Then, in a following If/Else-construct, the boolean variable for the respective decision branch is checked. The body of the If clause contains the mapping of the branch that corresponds to a positive decision specified in the ebBP model. The next `ToLink` element of the ebBP decision is mapped in the same manner and nested to the Else clause of the previous If/Else construct. This is done for all decision branches of the ebBP model.

Listing 4.3 shows the mapping as a simplified extract of the BPEL XML code.

The last Else clause throws an exception which signals that no previous condition evaluates to true. This means something went wrong and the process has to end.

```
<sequence>
 <scope>
  <sequence>
   <if>
    <condition>lastBusinessDocument/MetaBlock/IsPositiveResponse = true()
    </condition>
    <assign>
    <!-- Assign Var_local_expression_evaluation_result to true()-->
    </assign>
    <else>
     <assign>
      <!-- Assign Var_local_expression_evaluation_result to true()-->
     </assign>
    </else>
   </if>
  </sequence>
 </scope>
 <if>
  <condition>Var_local_expression_evaluation_result = true()
  </condition>
  <scope>
   <sequence>
    <!-- Mapping of the branch, which is used if the condition is true -->
   </sequence>
  </scope>
  <else>
   <scope>
    <sequence>
     <scope>
      <sequence>
       <if>
        <!-- Check of the condition for the next ToLink-element of the ebBP model -->
       </if>
      </sequence>
     </scope>
```

```
36      <if>
37       <condition>Var_local_expression_evaluation_result = true()
38       </condition>
39       <!-- Mapping of the branch, which is used if the condition is true -->
40       <else>...</else>
41    ...
42 </sequence>
```

Listing 4.3: Simplified extract of the BPEL mapping for ebBP Decisions

Receipt Acknowledgement The mapping construct of the Receipt Acknowledgement comprises the mapping of the elements Receipt Acknowledgement, Receipt Acknowledgement Exception and their usage within the ebBP models. The handling of this mapping construct differs between requesting and responding role of a Requesting/Responding Business Activity.

The requesting role is attached to the sender of the business document. After having sent the business document, the requester is waiting for a confirmation. This is realized within a BPEL Scope and a BPEL Receive element. The Receive is waiting for a call that carries the Receipt Acknowledgement. When this business signal arrives it is assigned to global variables for later use, and if specified by QoS attributes in the ebBP models, the business signal is archived and its signature is checked using the corresponding Web services. This is visualized in figure 4.7. As mentioned before, the BPEL Empty element is used as a placeholder for other mapping constructs. The enclosing Scope has an EventHandlers element with one OnEvent Handler and one OnAlarm Handler.

The OnEvent Handler is responsible for a possible incoming call transmitting a Receipt Acknowledgement Exception. In case this event occurs, the business signal is assigned to global variables and, if specified by QoS attributes, archived. It is not necessary to throw an exception because this is done by the Receipt Acknowledgement handling procedure at the responding party. There, an exception is thrown, caught by the outer Business Collaboration Fault Handler and forwarded to the outer Business Collaboration Event Handler of the requesting party. The consequence is a controlled termination of both processes.

The OnAlarm Handler of the Scope maps the Time To Acknowledge Receipt element of the ebBP model. The timer starts after the requesting side has transmitted the corresponding Business Document. If a timeout occurs, a timeout exception is thrown to be caught by the Fault Handler of the Requesting/Responding Business Activity construct and not, like all other exceptions, by the outer Fault Handler of the Business Collaboration. The consequence is the incrementation of the retry count variable.

The responding role is responsible for sending business signals depending on ebBP QoS attributes and the content of the business document. The BPEL mapping is visualized in figure 4.7 on the right-hand side. The ebBP QoS attribute `isIntelligibleCheckRequired` states whether the received business document has to be checked against its XML schema. If this attribute is set to true, the document is assigned, together with other fields, to a message and is sent to the Web service responsible for XSD validation using an Invoke element. The Invoke's output variable is checked by an If/Else-construct for validation success. In case of success, the Receipt Acknowledgement is created and assigned to local variables. The business signal is

signed and archived, if specified by QoS attributes, and sent to the requesting process.

In case of validation failure, the same is done for creating and assigning the Receipt Acknowledgement Exception business signal. In this case, at the end of the flow, an exception is thrown to the outer Fault Handler of the enclosing Business Collaboration construct which is responsible for informing the partner process of a fault and for ending the own process. If `isIntelligibleCheckRequired` is set to false, the Receipt Acknowledgement is sent and handled as mentioned above without XSD validation and If/Else construct checking the success of validation.

The mapping of the Time To Acknowledge Receipt is used only within the construct for the requesting role using the above mentioned OnAlarm Handler. The reasons for this are the same as pointed out in the paragraph for mapping the Acceptance Acknowledgement and the Time To Acknowledge Acceptance.

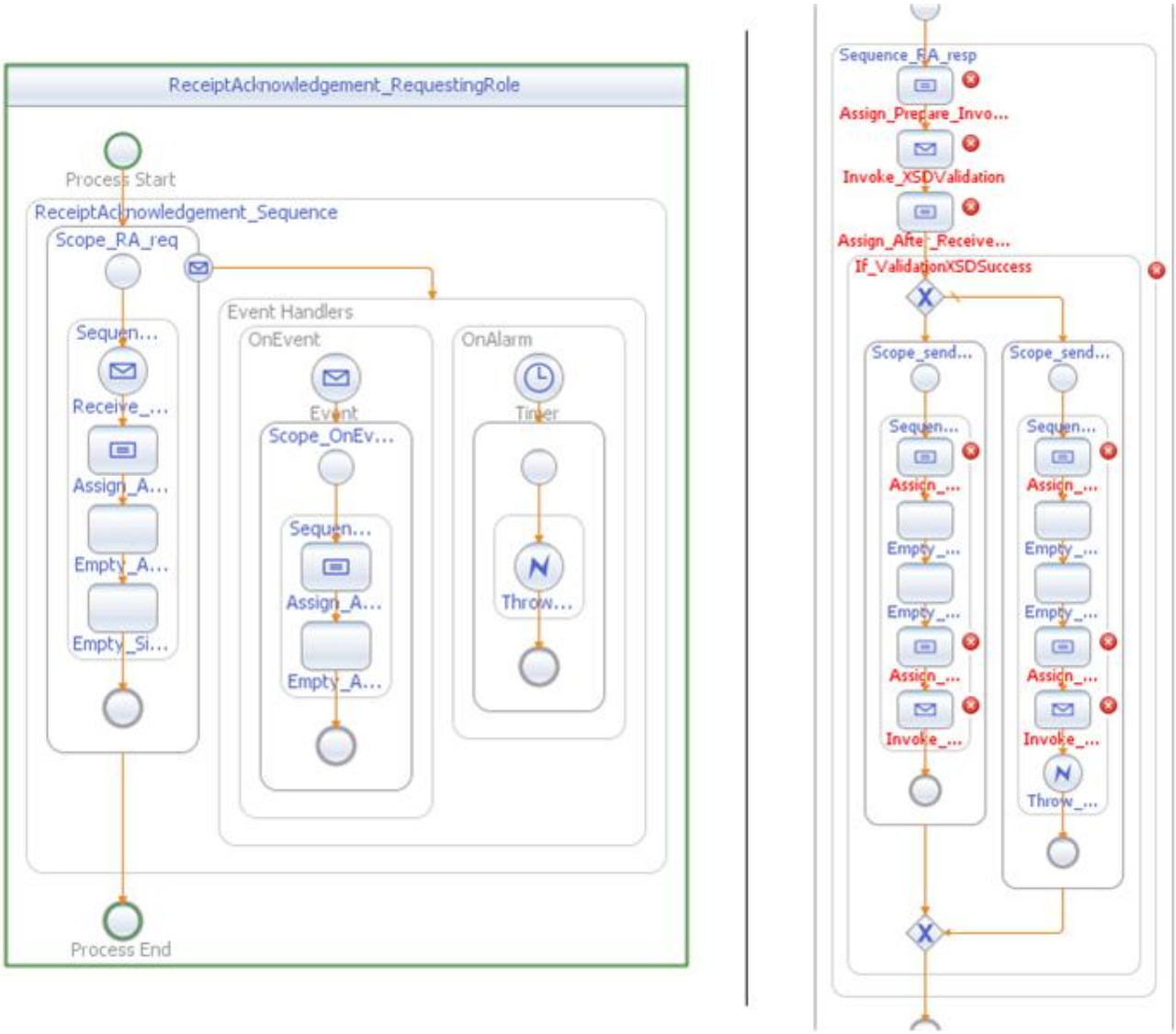


Figure 4.7: Mapping construct for ReceiptAcknowledgement

Recursion Call The mapping construct for the recursion call is used instead of a sequence of Business Transaction Activity constructs with corresponding Decision constructs as described in the paragraph of the Business Collaboration construct. A Collaboration Activity and the referenced Business Collaboration is represented by a separate BPEL process. If the ebBP model includes a Collaboration Activity, a new process has to be called by the BPEL process. For this purpose, the attributes for the new process (child process) are assigned and an Invoke of the method of the child process is performed. It is important for error message handling of the outer Event and Faulthandlers of the Business Collaboration construct to set the boolean variable for active child processes to true. Then, a Receive follows that waits for the Invoke of the result from the child process.

The Decision that follows a Collaboration Activity in an ebBP choreography is mapped by a BPEL If construct that checks the child process's Recursion Stop Message for errors and their error codes. If an error occurs an exception is thrown to the outer Fault Handler. If not, the flow can go on.

After receiving the output message of the recursion, the boolean variable for active child processes is assigned to false.

The mapping of the call of a child process is shown as simplified XML BPEL code in listing 4.4.

```
<sequence>
 <assign name="Assign_Increment_Recursion_Counter"/>
 <invoke name="Invoke_Start_Collaboration_Activity"/>
 <assign name="Assign_Child_Process_Active"/>
 <receive name="Receive_Recusion_Message"/>
 <assign name="Assign_No_Active_Child_Process"/>
 <if name="If_Error_In_Recursion">
  <sequence>
   <throw name="Throw_AnyProtocolFailure"/>
  </sequence>
 </if>
</sequence>
```

Listing 4.4: Simplified extract of the mapping for a recursion call

Requesting / Responding Business Activity This mapping construct is identical for the mapping of Requesting and Responding Business Activities. But it is a role specific mapping. This means the construct for the requesting role and the construct for the responding role are different. If a party has the requesting role within the Requesting Business Activity, it has to perform the responding role within the Responding Business Activity and vice versa.

The mapping for the responding role is a Sequence starting with the Receive of the business document. After that, the message is assigned to different global variables for later use, and, if specified by ebBP QoS attributes, the document is archived and its signature as well as the authorization of the sender are checked using the corresponding BPEL mapping constructs for these purposes. After that, the Sequence uses the constructs for handling the Receipt Acknowledgement and the Acceptance Acknowledgement before assigning and invoking the business document towards the backend system.

The requesting role, as visualized in figure 4.8, is enclosed within a While loop which represents the mapping of the Retry Count. The termination condition is reaching the retries specified in the ebBP model. Before entering the While-loop, a variable for counting the retries is set to zero and a variable for the maximum possible retries is set to the value specified in the ebBP model.

The body of the loop has a construct that is responsible for sending the business document to the partner process and that is composed of an Assign and an Invoke element. This is followed by a Scope containing a Flow with one branch for handling the Receipt Acknowledgement and with another branch for handling the Acceptance Acknowledgement. In both cases, the corresponding construct for the requesting role has to be included. For more details, see the paragraphs of these constructs.

If the Flow ends, the constructs for Receipt Acknowledgement and Acceptance Acknowledgement ended correctly and the loop is left by setting the counter of retries to the maximum number using the Assign element. The Scope within the loop has a Fault Handler with a Catch element. This is for catching the timeout exception of the Receipt Acknowledgement construct. As specified in chapter 4.1.2, a timeout during the handling of Receipt Acknowledgement increases the retry counter. This is mapped by an Assign of the variable for actual retries within the Catch element that increases the latter if the maximum number of retries is not reached yet. If the maximum number of retries is reached within the Fault Handler, which means delivery of the Receipt Acknowledgement failed, an exception is thrown by the Fault Handler to the outer Fault Handler of the enclosing Business Collaboration. This is done in order to terminate the process with failure messages to backend system, child processes and partner process.

Figure 4.8 shows a simplified visualization of the mapping of the while construct.

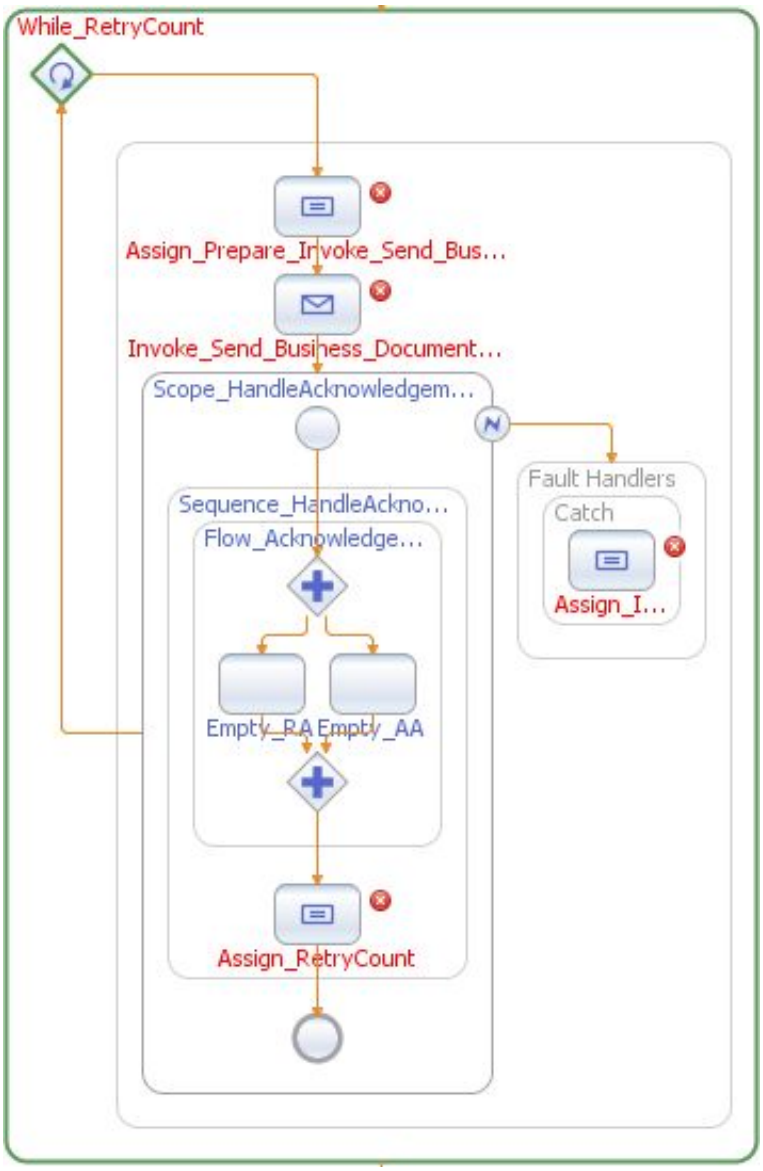


Figure 4.8: Mapping construct for Requesting Business Activity of the Requesting Role

Runtime Timeout Negotiation The construct of the timeout negotiation is used if a Business Transaction Activity or a Business Collaboration has `runtime` specified as Time To Perform type within the ebBP model. This means, no concrete duration value is specified. Therefore, the initiator process invokes a method for getting a duration for the corresponding Time To Perform from its backend. Checks for consistency, e.g., whether the sum of the duration of all Business Transaction Activities is less than the duration of their enclosing Business Collaboration are assumed to be performed by the linked backend system and are not performed within the BPEL process. After the receipt of the duration it is assigned to the corresponding OnAlarm Event. A next step in this Sequence assigns the duration to the input message for the Invoke to distribute it to the partner process. The initiator waits for a Reply of the partner process that confirms the timeout negotiation.

The responding process waits with a Receive for this duration, assigns it to the OnAlarm Event and confirms the timeout negotiation with a Reply to the initiator process.

Signature Creation This construct (shown in XML BPEL listing 4.5) is used if it is necessary that an outgoing message contains a signature. For this purpose, the message has to be assigned to the input variable of the service Invoke together with other addressing attributes. The output variable containing the signed message is assigned to the corresponding global variable for further handling of this message.

```
1 <sequence>
2  <assign name="Assign_Prepare_Invoke_SignatureCreation">
3  </assign>
4  <invoke name="Invoke_SignatureCreation"/>
5  <assign name="Assign_After_Receive_from_Var_Output_SignatureCreation">
6  </assign>
7 </sequence>
```

Listing 4.5: Mapping construct for calling the Signature Creation Web service as BPEL XML

Signature Check This construct is used to implement the call of the Web service signature check. The usage of this construct depends on the setting of QoS attributes of the ebBP model as pointed out in chapter 4.1.2. For a call, a message containing the latest business document to be checked has to be assigned. After this, the Web service for checking the signature is called. The received output of the service is assigned to a variable which is checked in the following if-clause. If the signature check fails an exception is thrown, otherwise, the process can proceed. Figure 4.9 visualizes this construct.

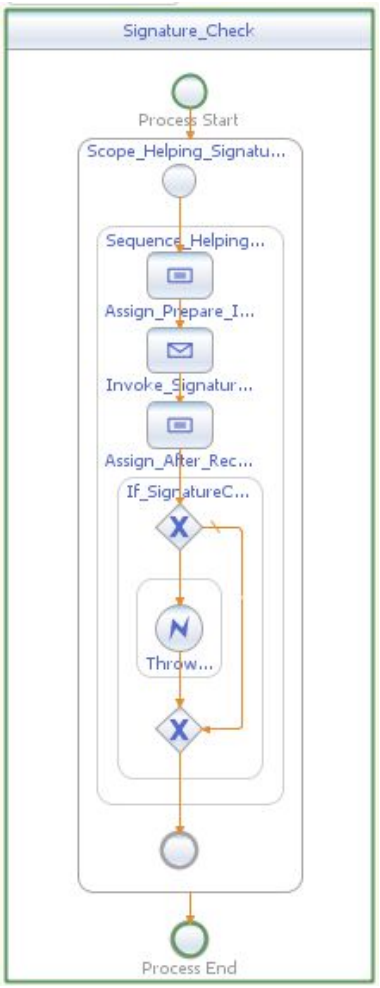


Figure 4.9: Mapping construct for a call of the Signature Check service

4.1.1.2 Validation of Mapping Constructs

Most simple BPEL constructs have been tested during the platform selection process. So the validation of all mapping constructs introduced in section 4.1.1.1 is not necessary. Thus, only more complex constructs such as fault and event handlers as used in this project are checked.

The tests in the following list have more than one goal: Investigating the functionality of BPEL elements which are not used regularly and validating openESB support/implementation of these elements.

1. **Test scenario:** After a "Throw" activity the process has an "Assign" and a "Reply". The fault handler catches all faults.

   Question: Will the scope be left at once after the "Throw" as prescribed by the BPEL standard or will the next steps be executed?

   Result: The scope is left after the "Throw" - the assign and reply is ignored.

2. **Test scenario:** Fault handling with BPEL - Two BPEL-Processes A and B; A calls process B; an exception occurs in process B.

   Question: How can the exception be reported to Process A?

   Result: In the WSDL file of process B a fault must have been declared, process B has to Reply a "fault response", then process A can catch the fault with a BPEL fault handler.

3. **Test scenario:** In a sequence a fault is thrown. This sequence is encapsulated in a scope A. This scope has a catch-all fault handler which catches the fault and then throws another fault. Scope A is enclosed within another scope B which also has a catch-all fault handler.

   Question: Which fault handlers catch the faults?

   Result: The first fault from the sequence is caught by the fault handler of the inner scope A. The fault thrown by this fault handler is caught by the outer scope B. So faults can be delegated step by step to superordinate fault handlers.

4. **Test scenario:** This scenario consists of two scopes. The inner scope has onAlarm-Event-Handler which is set to 3 seconds and a catch-all fault handler which replies a fault message with an identifier for the inner scope to the caller. The outer scope also has a catch-all fault handler which also returns a fault message. This fault message carries an identifier for the outer scope. The inner scope simply waits 1 minute in order to trigger the onAlarm handler. The onAlarm handler thus throws a fault (BPEL activity "Throw").

   Question: Which fault-handler catches the fault?

   Result: The fault handler of the inner scope catches the fault.

5. **Test scenario:** The BPEL standard postulates the implementation of Isolated Scopes. Therefore, an XML attribute "`isolated`" is introduced for the XML element "`scope`". When two isolated scopes try to access the same variable, the access should be serialized. (See [JEA+07] Section 12.8)

Question: Can isolated scopes be used in this case study? Does GlassFish support isolated scopes?

Result: No - Warning in NetBeans "Isolated Scopes are not supported", Tag will be ignored; Test failed

6. **Test scenario:** This scenario comprises two independent BPEL processes. Process A implements a timer. A one-way-message starts the timer. When the timer ends, a one-way-message is sent to process B, whose main scope has an event handler to handle the receipt of this message. In its main scope process B starts the timer of process A and waits for a little while.

   Question: How can the BPEL-engine distinguish between process instances when multiple messages are sent to an endpoint which is listened to by an event handler?

   Result: With correlation sets. Correlations can be used on receive, reply, invoke, onMessage and onEvent. They map an identifier of one message to the one of another message. The NetBeans-BPEL-Editor offers the possibility to generate user-defined correlations. A logging of the ID at the event handler showed that multiple messages with different timers were received by the expected process instances.

4.1.2 Realization of QoS Features

For realizing the QoS-Features that can be specified in ebBP models (cf. section 3.2), there are several options. The options used in this work are:

- Usage of (WS-*) Standards
- Development of Web services
- Usage of BPEL constructs

If the attribute `isIntelligibleCheckRequired` is active (set to true), the document has to be checked for syntactical and semantic conformance to the XML and NES guidelines. For this reason, the transmitted XML documents have to be checked for their XML conformance. Thus, XSD validation has to be performed. Therefore, an XMLValidation Web service is necessary which receives an XML file, checks this file and returns either true if it is valid or false otherwise. This Web service checks the syntax of the XML file, but there is no check of the semantics. However, the NES group provides in addition to XSD files for syntactical validation also Schematron files which allow a semantic check of the XML files with the help of an XSLT (eXtensible Stylesheet Language Transformation) processor. For this purpose an additional Web service has to be provided which is quite similar to the XMLValidation Web service but checks the semantic validity of the document (Web service SchematronValidation).

The attributes `isNonRepudiationRequired` and `isNonRepudiationReceiptRequired` have to guarantee that neither the sender nor the receiver can deny the receipt of a business document.

For this reason, the sender of the business document for the attribute `isNonRepudiationRequired` respectively the sender of the receipt acknowledgement for the attribute `isNonRepudiationReceiptRequired` have to sign the according message. This message has to be persisted on both sides. For persisting the messages the simplest way is to use a Web service which writes the transmitted messages to the file system and saves the meta-data in a database. For using signatures with Web services, the WS-* standards WS-Security [OAS06] and WS-Trust [LKN+07] provide methods and mechanisms to create and validate signatures, but the Security Token concept of WS-Trust which needs a central instance to issue the tokens, is inapplicable for this work, and the signature mechanisms of the WS-* standards are using the SOAP message header to put the signature information into the message. This makes the signature completely transparent for BPEL processes because the used BPEL engine does not allow to access the SOAP message header. But the XML signature has to be contained in the archived file because of non-repudiation requirements so that the realization of signatures using WS-* standards has to be canceled. This is the reason why a pair of Web services for creating and checking an XML signature has to be established. For both attributes discussed in this paragraph the Web services SignatureCreation and SignatureCheck are introduced for creating and validating signatures.

The QoS attribute `hasLegalIntent` is not precisely defined in ebBP (cf. [YWM+06], section 3.4.9.7). Thus, in this work, `hasLegalIntent` is realized as an aggregated attribute which sets every of the following attributes to true:

- `isGuaranteedDeliveryRequired`
- `isIntelligibleCheckRequired`
- `isNonRepudiationRequired`
- `isNonRepudiationReceiptRequired`

If the attribute `isConcurrent` is active the needed resources for this transaction have to be locked. In the context of Web services the locking of resources is envisaged to be realized using WS-Coordination [NRFJ07] with the sub-standards WS-AtomicTransaction [NRLW07] and WS-BusinessActivity [NRLF07]. However, for none of these standards, a fully functional implementation could be found. Writing an own implementation within the project was, due to limited resources and time, not possible. Another approach for realizing this locking mechanism is to write a semaphore Web service which controls the access to shared resources. The review of the NES profiles showed that there are no shared resources with limited access in the use case of this project. So the attribute `isConcurrent` has no direct impact on this work's use cases and therefore the development of the semaphore Web service has been canceled in favor of other project priorities.

All the attributes concerning time (`TimeToPerform`, `TimeToAcknowledgeReceipt`, `TimeToAcknowledgeAcceptance`) are realized using timers in the BPEL processes.

For the attribute `isGuaranteedDeliveryRequired` one realization option is the usage of the WS-ReliableMessaging [FPD+07] standard. Another possibility is to design the BPEL pro-

cess in such a way that systematic checks in the process avoid consistency issues, i.e., distributed commit techniques could be applied. The standard WS-ReliableMessaging has several implementations which are relatively stable and simple to use. This is the reason why WS-ReliableMessaging has been used to implement the `isGuaranteedDeliveryRequired` attribute for communication between the BPEL processes. For other message transmissions in this project, e.g., BPEL to Backend, a secure and reliable connection is assumed to be available. Note that this is not far from realistic because the BPEL process and the Backend implementation of one partner are likely to be run on the IT resources of one IT department. If this is not the case, WS-ReliableMessaging and distributed commit mechanisms ar still an option.

The attribute `isAuthorizationRequired` checks whether an user is authorized to perform the action she wants to perform. In the case of this project the authorization check performs a check of the privilege of a user to send a particular business document in a particular profile, e.g., send an order in NES profile three. This functionality is realized by using a Web service (Web service AuthorizationCheck) which gets the parameters user, profile and message type and returns whether the user has this privilege (true) or not (false).

The `retryCount` for the ebBP delivery retries of messages can be realized as a BPEL loop.

For the different QoS security features (`isAuthenticated`, `isConfidential` and `isTamperDetectable`) a differentiation between transient (on network level) and persistent (on application/document level) realization as arranged for by ebBP is necessary. Each of these attributes is realized with Secure Sockets Layer (SSL) on network-level which ensures encryption of the transport channel (this realizes `isConfidential`), authentication by certificates of one or both sides (realizing `isAuthenticated`) and provides a mechanism which detects non-authorized modification of the transmitted content by using a hash-function. Note, that Point-to-Point security is sufficient for this work's uses cases because intermediaries are not assumed.
If the attribute `isConfidential` is set to persistent, WS-Security is used to encrypt the XML message. To realize the attributes `isAuthenticated` and `isTamperDetectable` on document-level an XML signature is added to the XML document. First, this signature can be used to authenticate the person who signed the message, and second, an XML signature generates a hash-code value for the document to be signed. The validation of this hash code fails if the document is not completely equivalent to the signed XML document.

For a listing of the QoS attributes with their realization strategies see table 4.1.

QoS Attribute	Realization strategy
isAuthenticated	transient: TLS persistent: Web Service SignatureValidation
isConfidential	transient: TLS persistent: WS-Security
isTamperDetectable	transient: TLS persistent: Web Service SignatureValidation
isIntelligibleCheckRequired	Web Service XMLValidation Web Service SchematronValidation
isNonRepudiationRequired	Web Service Archive Web Service SignatureCreation
isNonRepudiationReceiptRequired	Web Service Archive Web Service SignatureCreation
timeToAcknowledgeReceipt	BPEL `onAlarm`
timeToAcknowledgeAcceptance	BPEL `onAlarm`
isAuthorizationRequired	Web Service AuthorizationCheck
retryCount	BPEL `while`
isGuaranteedDeliveryRequired	WS-ReliableMessaging
hasLegalIntent	defined in terms of other QoS attributes
isConcurrent	not relevant
timeToPerform	BPEL `onAlarm`

Table 4.1: ebBP QoS attributes andand realization strategies

4.2 Design of WS-Interfaces

In order to establish a fully machine readable communication with XML documents between two or more partners naming conventions and common message formats have to be defined. This section covers all aspects concerning standards and interfaces of all kinds of communication in this work.

4.2.1 Design of the Messages

In this project, it is necessary to communicate meta-information such as the `MessageType` or the `MessageID` in addition to the contents of the business document. Hence, a suitable place to put these information within the transmitted document has to be found. Considering the effort for sending extra messages and issues concerning correlation as well as failure handling it is decided to transmit meta data together with the business document in a single message. So, the so-called `MetaBlock` carrying relevant meta data is introduced as a non omissible part of a message.

The idea to use the SOAP message headers in a way similar to most WS-* standards had to be abandoned due to the tested BPEL engines. None of them permits access to the SOAP message headers. Thus, in every transmitted message the meta-information is placed as `MetaBlock` in the SOAP message body, just before the actual body of the message containing the business information. This `MetaBlock` contains several information about the transmitted message. These are:

- UUID: The unique universal identifier of a concrete execution of a profile. The UUID is common to all BPEL processes used within the context of a concrete profile execution.
- RecursionCount: This element determines the depth of recursive invocation and is responsible for a unique identification of recursive processes. The recursion count follows the pattern `(Type)Nr.*` .
- ProfileID: Identifies the NES profile in which this message is used.
- MessageType: Specifies the type of the message, e.g., Order or Invoice.
- MessageID: The identifier of the message, starting with 1, incremented with each message.
- IsPositiveResponse: States whether the business document encapsulated in the body is a positive response or not in terms of the corresponding ebBP model.
- Signature: The tag including the signature, if present.

```
1 <xsd:element name="MetaBlock" type="MetaBlockType" />
2 <xsd:complexType name="MetaBlockType">
3    <xsd:sequence>
4       <xsd:element name="UUID" type="UUIDtype" minOccurs="1" maxOccurs="1"/>
5       <xsd:element name="RecursionCount" type="xsd:string" minOccurs="1" maxOccurs="1"/>
```

```
6       <xsd:element name="BCRolenameID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
7       <xsd:element name="ProfileID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
8       <xsd:element name="MessageType" type="xsd:string" minOccurs="1" maxOccurs="1" />
9       <xsd:element name="MessageID" type="xsd:int"  minOccurs="1" maxOccurs="1" />
10      <xsd:element name="IsPositiveResponse" type="xsd:boolean" minOccurs="0" maxOccurs="1"
            />
11      <xsd:element ref="ds:Signature" minOccurs="0" maxOccurs="1"/>
12    </xsd:sequence>
13 </xsd:complexType>
14 <xsd:simpleType name="UUIDtype">
15    <xsd:restriction base="xsd:string">
16       <xsd:pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
17    </xsd:restriction>
18 </xsd:simpleType>
```

Listing 4.6: The structure of the MetaBlock type

The MetaBlock is a part of every message sent within a NES profile in this project. All in all, there are three types of standard messages used in this project for different purposes. These are:

- **StandardMessage** (see listing 4.7) for transmitting arbitrary XML content in the body of the standardMessage, which is specified as `xsd:any`. This type is used to send, e.g., business documents like invoices or orders.

```
1  <xsd:element name="standardMessage" type="tns:standardMessageType" />
2  <xsd:complexType name="standardMessageType">
3     <xsd:sequence>
4        <xsd:element ref="mb:MetaBlock" />
5        <xsd:element name="body" type="tns:bodyType" />
6     </xsd:sequence>
7  </xsd:complexType>
8  <xsd:complexType name="bodyType">
9     <xsd:sequence>
10       <xsd:any processContents="strict" namespace="##any" />
11    </xsd:sequence>
12 </xsd:complexType>
```

Listing 4.7: The structure of the StandardMessageType

- **StandardMessagePlusBoolean** (see listing 4.8), which includes a `xsd:boolean` in the body tag instead of the `xsd:any` of the common standardMessage. The main purpose of this message type is to send validation responses (Web services SignatureValidation, SchematronValidation, XSDValidation, AuthorizationCheck).

```
1  <xsd:element name="standardMessagePlusBoolean" type="
       tns:standardMessagePlusBooleanType" />
2  <xsd:complexType name="standardMessagePlusBooleanType">
3     <xsd:sequence>
4        <xsd:element ref="mb:MetaBlock" />
5        <xsd:element name="body" type="tns:bodyType" />
6     </xsd:sequence>
7  </xsd:complexType>
8  <xsd:complexType name="bodyType">
9     <xsd:sequence>
10       <xsd:element name="validated" type="xsd:boolean" minOccurs="1" maxOccurs="1" />
11    </xsd:sequence>
```

```
12 </xsd:complexType>
```

Listing 4.8: The structure of the StandardMessagePlusBoolean type

- **StandardMessagePlusString** (see listing 4.9) is almost the same message type as the StandardMessagePlusBoolean, but it has an `xsd:string` in place of the `xsd:boolean` to send user names or other short texts in messages. Among others, it is needed for the AuthorizationCheck Web service.

```
1  <xsd:element name="standardMessagePlusString" type="tns:standardMessagePlusStringType
       " />
2  <xsd:complexType name="standardMessagePlusStringType">
3     <xsd:sequence>
4        <xsd:element ref="mb:MetaBlock" />
5        <xsd:element name="body" type="tns:bodyType" />
6     </xsd:sequence>
7  </xsd:complexType>
8  <xsd:complexType name="bodyType">
9     <xsd:sequence>
10       <xsd:element name="additional_information" type="xsd:string" />
11    </xsd:sequence>
12 </xsd:complexType>
```

Listing 4.9: The structure of the StandardMessagePlusString type

In addition to these message definitions, for some Web services there are particular solutions for very special messages, e.g., the `ExpressionEvaluationRequestMessage` for the ExpressionEvaluation service. These will be discussed in the corresponding sections.

4.2.2 The Correlation Set

Using a BPEL Business Process means using instances of the respective process specification. Clearly, it is possible that more than one instance of a process specification is running at the same time within the same engine instance. Due to this fact, there is a need for a mechanism to direct messages to the matching instances. Within BPEL, this mechanism is realized by the construct of Correlation Sets. A Correlation Set is a set of properties that are related with message attributes. A Correlation Set is built only from attributes that appear in all possible messages. The most important characteristic of such a set is uniqueness. This means a concrete composition of instances of all properties of the set identifies exactly one process instance. If all values of the correlating attributes are available, the Correlation Set can be initiated. This means these attributes have to retain their values during the whole process. If a message's Correlation Set is not initiated, the message has to be transmitted to all processes the Correlation Set matches. For more information see [JEA+07].

In this work, as in most BPEL projects, it is possible that a company has multiple active processes at the same time of the same type with multiple partners. Thus, Correlation Sets are needed. Below, the usage and application of this construct within this project is described in more detail.

For the Correlation Set some attributes of the Meta Block, which is common to all messages, are chosen to build an unique identification of process instances. The following listing shows the elements of the Meta Block used for the Correlation Set:

- `uuid:` As mentioned above, the UUID identifies a collaboration instance including all recursive processes at each party. With this attribute, a unique identification of the process execution is given.
- `roleNameID:` This attribute specifies the executing party of the process instance.
- `profileID:` This attribute specifies the profile performed by the process instance.
- `recCount:` Because each process instance can have multiple recursive calls, it is necessary to distinguish between the instances at each recursive depth. For this purpose, this attribute is used therefor.

Listing 4.10 shows the definition of the properties of the Correlation Set. This definition has to be done for each WSDL interface. Because of the large number of interfaces and the fact that these properties have to be used in each file, they are defined once within a separate WSDL file and reused by each WSDL interface by using an XML import statement. Each property definition is defined within a `property` element of the corresponding namespace by a name and type which has to be an XML Schema Simple Type.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions
3          name="Correlation"
4          targetNamespace="http://lspi.wiai.uni-bamberg.de/ss08/pi3/b2bi/wsdl/bpel/
               Correlation"
5          xmlns="http://schemas.xmlsoap.org/wsdl/"
6          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7          xmlns:tns="http://lspi.wiai.uni-bamberg.de/ss08/pi3/b2bi/wsdl/bpel/Correlation"
8          xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop">
9
10         <vprop:property
11                 name="prop_roleNameID"
12                 type="xsd:string" />
13         <vprop:property
14                 name="prop_profileID"
15                 type="xsd:string" />
16         <vprop:property
17                 name="prop_recCount"
18                 type="xsd:string" />
19         <vprop:property
20                 name="prop_uuid"
21                 type="xsd:string" />
22
23 </definitions>
```

Listing 4.10: Definition of properties for a Correlation Set within a WSDL file

After definition of the properties as shown in listing 4.10 the properties have to be associated with the attributes of the various messages defined within a WSDL interface. Listing 4.11 gives an example of the mentioned association for one message. For this, each property is associated with a query on the corresponding attribute of the specified message. Definitions for other messages only differ in attributes `messageType` and `part`.

```
...
        <vprop:propertyAlias propertyName="cor:prop_uuid"
                        messageType="tns:Backend2BPELNoQosServiceUUIDRequestMessage" part="
                            standardMessage">
                                <vprop:query>mb:MetaBlock/mb:UUID</vprop:query>
        </vprop:propertyAlias>

        <vprop:propertyAlias propertyName="cor:prop_profileID"
                        messageType="tns:Backend2BPELNoQosServiceUUIDRequestMessage" part="
                            standardMessage">
                                <vprop:query>mb:MetaBlock/mb:ProfileID</vprop:query>
        </vprop:propertyAlias>

        <vprop:propertyAlias propertyName="cor:prop_roleNameID"
                        messageType="tns:Backend2BPELNoQosServiceUUIDRequestMessage" part="
                            standardMessage">
                                <vprop:query>mb:MetaBlock/mb:BCRolenameID</vprop:query>
        </vprop:propertyAlias>

        <vprop:propertyAlias propertyName="cor:prop_recCount"
                        messageType="tns:Backend2BPELNoQosServiceUUIDRequestMessage" part="
                            standardMessage">
                                <vprop:query>mb:MetaBlock/mb:RecursionCount</vprop:query>
        </vprop:propertyAlias>
...
```

Listing 4.11: Usage and association of correlation set properties

Finally, the Correlation Set has to be specified within the BPEL process definition. Each BPEL file of this work defines it as shown in listing 4.12. The set is identified by a name and uses the `properties` attribute for correlation with the properties defined within the WSDL interfaces. These attributes, in essence, determine which property is used for this concrete Correlation Set.

```
...
        <correlationSets>
          <correlationSet xmlns:ns2="http://lspi.wiai.uni-bamberg.de/ss08/pi3/b2bi/wsdl/
              bpel/Correlation"
                name="process_message_correlation_set"
                properties="ns2:prop_uuid ns2:prop_roleNameID ns2:prop_profileID
                    ns2:prop_recCount"/>
        </correlationSets>
...
```

Listing 4.12: Definition of a Correlation Set within a BPEL file

Besides all specifications as mentioned above a Correlation Set has to be initiated. In this work, this is done in context of the generation and distribution of the UUID.

4.2.3 Naming Conventions of the WSDLs

Due to the fact that the Web service interfaces which consist of the WSDL file and the used XSD files are needed for at least two, sometimes three of this project's software components it is necessary to install policies for how to name the WSDL files, XSD files and related files. Thus, the following naming conventions are introduced.

First, the naming conventions of the WSDL files are described. The name of WSDL files and the name of corresponding Web services is computed by the following rules:

- Type: { BPEL2BPEL | BPEL2Service | BPEL2Backend | Backend2BPEL }
- Only for type BPEL2Service: { Archive | SignatureCreation | SignatureCheck | ValidationXSD | ValidationSchematron | UUID | ExpressionEvaluation }
- Security: { NoQos | Ssl | WsSecurity | SslWsSecurity }
- Reliability: { RM } or none
- Only for type BPEL2BPEL: { <roleId of the ebBP process> }
- Postfix: Service

As as result, e.g., the WSDL provided by the backend that has no security and no reliability configured is going to be named `BPEL2BackendNoQosService`.

Second, the XSD files have the requirement to employ the target namespace `http://lspi.wiai.uni-bamberg.de/ss08/pi3/b2bi/schema/{XSD_FILE_NAME}`.

4.2.4 Web Service Interfaces

This section describes the structure of the WSDL file contents and in particular how the application of QoS technologies can be requested from the openESB BPEL platform using WS-Policy assertions.

4.2.4.1 General Structure

All WSDL files can be divided in an interface definition and an implementation part. The interface definition part describes data types, messages and port types. The implementation part provides information where to find and how to call a service. Therefore it includes the bindings, ports and services.

In this work, all WSDL files employ this given structure:

Interface definition part: In the interface definition part no data types (XML tag `<types>`) are directly defined but the `StandardMessage`, the `MetaBlock` and the *XML digital signature* XSD files are imported. If the service uses more "message types" than the `StandardMessage`, these types, e.g., `StandardMessagePlusBoolean`, also must be imported in this section of the WSDL.
Most services only provide one or two messages: a receive of a `StandardMessage` and sometimes a reply. So, the definition of messages (XML tag `<message>`) is rather short. An example of a message definition is provided in listing 4.13. The only important issue here is

to respect the naming conventions: the message name must be of the prementioned format - “BPEL2BackendSslWsSecurityServiceReceiveMessage” in the provided example listing.
The last section of the abstract part are port types (XML tag `<portTypes>`) which combine messages to operations and operations to a port type. Here most of the WSDLs only have one single port type with a single operation: Receiving a message of the message type specified before - containing a StandardMessage. In most cases this a single one-way operation but some of the supporting Web services do also use a request-response operation. In listing 4.14 the port type of the previous example “BPEL2BackendSslWsSecurityService” is shown. Naming conventions must be respected here too.

```
...
  <message name="BPEL2BackendSslWsSecurityServiceReceiveMessage">
    <part name="standardMessage" element="stm:standardMessage"/>
  </message>
...
```

Listing 4.13: WSDL message definition for a BPEL2Backend service

```
...
  <portType name="BPEL2BackendSslWsSecurityServicePortType">
    <operation name="BPEL2BackendSslWsSecurityServiceReceive">
      <input message="tns:BPEL2BackendSslWsSecurityServiceReceiveMessage"/>
    </operation>
  </portType>
...
```

Listing 4.14: WSDL port type definition for a BPEL2Backend service

Implementation Part: The implementation part begins with the definition of bindings (XML tag `<binding>`). A binding enriches a port type by adding concrete information about the used protocol. In this work SOAP with transport over HTTP is always chosen. So, some SOAP specific definitions have to be added (See listing 4.15; for more information about the PolicyReference see the next section 4.2.4.2).
Last, the service is defined with the tag `<service>`: Herein, a port is defined which includes the concrete address where the service can be found. In our case the address always is a HTTP(S) URI (Example provided in listing 4.16).

```
...
  <binding name="BPEL2BackendSslWsSecurityServicePortBinding" type="
      tns:BPEL2BackendSslWsSecurityServicePortType">
    <wsp:PolicyReference URI="#BPEL2BackendSslWsSecurityServicePortBindingPolicy" />
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="BPEL2BackendSslWsSecurityServiceReceive">
      <soap:operation soapAction=""/>
      <input>
        <wsp:PolicyReference URI="#
            BPEL2BackendSslWsSecurityServicePortBinding_receiveMessage_Input_Policy" />
        <soap:body use="literal"/>
      </input>
    </operation>
  </binding>
...
```

Listing 4.15: WSDL binding definition for a BPEL2Backend service

```
1 ...
2   <service name="BPEL2BackendSslWsSecurityService">
3     <port name="BPEL2BackendSslWsSecurityServicePort" binding="
          tns:BPEL2BackendSslWsSecurityServicePortBinding">
4       <soap:address location="https://localhost:8181/BPEL2BackendSslWsSecurityService/
            BPEL2BackendSslWsSecurityService?wsdl"/>
5
6     </port>
7   </service>
8 ...
```

Listing 4.16: WSDL service definition for a BPEL2Backend service

Correlation Set: All the WSDL files which are used to communicate with the backend system include the information for the correlation set as described in section 4.2.2.

4.2.4.2 Various QoS Combinations

As described in section 4.1.2 some of the needed QoS features are realized by using WS-* standards. The WS-* standards used in this project are implemented in the Metro WS-Stack of GlassFish. In order to activate the required features, the WSDL files must include WS-Policy declarations to command the application server which standard to use.
In this project, the following three different standards have to be activated by WS-Policy descriptions[1]:

- SSL to guarantee secure delivery on the network layer (realizes `isAuthenticated`, `isConfidential` and `isTamperDetectable` in case the attribute is set to `transient`)
- WS-Security to realize `isConfidential` in case the attribute is set to `persistent`
- WS-ReliableMessaging to realize reliable communication (`isGuaranteedDeliveryRequired`)

These three features can be combined arbitrarily - so there exist the following eight combinations:

- noQos: all features deactivated - no WS-Policy declaration needed
- Ssl: To activate SSL, first, a WS-Policy assertion has to be included and referenced in the WSDL files (See: Listing 4.17) and, second, also the protocol in the concrete part of the WSDL has to be changed to "`https://`".
- WS-Security: To activate WS-Security a WS-Policy statement is also needed which can be found in Listing 4.18 and 4.19.
- RM: The WS-Policy assertion to use reliable messaging is shown in Listing 4.20

[1] All of the following three standards (WS-Security, SSL and WS-ReliableMessaging) need WS-Addressing as support to fulfill their function - so in the WS-Policy listings WS-Addressing is always enabled, too.

- SslWsSecurity (See paragraph “WS-Policy Combinations of WS-* standards”)
- SslRM (See paragraph “WS-Policy Combinations of WS-* standards”)
- WsSecurityRM (See paragraph “WS-Policy Combinations of WS-* standards”)
- SslWsSecurityRM: all three features have to be combined (See paragraph “WS-Policy Combinations of WS-* standards”)

```
...
  <wsp:Policy wsu:Id="BPEL2BackendSslServicePortBindingPolicy">
        <wsp:ExactlyOne>
            <wsp:All>
                <wsaws:UsingAddressing xmlns:wsaws="http://www.w3.org/2006/05/addressing/
                    wsdl"/>
                <sp:TransportBinding>
                    <wsp:Policy>
                        <sp:TransportToken>
                            <wsp:Policy>
                                <sp:HttpsToken RequireClientCertificate="false"/>
                            </wsp:Policy>
                        </sp:TransportToken>
                        <sp:Layout>
                            <wsp:Policy>
                                <sp:Lax/>
                            </wsp:Policy>
                        </sp:Layout>
                        <sp:IncludeTimestamp/>
                        <sp:AlgorithmSuite>
                            <wsp:Policy>
                                <sp:Basic128/>
                            </wsp:Policy>
                        </sp:AlgorithmSuite>
                    </wsp:Policy>
                </sp:TransportBinding>
                <sp:Wss10/>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
 ...
```

Listing 4.17: WS-Policy fragment for activating SSL

```
...
    <wsp:Policy wsu:Id="BPEL2BackendWsSecurityServicePortBindingPolicy">
        <wsp:ExactlyOne>
            <wsp:All>
                <wsaws:UsingAddressing xmlns:wsaws="http://www.w3.org/2006/05/addressing/
                    wsdl"/>
                <sp:SymmetricBinding>
                    <wsp:Policy>
                        <sp:ProtectionToken>
                            <wsp:Policy>
                                <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/
                                    ws/2005/07/securitypolicy/IncludeToken/Never">
                                    <wsp:Policy>
                                        <sp:WssX509V3Token10/>
                                    </wsp:Policy>
                                </sp:X509Token>
                            </wsp:Policy>
                        </sp:ProtectionToken>
                        <sp:Layout>
```

```
                            <wsp:Policy>
                                <sp:Strict/>
                            </wsp:Policy>
                        </sp:Layout>
                        <sp:IncludeTimestamp/>
                        <sp:OnlySignEntireHeadersAndBody/>
                        <sp:AlgorithmSuite>
                            <wsp:Policy>
                                <sp:Basic128/>
                            </wsp:Policy>
                        </sp:AlgorithmSuite>
                    </wsp:Policy>
                </sp:SymmetricBinding>
                <sp:Wss11>
                    <wsp:Policy>
                        <sp:MustSupportRefKeyIdentifier/>
                        <sp:MustSupportRefIssuerSerial/>
                        <sp:MustSupportRefThumbprint/>
                        <sp:MustSupportRefEncryptedKey/>
                    </wsp:Policy>
                </sp:Wss11>
                <sp:SignedSupportingTokens>
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken="http://schemas.xmlsoap.org/ws
                            /2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken10/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SignedSupportingTokens>
                <sc:KeyStore wspp:visibility="private" location="D:\Programme\AppServer\
                    domains\domain1\config\keystore.jks" type="JKS" storepass="XXX-PWD"
                    alias="xws-security-server"/>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    ...
```

Listing 4.18: WS-Policy fragment for activating WS-Security

```
...
    <wsp:Policy wsu:Id="
        BPEL2BackendWsSecurityServicePortBinding_receiveMessage_Input_Policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <sp:EncryptedParts>
                    <sp:Body/>
                </sp:EncryptedParts>
                <sp:SignedParts>
                    <sp:Body/>
                    <sp:Header Name="To" Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="From" Namespace="http://www.w3.org/2005/08/addressing"
                        />
                    <sp:Header Name="FaultTo" Namespace="http://www.w3.org/2005/08/
                        addressing"/>
                    <sp:Header Name="ReplyTo" Namespace="http://www.w3.org/2005/08/
                        addressing"/>
                    <sp:Header Name="MessageID" Namespace="http://www.w3.org/2005/08/
                        addressing"/>
                    <sp:Header Name="RelatesTo" Namespace="http://www.w3.org/2005/08/
                        addressing"/>
                    <sp:Header Name="Action" Namespace="http://www.w3.org/2005/08/
                        addressing"/>
```

```
17                      <sp:Header Name="AckRequested" Namespace="http://schemas.xmlsoap.org/ws
                            /2005/02/rm"/>
18                      <sp:Header Name="SequenceAcknowledgement" Namespace="http://schemas.
                            xmlsoap.org/ws/2005/02/rm"/>
19                      <sp:Header Name="Sequence" Namespace="http://schemas.xmlsoap.org/ws
                            /2005/02/rm"/>
20                  </sp:SignedParts>
21              </wsp:All>
22          </wsp:ExactlyOne>
23      </wsp:Policy>
24  ...
```

Listing 4.19: WS-Policy fragment to indicate which parts of a message should be signed or encrypted

```
1  ...
2          <wsp:Policy wsu:Id="BPEL2BPELNoQosRMServicePortBindingPolicy">
3                  <wsp:ExactlyOne>
4                          <wsp:All>
5                                  <wsaws:UsingAddressing
6                                          xmlns:wsaws="http://www.w3.org/2006/05/addressing/
                                               wsdl" />
7                                  <wsrm:RMAssertion />
8                          </wsp:All>
9                  </wsp:ExactlyOne>
10         </wsp:Policy>
11 ...
```

Listing 4.20: WS-Policy fragment to activate WS-ReliableMessaging

WS-Policy Combinations of WS-* standards: Policy assertions have to be combined if more then only one QoS feature should be activated. The combination of policy assertions is rather simple: they can be combined in one big declaration and then GlassFish activates all needed standards. The combination of WS-Security, SSL and reliable messaging is shown (simplified) in listing 4.21.

```
1  ...
2    <wsp:Policy wsu:Id="
         BPEL2BPELSslWsSecurityRMcb_profile3global_role_customerServiceBindingPolicy">
3                  <wsp:ExactlyOne>
4                          <wsp:All>
5                                  <wsaws:UsingAddressing xmlns:wsaws="http://www.w3.org
                                       /2006/05/addressing/wsdl"/>
6                                  <sp:TransportBinding>
7                                          <!--
8                                                  ... see SSL ...
9                                          -->
10                                 </sp:TransportBinding>
11                                 <sp:Wss10/>
12                                 <wsrm:RMAssertion/>
13                                 <sp:SymmetricBinding>
14                                         <!--
15                                                 ... see WS-Security ...
16                                         -->
17                                 </sp:SymmetricBinding>
18                                 <sp:Wss11>
19                                         <wsp:Policy>
20                                                 <sp:MustSupportRefKeyIdentifier/>
21                                                 <sp:MustSupportRefIssuerSerial/>
```

```
                                        <sp:MustSupportRefThumbprint/>
                                        <sp:MustSupportRefEncryptedKey/>
                                </wsp:Policy>
                        </sp:Wss11>
                        <sp:Trust10>
                                <wsp:Policy>
                                        <sp:RequireClientEntropy/>
                                        <sp:RequireServerEntropy/>
                                        <sp:MustSupportIssuedTokens/>
                                </wsp:Policy>
                        </sp:Trust10>
                        <sc:KeyStore type="JKS" storepass="XXX-PWD" wspp:visibility
                            ="private" alias="xws-security-server" location="D:\
                            GlassFishESB\glassfish\domains\domain1\config\keystore.
                            jks"/>
                </wsp:All>
        </wsp:ExactlyOne>
  </wsp:Policy>
...
```

Listing 4.21: Combination of WS-Security, SSL and WS-ReliableMessaging

KeyStore and TrustStore: To use WS-Security the certificates of the partners have to be saved in server TrustStore and own certificates have to be saved in the KeyStore. Depending on whether the WSDL is used on the client or on the server side it must reference the KeyStore or the TrustStore. On the server side a KeyStore declaration must be present (See listing 4.22). On the client side the TrustStore must be referenced as shown in listing 4.23 and moreover a CallbackHandler has to be defined.

```
...
  <wsp:Policy wsu:Id="
      BPEL2BPELSslWsSecurityRMcb_profile3global_role_customerServiceBindingPolicy">
        <wsp:ExactlyOne>
                <wsp:All>
                        ...
                        <sc:KeyStore type="JKS" storepass="XXX-PWD" wspp:visibility
                            ="private" alias="xws-security-server" location="D:\
                            GlassFishESB\glassfish\domains\domain1\config\keystore.
                            jks"/>
                        ...
                </wsp:All>
        </wsp:ExactlyOne>
  </wsp:Policy>
...
```

Listing 4.22: Server Side: KeyStore

```
...
  <wsp:Policy wsu:Id="
      BPEL2BPELSslWsSecurityRMcb_profile3global_role_customerServiceBindingPolicy">
        <wsp:ExactlyOne>
                <wsp:All>
                        ...
                        <sc:CallbackHandlerConfiguration wspp:visibility="private">
                                <sc:CallbackHandler default="XXX-USER" name="
                                    usernameHandler"/>
                                <sc:CallbackHandler default="XXX-PWD" name="
                                    passwordHandler"/>
                        </sc:CallbackHandlerConfiguration>
```

```
10                                  <sc:TrustStore wspp:visibility="private" location="D:\
                                        GlassFishESB\glassfish\domains\domain1\config\cacerts.
                                        jks" type="JKS" storepass="XXX-PWD" peeralias="xws-
                                        security-server"/>
11                                  ...
12                          </wsp:All>
13                  </wsp:ExactlyOne>
14          </wsp:Policy>
15  ...
```

Listing 4.23: Client Side: TrustStore and CallbackHandler

Policy referencing: Of course, the definition of policies itself is not enough to enable the needed functionality. The definition must be linked to the binding in the implementation part of a WSDL file. How this can be achieved has already been shown in listing 4.15 in the previous section. There, it is also shown that the message specific policies must be referenced by the concrete operations in the binding definition as well (also listing 4.15; line 8).

Remark: The use of single WS-* Standards and also the combinations of SSL/WS-RM and WS-Security/WS-RM is supported by NetBeans. So, the WS-Policy assertion shown can easily be generated by NetBeans. Only the combinations of WS-Security/SSL and activating all three QoS features is not directly supported by NetBeans and thus the WSDL must be customized by hand. For more details of using NetBeans to create and use "secure" and reliable services see, e.g., `http://www.netbeans.org/kb/docs/websvc/wsit.html` and the WSIT Tutorial[2].

[2]Available at: `http://java.sun.com/webservices/reference/tutorials/wsit/doc/index.html`

4.3 Architectures and Implementations

In this section, the architecture and technical details of the implementation, of the whole system and of all its components will be explained. In each subsection, the level of detail will increase step by step.

4.3.1 Overall Architecture

At first, the overall architecture is illustrated. As shown in figure 4.10, the architecture of this use case consists of the three autonomous components Translator, Backend and Web services. Each collaboration party's integration system is a composition of a BPEL orchestration, a Backend system and a number of Web services for realization of QoS features or as utilities. The Translator is used to transform the NES profiles, modeled as global ebBP choreographies, into local BPEL orchestrations and WSDL files for each participating party considering the specified QoS features and their realization strategies.

The Backend system is used for testing the generated processes and Web services by simulating a real Backend system at each party.

Both, Backend system and BPEL processes use Web services during the realization of QoS features. Each party has to implement its own Web services conforming to the WSDL interfaces generated by the Translator.

The entire interaction between the two parties' BPEL processes, between Backend system and corresponding BPEL process and with supporting Web services is based on the generated WSDL interfaces. This interaction is protected by use of WS-* standards or SSL/TLS, visualized as padlock in figure 4.10. The network connection between Backend system and Web services of a integration partner is assumed to be secure in terms of reliability and security (cf. page 90).

The architecture and implementation details of each of the three components is explained in the following sections.

4.3.2 Translator Architecture and Implementation

The translator is implemented by several interconnected components that subsequently transform an ebBP input process into an BPEL output process together with related WSDL interfaces. Thus, this section starts out with a description of the interplay between the translator components and proceeds with discussing the components individually.

4.3.2.1 General Architecture

The architecture of the Translator can be compared with Michael Porter's value chain (see figure 4.11). This metaphor has been used to create a common understanding of the underlying

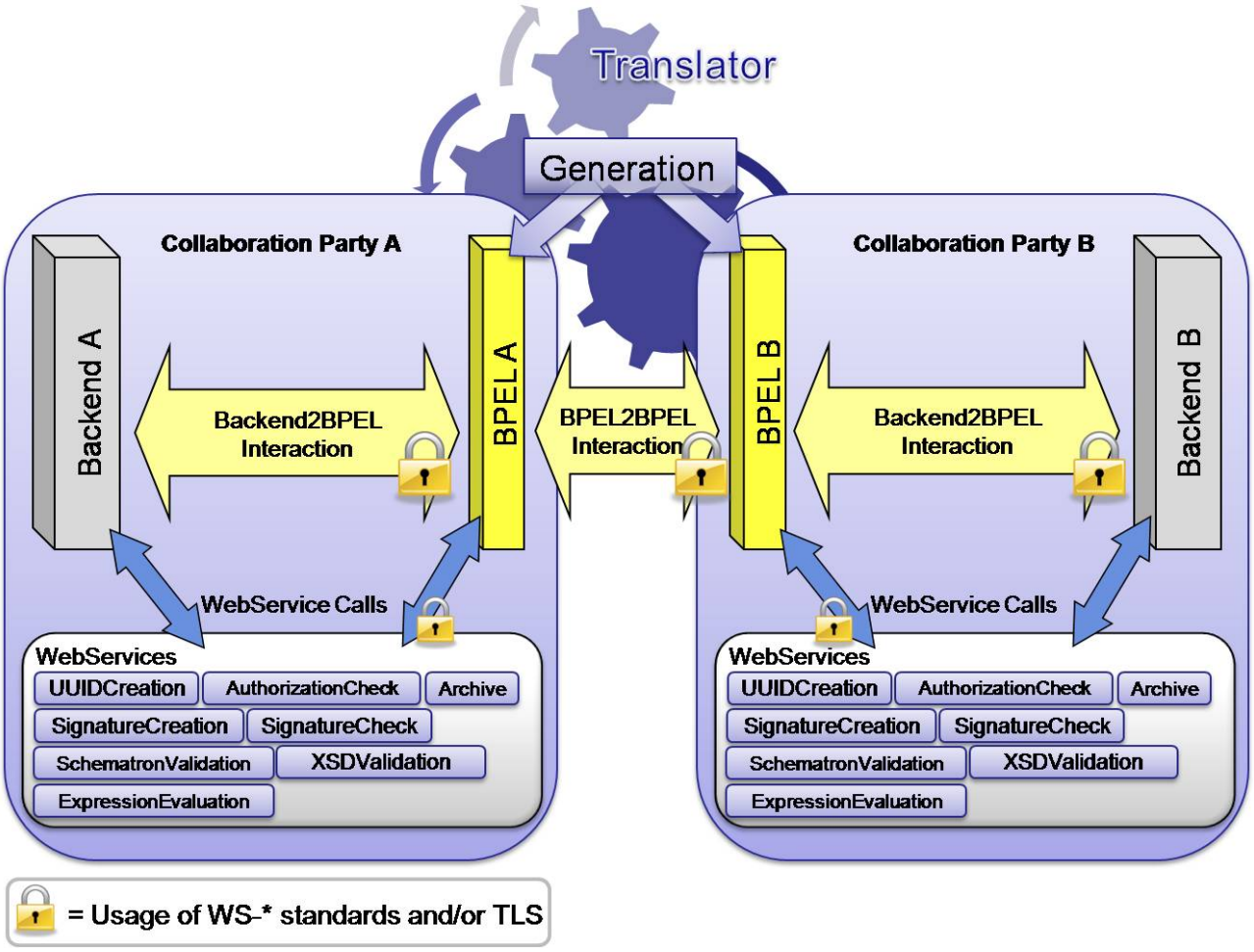


Figure 4.10: Overall B2Bi architecture

work flow and architecture of the translator.

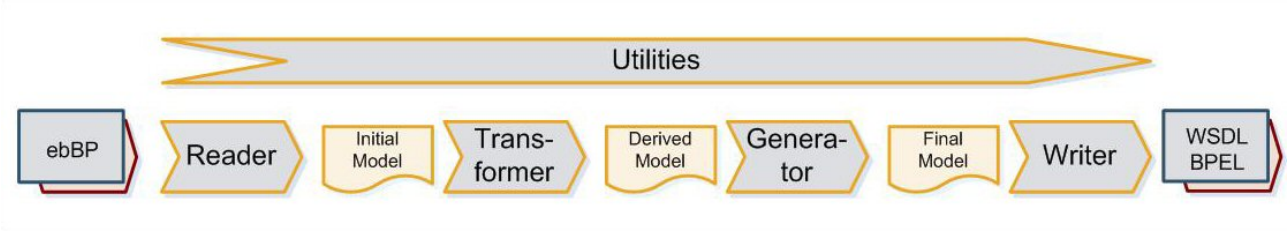


Figure 4.11: Architecture of translator as a Value Chain

The ebBP models of the NES are used as inputs. The function of the Reader is to read one of these profiles and to save it in a Java object structure called Initial Model which is described in detail below. This Initial Model is used by the next production step, the transformer, as input parameter. The purpose of the transformer is to filter the relevant information out of the Initial Model and to fill a model called Derived Model which is easier to handle for the next production steps and which is also described in detail below. The generation step can now use this Derived Model in the generation process, also described in detail below, which provides a Java object structure at its end. These objects represent the structure of all BPEL processes to be generated. This object structure is called Final Model and is used as input parameter for

the last production step which persists these BPEL processes to the file system.

The value chain also contains a secondary process called Utilities. These utilities are used by the primary production steps amongst others to fill WSDL templates or to load different properties for the generation process.

Initial Model The object structure of the Initial Model has been built using the JAXB binding compiler XJC in order to use the unmarshaling functionality of the JAXB 2.0 Binding Framework. The input of XJC has been the ebBP specification XML schema file `ebbp-2.0.4.xsd`.

Derived Model The Derived Model is a specialization of the Initial Model. It is reduced to the parts that are needed by the generator component for the mapping and additionally optimized for better processing. For example, the constructs join and fork which are not used at all in the ebBP modeling and are available in the Initial Model are not available in the Derived Model.

Final Model The object structure of the Initial Model has been built using the JAXB binding compiler XJC in order to use the marshaling functionality of the JAXB 2.0 Binding Framework which is described in section 2.6. The input of XJC has been the WS-BPEL specification XML schema file `ws-bpel_executable.xsd`.

4.3.2.2 Architecture and Implementation of the main components

For the main components, namely reader, writer, transformer and generator, special constructs and design principles are used.

Every component is abstracted by an interface that contains only one operation which is doing the job the component is responsible for. An instance of this interface can be obtained by a factory to hide the corresponding implementation for the component. The implementations of the mentioned components are described in the next sections.

Additionally, all components have a specific controller that implements the interface that can be obtained by the factories. This controller is in charge of the control flow in the component and delegates the flow to other parts to ensure the fulfillment of the contract of the interface.

4.3.2.3 Architecture and Implementation of Reader

The reader component handles the loading of the Initial Models. This is done by the following steps:

1. validate ebBP XML file
   The XML file is validated against the `ebBP.xsd` schema by the utilities component.

2. create the Initial Model
   Using a JAXB mapping, the initial model is created from the valid ebBP XML file by the utilities component. The functionality of JAXB is explained in detail in section 2.6.

As described in section 4.3.2.2 the main controller only delegates the work to other parts. In this case, these are the XML validation part and the XML reading part of the utilities component.

4.3.2.4 Architecture and Implementation of Transformer

The transformer component handles the transformation of the Initial Model into the more specialized Derived Model. The process can be divided into three steps. At first, all the elements that are used by the business collaboration and the basic business collaboration are created in the specific order:

1. Create all business signals.
2. Create all business documents.
3. Create all business transactions.
4. Create all basic business collaborations.

After these parts are created, the next two steps are only performed in the context of a single business collaboration. The second step is to create the main elements of a business collaboration. Advantageously, these elements can be created separately.

1. Create all business transaction activities.
2. Create all collaboration activities.
3. Create all final states.

In the third step all the created elements of a business collaboration are linked together.

1. Create the start element and link it to the first element.
2. Create the decision elements and set the to and from attributes.

To decouple the separate parts of the transformer the `Transformer` interface is introduced and used for delegation. This interface only expects the Initial Model which is the input and the Derived Model which is the output and is enriched by each call. Each component that implements one of the tasks mentioned above implements this interface, e.g., there are dedicated

classes for generating the representation of Business Collaborations or Business Transactions. A part of the Java class hierarchy that results from this concept is visualized in figure 4.12 using a UML class diagram. The control flow of the generation and hence the order of calling the respective `Transformer` classes roughly corresponds to the listings above.

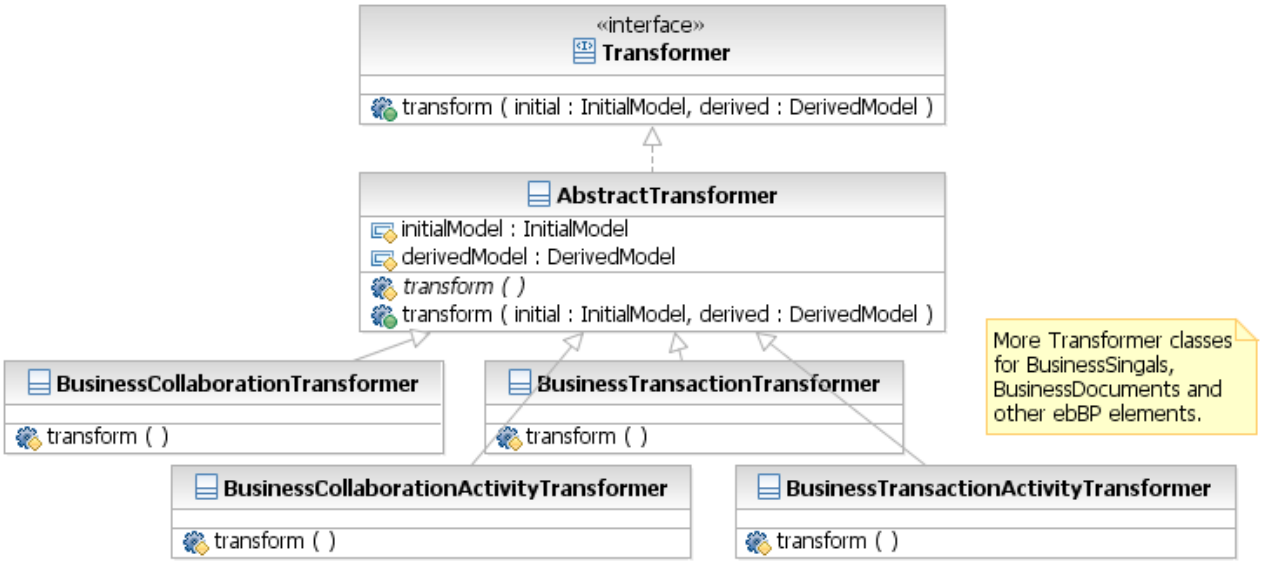


Figure 4.12: Part of the Transformer class hierarchy

4.3.2.5 Architecture and Implementation of Generator

The generator consists of two important parts, the flow and the WSDL component. The WSDL component is used by the flow component and because of this dependency the WSDL component is explained first.

wsdl The WSDL component creates WSDL files that are needed for the partner links in the BPEL process. Therefore, an API has been created to simplify the generation of WSDL files for the flow component. The interface the flow component uses for creating the WSDL interfaces only consists of three factories plus the WSDL generator interface.

- service WSDL generator factory
  Used for services like uuid service and others described in section 4.3.4.
- BPEL WSDL generator factory
  Used for BPEL to BPEL WSDLs for interprocess communication.
- backend WSDL generator factory
  Used for BPEL to backend or backend to BPEL WSDLs.

Each factory returns the same interface for each requested WSDL generator. As a result the API for the flow component is minimalistic and simple. So the creation of every WSDL file is

done with the same interface, however the implementation is abstracted due to the described factories above.

In order to generate a WSDL interface, the flow component calls one of the factories that then returns the WSDL generation interface with the requested implementation. Next, the interface is called to generate the WSDL interface. For this task, the following information is needed:

- security type
  Whether this interface should have none, ssl, ws security, or ssl and ws security enabled.
- reliability
  Whether this interface should be reliable or unreliable.
- side
  Whether this interface is used for a call or is provided for another part.

After the task is complete the caller retrieves an object that contains all relevant information about the result of the generation process, e.g., the path of the WSDL interface file and the paths of the XSD files that are referenced by the WSDL interface etc..

In the following, the implementation of the WSDL API is explained. The generation process is very similar for all generations and is centralized in one class which is configured by the property files `programm.properties`, `role.properties` and the information passed in with the generation call. Only the referenced XSD files can differ, and, therefore, for every WSDL type there is a class that contains this information and is asked during the process.

The steps of the generation of the WSDL interface occur in the following order:

1. Create the WSDL string generator.
2. Retrieve relevant information like the deployed url, the current role id, the template path and the output path.
3. OPTIONAL: Set trust store information by security type.
4. OPTIONAL: Set key store information by security type.
5. OPTIONAL: Copy the WSDL correlation interface.
6. Write the WSDL interface into a file.
7. Tidy the WSDL file .
8. Get the paths to the referenced XSD files and copy them to the output directory.
9. Validate the generated WSDL interface.

For the generation of the WSDL interfaces the Velocity[3] templating engine is used. Every type of WSDL interface is available at design time in form of a template that is injected with specific parameters in order to create a fully functional WSDL interface. The templates are stored in the file system and the location can be configured with the `programm.properties` file. Each WSDL file is injected with the following information:

- WSDL file name
  The name of the WSDL interface and therefore also the WSDL service name.
- key store
  See section 4.2.4.2.
- trust store
  See section 4.2.4.2.
- url
  The url the WSDL file is deployed to.
- side
  The side the WSDL file is used at; can be either the server side providing the interface for calls or simply the client side to call it.

The usage of the template engine is described below:

1. Retrieve the template path.
2. Create the template instance.
3. Collect the configuration information.
4. Inject the information into the template.
5. Get the output file path.
6. Write the template to the file system.

Note that, instead of writing directly to the file system as for the WSDL file, the trust store and key store are only created as strings in the translator and then further processed.

At the end of the WSDL generation process the files are created in a folder structure specified by the `programm.properties`. By default, the following structure is used:

WSDL Files and XSD files per WSDL:
`target/customer|supplier/wsdl/servicefolder`

servicefolder:
`backend2bpel/`

[3] `http://velocity.apache.org/`

```
bpel2backend/
bpel2bpel/tprocess_name/bc_role_name_id/
bpel2service/archive/
bpel2service/uuid/
bpel2service/signature/creation/
bpel2service/signature/check/
bpel2service/validation/xsd/
bpel2service/validation/schema/
bpel2service/authorization/
bpel2service/expressionevaluation/
```

flow The flow component is the heart of the translator. It controls the generation of the Final Model from the Derived Model by using the concepts of the mapping constructs. The component consists of two parts, namely the controller and the creator, which are explained in detail below.

controller The controller part controls the creation of the BPEL process in the Final Model. The mapping constructs that describe/specify the mapping from ebBP to BPEL have influenced this part in so far as every controller implements at least one mapping construct. For special cases two mapping constructs are consolidated in one controller for redundancy reduction.

There are two kinds of controller, the main flow controller and the helping flow controller. The main flow controllers are in charge of the mapping of the ebBP tags like `business collaboration` and `business transaction activity` which are the main components in the ebBP process. The helping flow controllers, however, are responsible for the additional constructs like UUID distribution which only has been introduced due to the lack of id synchronization. For each of the controller types there is a factory that creates them and returns always the same interface. This interface has only one method that expects as parameters the Derived Model, which is the input of the generation, the Final Model, which is the output, and, additionally, some temporary variables which are used for having a global shared state during the generation process to ease the generation of the Final Model. This interface simplifies the delegation of the flow to another controller. So if a controller is confronted with a tag that it cannot handle or is not responsible for the work is delegated to the next flow controller which is retrieved via the factories. As a result, the complexity for each controller is reduced because the work can simply be delegated and the other controller itself can delegate and so on and so forth. In contrast to that, the helping flow controllers only help the main flow controllers. Therefore, the delegation process is controlled by the main flow controllers because the helping flow controllers cannot create other controllers via factories.

The controllers themselves either execute the requesting or responding part of a BTA depending on the role in the current `business transaction activity` which is always determined ex ante. During their execution some other components are called like the *creator* component or the *bc name mapping* component. This is needed because every `business collaboration` is mapped to a specific text string that identifies the type of `business collaboration` in the BPEL process at run time. The mapping information is set in the `role.properties`, and, after

loading it, it can be easily accessed by the flow component via the bc name mapping.

creator In order to simplify the BPEL generation for the flow controller the *creator* component has been created. Its main responsibility is to provide methods that generate frequently used BPEL elements in order to reduce the failure rate and code redundancy. Not every BPEL element can be created with that component, but the most complex ones can comfortably be mapped.

To create a BPEL element, configuration parameters are passed to the relevant method. Enumerations are the most used data type for enhancing readability. The methods either return the created component or a result object that references all created BPEL elements during the call.

The BPEL elements that can be created with the creator component are listed below:

- Receive
- Invoke
- OnEvent
- Throw
- Copy
- Assign
- Sequence
- Scope
- Variables
- Variable
- If

Especially the creation of the Receive, Invoke and OnEvent is comfortable because the relevant WSDL files are created automatically in the background by the WSDL component which is called by the creator component.

The creation of the Copy element is also very powerful due to method overloading, e.g., a string can be copied to a variable just as well as a specific part that is selected via XPath from a complex variable can be copied to another specific part of another complex variable.

4.3.2.6 Architecture and Implementation of Writer

The writer component handles the writing of the actual BPEL XML files. For each process in the Final Model, the following steps are performed:

1. Determine the output folder and filename:
   Using the utilities component, the output folder and the filename together with several other properties are loaded.

2. Write the process to the file system:
   Using JAXB a BPEL file is created in the determined output folder with the computed filename using the utilities component. The functioning of JAXB is explained in section 2.6.

3. Format the BPEL file:
   Using the utilities component the BPEL file is formatted to guarantee better readability using indentations etc..

4. Clean the BPEL file:
   Some namespace issues are resolved using regular expressions.

5. Validate the BPEL file:
   Using the utilities component the BPEL file is validated against the BPEL XML schema definition.

As described in section 4.3.2.2, the main controller of the writer component only delegates the work to other parts, in this case the property loader part, the helping part, the XML validation and the XML writing part of the utilities component.

Output At the end of the complete process the BPEL files are created as well as the WSDL and XSD files that are referenced by the BPEL files are copied to the location specified in the `program.properties`.

The default output path can be expressed like this:
`target/customer|supplier/bpel/tprocess_name/`

4.3.2.7 Architecture and Implementation of Utilities

The utilities component consists of several independent parts that are used by the main components to accomplish their tasks.

Property Loader The property loader encapsulates property files and provides a simple API to access the property files.

XML Writer The XML writer enables writing BPEL models to XML files using JAXB.

XML Reader The XML reader enables reading ebBP files to Java classes using JAXB.

XML Validator The XML validator validates XML files against specific XSD schema files. The factory provides validators for ebBP, BPEL and WSDL files, however for WSDL files a mock object is used due to the usage of WSDL extensions which are not part of the XSD and result in an error if validated.

XML Namespace Extractor The XML namespace extractor's function is to retrieve the target namespace of XSD files using XPATH functionality.

Helper The helper contains several useful methods, e.g., tidying up an XML file, checking whether a file exists, loading an XML file into a string and a few more.

4.3.3 Backend Architecture and Implementation

This section describes the architecture and implementation of the *dummy* backend for testing the generated BPEL processes. This comprises a description of the application layers, technology specific modules and the functionality for handling NES profiles.

4.3.3.1 General Architecture and Implementation

The purpose of the Backend is to test and validate the functionality of the generated BPEL processes as well as the QoS-features. The Java Enterprise Edition (JEE)[4] has been selected as platform for implementing the Backend. This platform offers advanced technologies for implementing business components and Web based user interfaces and is supported by different vendors like Oracle, IBM or JBoss. Technically speaking, a major reason for choosing JEE is the availability of middleware services like as transaction management, naming, persistency, security and a standard format for packaging and deploying application modules. Note, that such services considerably simplify the development of enterprise level software. The GlassFish application server has been chosen as implementation of the JEE framework because it closely conforms to the technology specifications included in JEE. Further, as GlassFish together with open-ESB has been selected as execution platform for this project's BPEL processes also choosing GlassFish as application server for implementing the Backend promises advantages in terms of interoperability as well as development time.

One major problem determines the general architecture of the backend system to a large extent. The NES business documents are subsets of UBL business documents, but unfortunately specializations of the same UBL document differ from profile to profile, i.e., there are several profiles with XML schema definitions in the same namespace, but with different attributes. Thus, conflicts occur when putting all profile-specific, JAXB-generated business document classes in one classpath. For example an order.xsd of one NES profile has items with the attribute ItemDescription of type String, and another order.xsd from another profile has the same attribute, but of type List. To avoid this problem, profile-specific document handling is done in a separate EJB module with a separate business documents classes library for each profile as described in section 4.3.3.2. These profile modules cannot be integrated in one JEE Enterprise Application project (.ear), because this also means one classpath for all modules therein. The profile-independent components, however, are combined in a single Enterprise Application (*b2biTestToolEnterpriseApplication*). These are the *CommonModule*, the *ReceiverModule*, the web user interface *b2biTestToolEnterpriseApplication-war*, the *ArchiveService* and the *SignatureCreationService* as depicted in figure 4.13.

The architecture of the Enterprise Application can be divided into the three layers persistency, application and communication.

Persistency As the backend is mainly working as a message exchange system, it must be decided whether messages should be persisted permanently or just should be kept in volatile

[4] `http://jcp.org/en/jsr/detail?id=244`

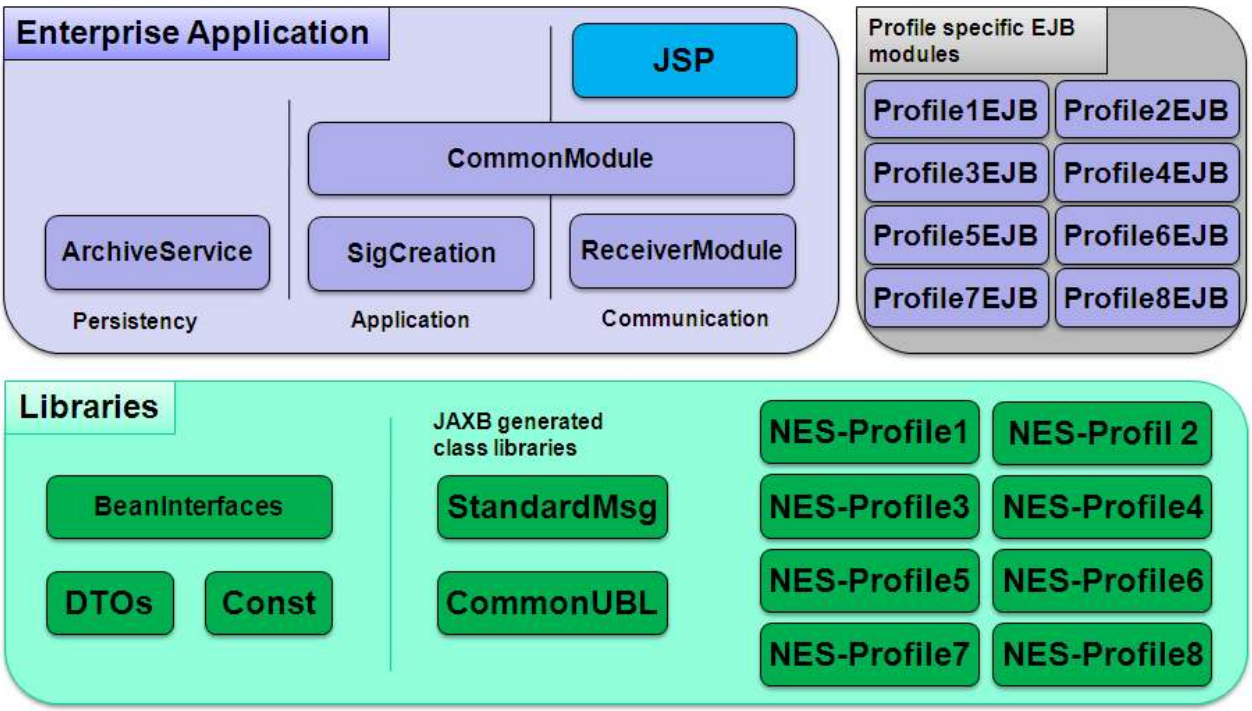


Figure 4.13: Components of the backend

memory. Since the backend is just a test tool, the latter would still be appropriate. Two reasonable arguments, however, speak in favor of the former:

1. Persisting all messages offers better test and debug support.
2. The QoS-feature *non-repudiation-required* needs backend functionality to persist the whole exchange of all business documents on both sides anyway, which is implemented as the so-called ArchiveService Web service. Thus, the implementation overhead is manageable.

The EJB module *ArchiveService* is the implementation of the web service *Archive* described in section 4.3.4.1. The only difference is that it does not provide any WSDL interfaces, but an EJB remote interface. It fulfills the function of a message inbox. Upon arrival of a new permissible message, it is persisted as an unhandled message to be picked up by the user interface.

Application This layer, in particular the *CommonModule*, provides functions for retrieving unhandled messages from the *ArchiveService* and creating and handling messages of types Application Response, Error and Protocol Success as these are commonly used by every profile. The *MessageHandlingBean* also fulfills communication duties. The *SignatureCreationService* is another part of this layer. Its implementation is explained in section 4.3.4.4. For sending a digitally signed business document the backend uses this service to create a signature.

Communication For communicating with the BPEL process there is the *CommonModule* for sending business documents and other messages via the appropriate QoS-enabled WSDL

port and the *ReceiverModule* which provides WSDL ports for the BPEL process to send its messages to. For a detailed description of the message flow see 4.3.3.2.

Communication with the user is realized as a web application based on Java Server Pages (JSP) and the Java Server Faces (JSF) framework. Developing a stand-alone desktop application was discarded in favor of a web-based user interface because of application distribution benefits and good development support by the NetBeans IDE. Navigating through the application is intuitive: On the start page the user selects the NES profile and on the next page she selects her role (Customer or Supplier). Then, she can see a list with all unhandled messages and a button for starting a new process in case the role allows it. Handling a message takes the user to the *DetailView* page with content that depends on the selected message type. Each JSP or JSPF is connected to a backing bean which contains the server-side code. This bean implements life cycle methods defined by the JSF API, e.g., prerender() to update data-bound web controls before rendering takes place. Session handling is done via the *SessionBeanWeb*. It extends the AbstractSessionBean, also part of the JSF API, to support easy session scope data handling. It contains properties to represent cached data that should be made available across multiple HTTP requests for an individual user. It administers the data provider for the table of unhandled messages in *ActiveProcesses*. Also, it saves the selected role, profile, message handle and active view. To send a business document, the user needs to go to the detail view of the message type READY_TO_RECEIVE_X (with X being the business document to send; note that READY_TO_RECEIVE means the BPEL process is ready to receive the message from the backend and subsequently send it to the partner process). Here she can choose from a list of dummy documents and the QoS attributes she wants to enable: transient, persistent and/or signing the message.

Libraries As mentioned earlier, the NES derivations of the UBL business documents have different content in the same namespace. Considering this, there is a library with JAXB generated code for each profile as depicted on the bottom-right of figure 4.13. The *CommonUBL* and *StandardMessage* libraries also belong to the category of generated code. While *CommonUBL* holds the APPLICATION_RESPONSE message type and other commonly used definitions, the content of the *StandardMessage* library was generated from the StandardMessageType.xsd with the MetaBlock.xsd designed for this project. For communicating with the user interface, a library with Data Transfer Objects (DTO) was created to represent the minimum information about the messages and business documents needed using simple Java data types as members. The *BeanInterfaces* are the EJB remote interfaces needed by other beans (see next paragraph).

Interfaces Figure 4.14 shows how the EJBs are interconnected. When a message arrives, the *ReceiverModule* sends it to the *MessageDelegateBean* of the appropriate profile module (the profile of the message is indicated in the MetaBlock). There, the message is checked and handed over to the *CommonModule* for persisting it. When a message is sent by the user, it is checked and marshaled in the *ClientHandlerBean* of its profile module and handed over to the *CommonModule* which sends the message via its references to the process's WSDL port.

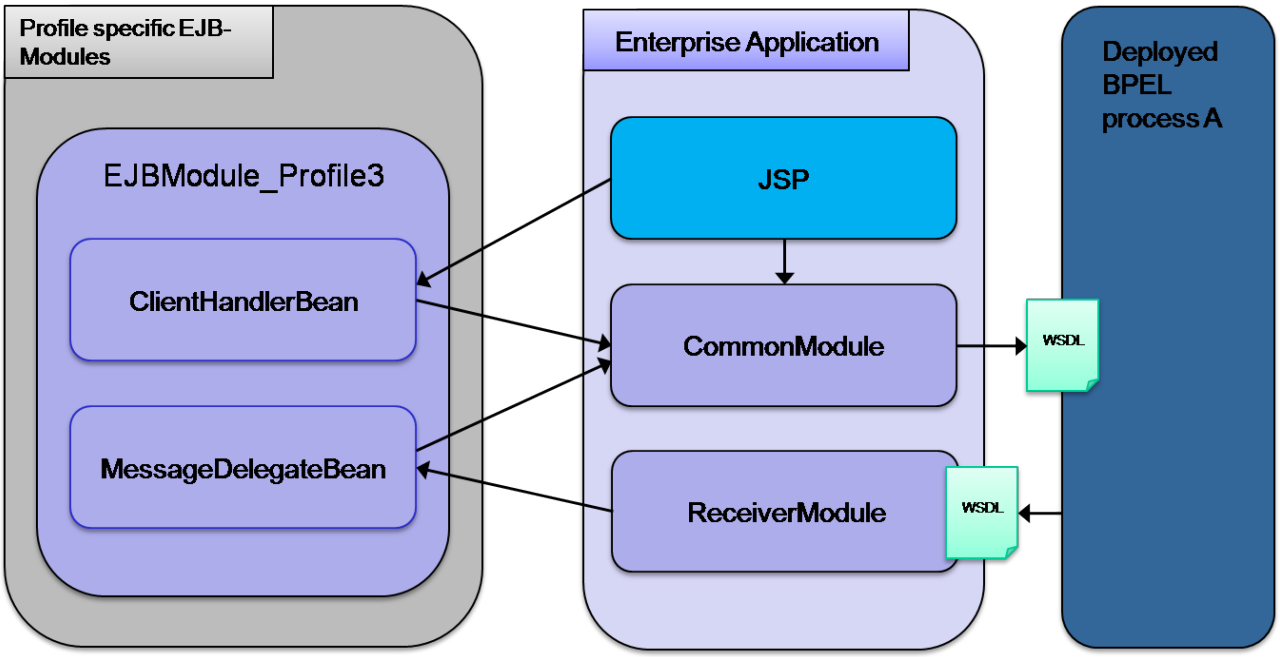


Figure 4.14: Interfaces between components of the backend

4.3.3.2 Architecture and Implementation of the Profile Handlers

Together with the Enterprise Application *b2biTestToolEnterpriseApplication*, the eight EJB modules for handling the profiles form the backend system.

The message flow of a message is shown in figure 4.15, as an example for the role „Customer“ in profile 3. In this profile („Basic Order only“) the customer starts the process. By sending the first message, a UUID_REQUEST, the BPEL engine starts a new process instance. This process requests a new UUID from the appropriate web service (see 4.3.4.6) and distributes it to the customer and supplier, the UUID_RESPONSE. Next a READY_TO_RECEIVE_ORDER arrives, which signals that the process is ready to receive the business document of type „Order“. By handling this message the user selects a dummy order from a list and sends it with the selected QoS options. The QoS selection must exactly match the QoS attributes the BPEL process was generated with.

Each profile handler module has a dummy content generator for creating test content for the profile-specific business documents. For example, an order is filled with order lines and random prices, an issue date, the sender host name etc.

Before sending and after receiving a message, a state machine checks the message type against the current process state. This is done to prevent wrong system states due to transfer errors that are common in distributed systems such as overtaking messages. If a received message is not allowed, it will be put into a Java Message Service (JMS) queue as a JMS ObjectMessage (this is done in the *ReceiverModule*). After receiving the next message, the queue is checked for waiting messages. If there is a permissible message for this process instance, it will be saved, if not, it will be put into the queue again until its maximum retry count is reached.

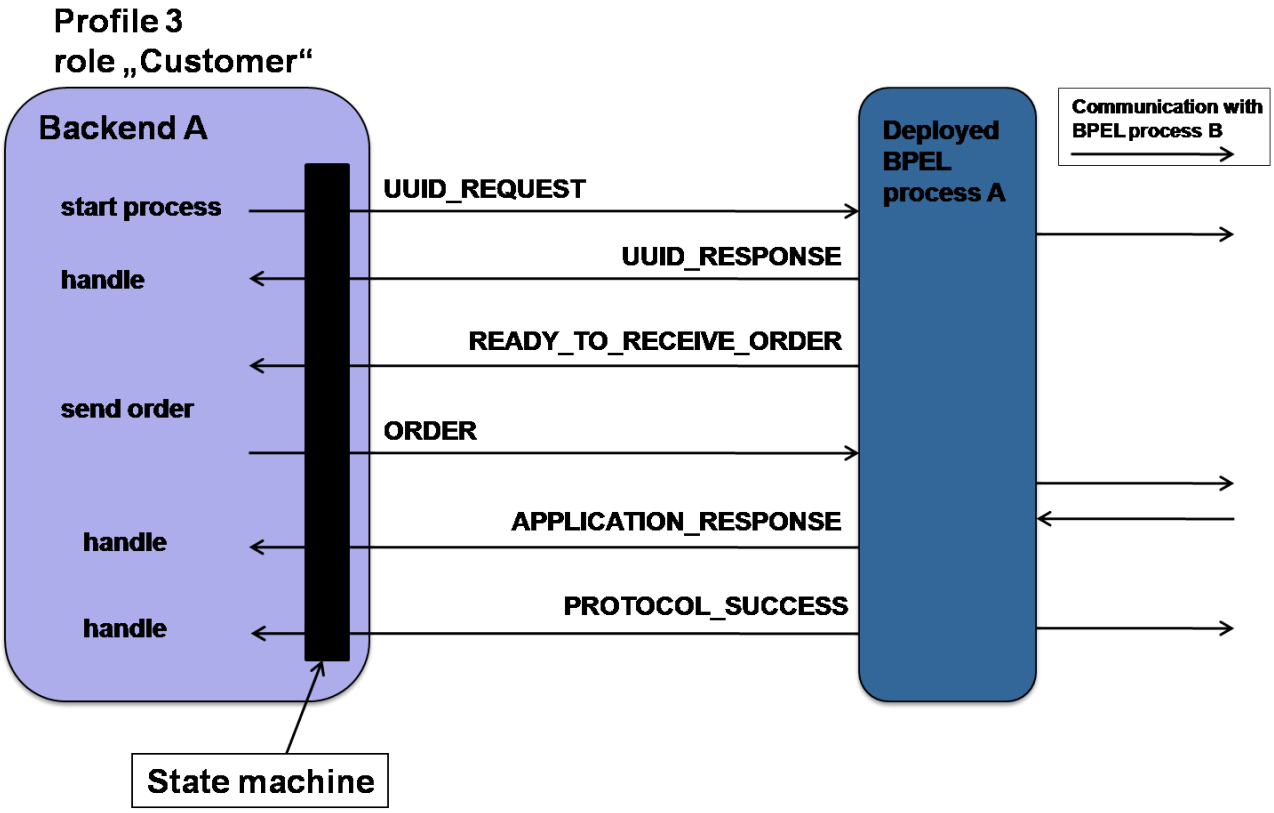


Figure 4.15: Message flow between backend and control process

State machine In order to implement a state machine in Java, the „State“ pattern, cf. [EGV04] pp. 305-313, offers an elegant solution. Figure 4.16 illustrates the pattern in a UML class diagram: The abstract *State* class provides all state transition methods implemented simply by throwing a *MessageNotAllowedException*. Concrete state classes extend this abstract class and override only the methods that represent allowed transitions in this state. These overridden methods change the state to the next. If a method is called that is not overridden by the current state, the code from the abstract class is executed which throws the mentioned exception. A *StateMachine* class holds a reference to the current state and realizes the mapping of message types to state transitions. The JEE Application Server environment allows for an easy way to persist the state for a process instance: Each state is a Java Persistence API (JPA) entity. The abstract state class defines the table and differentiates between the current extending state via the *@DiscriminatorColumn* annotation (see listing 4.24) which automatically uses the name of the class and puts it in the STATE column. As the primary key of this table the process's UUID is specified.

Listing 4.24: JPA annotation at the abstract *State* class

```
1 @Entity
2 @Table(name = "StatesProfile3")
3 @DiscriminatorColumn(name = "STATE", discriminatorType =
    DiscriminatorType.STRING, length = 100)
4 public abstract class StateEntity implements Serializable {
5         ...
6 }
```

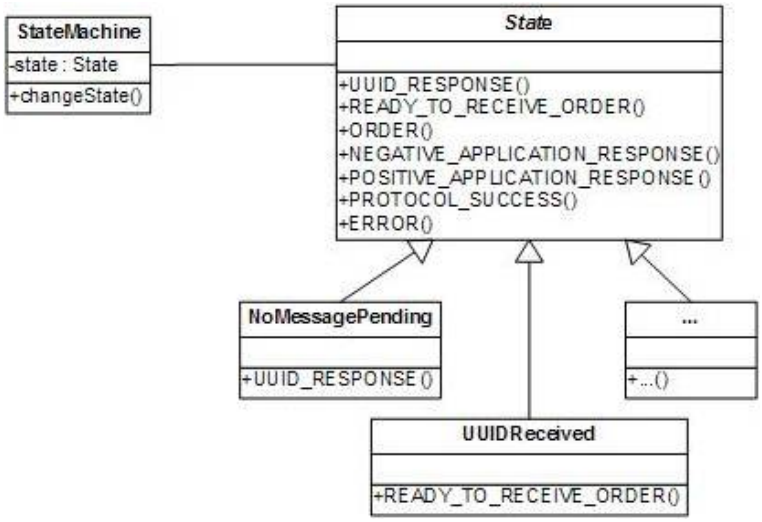


Figure 4.16: Implementation of the „State“ pattern

After the *Supplier* responded to the ORDER with an APPLICATION_RESPONSE, the BPEL process sends a PROTOCOL_SUCCESS which indicates that no protocol failure occurred. The content of the APPLICATION_RESPONSE message tells whether the outcome of this collaboration was a business success or failure. With this last message the process instance is terminated. For another illustration of the state machine see the state graph (figure 4.17) for the customer of profile 3.

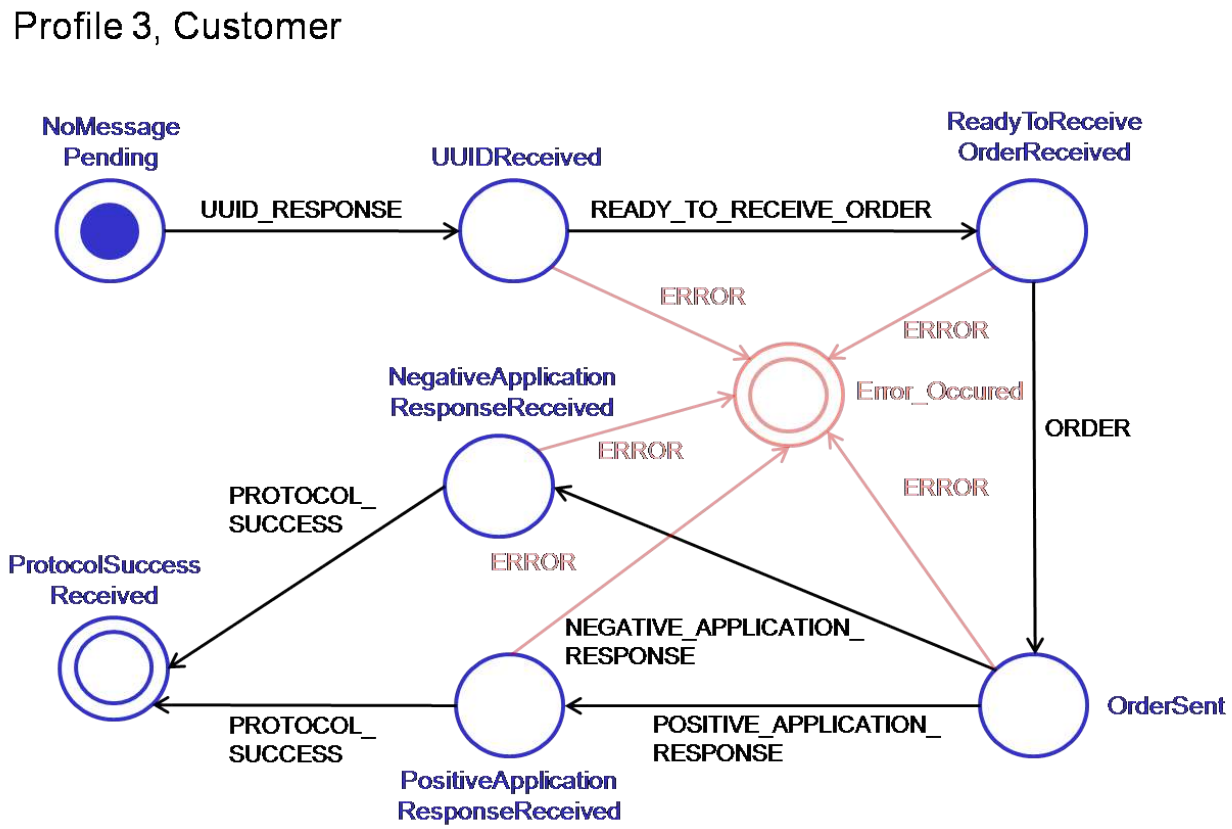


Figure 4.17: Customer's state graph of NES profile 3

4.3.4 Web Service Architectures and Implementations

Every Web service that processes business documents has four WSDL interfaces to offer different security related QoS-Features which are realized by WS-Security or SSL. For a detailed introduction to the used WSDL interfaces, see section 4.2.4. These four WSDL Interfaces for the security-QoS-Features are:

- NoQos
- SSL
- WS-Security
- WS-Security and SSL

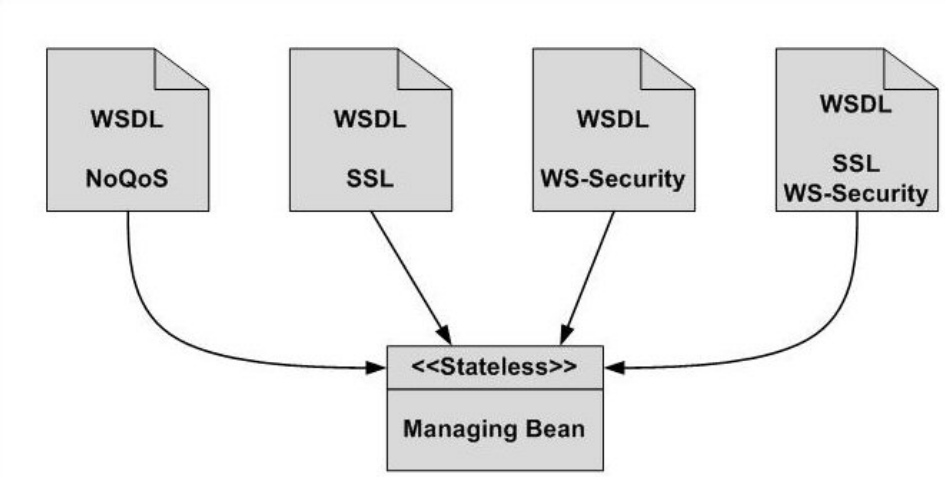


Figure 4.18: The four WSDL Interfaces to guarantee a SSL connection and WS-Security

All WSDL interfaces of one service refer to the same EJB stateless session bean which implements the business logic directly or prepares and delegates the Web service call for further processing. Full SSL and WS-Security support can be configured for the following Web services:

- Archive
- ExpressionEvaluation
- SchematronValidation
- SignatureCreation
- SignatureCheck
- XSDValidation

For Web services that do not handle any business documents, namely the Web services "AuthorizationCheck" and "UUID", comparable SSL or WS-Security support is not available, although this might be required depending in some projects. Technically speaking, implementing such support is not more complex than realizing security related QoS features for Web services that do exchange business documents.

All Web services are implemented as Enterprise Java Bean 3.0 modules which is a server-side component framework for modular construction of enterprise applications. The major advantage of the EJB concept is the strict separation of the EJB component and its runtime-environment, i.e., the container (the server). The container is responsible for providing a secure, transactional and distributed environment in which the Enterprise Java Beans can be executed.

Resources needed by a bean are communicated to the container by deployment annotations in the according bean. Such annotations are used, among other things, to instantiate other Enterprise Beans, create Web services on a stateless session bean basis or to use a persistence unit. In contrast to EJB 2.1, the EJB 3.0 specification uses the Java Persistence API (JPA)[5] which offers a very comfortable way of persisting plain old java objects (POJOs) which are annotated as Entities. The JPA provides an object-relational mapping and employs concepts also used in popular object-relational mapping frameworks like Hibernate[6].

The container also cares for EJB lifecycle-management: Creating and destroying bean instances or passivating and activating session beans.

4.3.4.1 Web Service: Archive

This Web service has the task of archiving and retrieving the business documents which are transmitted during the course of a business profile. Storage and retrieval of these messages can be seen as a means to ensure non-repudiation requirements.

Technically speaking, the Web service receives a message of type "StandardMessageType". Encapsulated in this message is the business document as well as meta data. The message is taken completely as is and being written (marshaled) to disk. Meta data such as the profile number, a unique message identifier and other information is written to a database table along with the filename of the archived message. In order to retrieve the archived messages the Archive Web service provides different methods to either retrieve a single specific message or a set of messages which belong to a collaboration instance.

There is also a method which permits to set the status of a message. The status indicates whether the message has been further processed or not.

This Web service comprises the following Enterprise Java Beans: The ArchivatorBean which provides all methods for marshaling or unmarshaling as well as functionality that inserts or updates database records when appropriate. The MessageDataBean which provides entities that contain a number of variables needed to properly marshal and unmarshal a given mes-

[5] `http://jcp.org/en/jsr/detail?id=220`
[6] `http://www.hibernate.org/`

sage. During the course of the archive operation these entities are written to or read from the database table. Finally, the WSArchiveBean which is a stateless Web service wrapper bean which does nothing more than call the ArchivatorBean's methods with the appropriate number of parameters and then returns the responses.

The operation of the archive service is as follows: Usually, the WSArchiveBean is being called along with a function name and parameters. The WSArchiveBean dispatches the method call to the corresponding method in the ArchivatorBean. Depending on the type of method called the arguments have to be an object of type StandardMessageType (for the purpose of marshaling to disk) or a combination of the variables declared in the MessageDataBean (this is the case when messages are to be retrieved from disk).

When archival of a message is specified the method *archiveMessage* is called. First, a unique filename is established, an output stream is set up and the message is written to the file. Second, the meta information from the message header and the filename that was established earlier are written to an object of type MessageData. This object is then stored in the database table.

When a single message is to be retrieved, the parameters uuid, rolenameID, recursionCount and messageID need to be supplied. Basically, these values are matched within the database table and the corresponding filename of the stored message is looked up. A new object of type StandardMessageType is created and the contents of the file are unmarshaled to this object. Finally the object is returned.

When multiple messages are to be retrieved, only uuid, rolenameID and recursionCount need to be specified since this is enough to identify the amount of messages specified. The process is the same as described above except that a list of type StandardMessageType with the number of messages found is returned.

When messages for a certain profile are requested, the number of parameters have to be rolenameID, profileID and status. This method returns a list of type StandardMessageType with all messages of a given profile that have a particular status.

When messages that have a particular status are requested, the number of parameters have to be a uuid and the desired status. This returns a list of type StandardMessageType containing the number of messages found. When a status change is desired the method setStatus has to be called with the parameters uuid, rolenameID, recursionCount and messageID. The status of the corresponding message is altered and set to true. Backend methods use the status to determine whether the particular message has been processed or not.

4.3.4.2 Web Service: AuthorizationCheck

This Web service provides a method to check the authorization of a person, organization or system (hereafter just called "user") identified by a name. For an authorization checkfor an action, e.g., sending an invoice in a certain profile, it is also necessary to get information about the document type and the NES profile in which the document type is sent (only both guarantee a unique identification of the action the user wants to perform). All in all, in order to check the

authorization of a user to perform a certain action a 3-Tuple of information with user name, sent document type and NES profile id is needed.

The MetaBlock already provides information about the document type sent and the NES profile, so this part of the originally received business document can be used for the authorization check request message. Additionally, a string with the name of the user is needed. For this reason, the StandardMessagePlusString (defined in file `standardMessagePlusString.xsd`) message type is used as input message to the AuthorizationCheck Web service.

The Web service interface receives this message and hands it over to the managing stateless session bean which checks the authorization information with the aid of a database. If a value that fits the requesting data is contained in the database the return value will be true, else it will be false.

In order to clarify the association between request and response the MetaBlock must be added to the return value. For this reason the return type is StandardMessagePlusBoolean which contains a boolean besides the MetaBlock.

The central bean of this Web service is the `AuthorizationManagerBean` which not only implements the logic for checking authorization but also contains functionality for adding and deleting authorization entries of users. All methods of this bean expect string arguments only, but the `BPEL2ServiceAuthorizationCheckNoQosService`, which is called by the BPEL process, needs to offer XML messages as input. Thus, another stateless session bean (`AuthorizationCheckBean`) is needed to extract the required information and redirect it to the `AuthorizationManagerBean`.

The `AuthorizationManagerBean` also persists the authorization information for every user and profile as `AuthorizationEntity`. This `AuthorizationEntity` is annotated as Entity and persisted using JPA functionality.

4.3.4.3 Web Service: SchematronValidation

The SchematronValidation service implements semantics validation of the different NES profile business documents. Schematron[7] is an XML dialect for defining assertions about relations between XML elements of an XML document. Each NES profile has its own Schematron files which are used to check the business documents received by this web service. Therefore, a different check depending on the profile and the message type has to be executed.

The Schematron `.sch` files are transformed step by step until the final result is a style sheet against which the body part of the Standard Message (which contains the business document) is validated. The resulting XML file contains a report of the validation process along with error and success messages. This file has to be processed to see if there were errors or not. The end result is sent back to the calling service to inform about the success or failure of the validation.

The Schematron Validation Service has not yet been implemented. Generally it would take an

[7] `http://www.schematron.com/`

object of type StandardMessageType as an argument. First of all the message's meta block has to be searched for the profile name and number in order to produce the right style sheet for validation. Once this has been done the profile's schematron file is processed and the output is a style sheet against which the business document contained within the message body can be validated. The validation result again is an .xml file containing the result of the validation process. This file has to be scanned for any errors that might have been encountered.

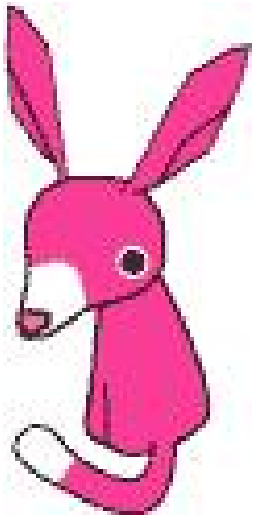


Figure 4.19: The Schematroll, mascot of Schematron

4.3.4.4 Web Service: SignatureCreation

In order to create a signature for an XML document the following standard procedure is necessary:

1. Create a key pair consisting of private and public key
2. Create an XML Signature using the private key [ERS+08]
3. Sign the XML Document with the signature and the private key.

This procedure is reflected in the architecture of the SignatureCreation Web service. A SignatureManager accepts the Web service call with a SignatureCreationRequestMessage. This special message contains, besides the MetaBlock and the common body of a message, the signers name (see `signatureCreationRequestMessage.xsd`). This additional information is needed to load the signer's key pair from the database or to create a new key pair for this signer if there is no key pair existent. The return type of this method is a StandardMessage which has exactly the body and the MetaBlock the SignatureCreationRequestMessage supplies except for the added Signature in the MetaBlock.

The Web service call is delegated from the SignatureManager to the DocumentSigner, which adds the XML Signature created by the SignatureCreater based on the XML document and

puts this signature at the correct place in the MetaBlock. Besides the signature, the private key of the signer is needed in order to sign the document correctly.

The SignatureCreater controls the creation of the signature which needs information about the private key, the signature algorithm and others.

The needed keys for the signing procedure are delivered by the KeyGenerator which either generates and stores the generated keys in case there are no keys for the signer or loads the keys of the signer from the database. Figure 4.20 visualizes the architecture of this Web service.

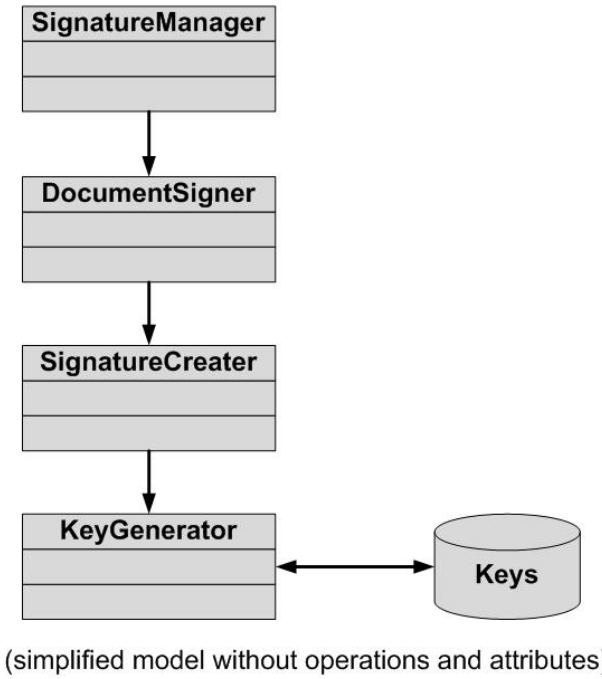


Figure 4.20: The SignatureCreation Web service, visualized as UML class diagram

In addition to the presented proceeding of creating a signature, there is one problem concerning the SOAP protocol and the BPEL engine. Both alter the XML namespace declarations of the submitted XML document which makes the document unusable for correct signature validation. Thus, it is necessary to introduce a mechanism ensuring the identical appearance before and after the transmission of the document. However, the XML documents used in this project use numerous XML namespaces which might be reordered or rewritten when transmitting them via SOAP. This is the reason why the mechanism of a namespace normalizer has to be introduced. This normalizer deletes every namespace prefix from the document and puts the namespace declarations into the tag in which they are needed. In addition to the normalization, the documents need to be serialized to the file system and de-serialized again because of a bug in JAXB which reconstructs the original document only after storing it to the file system. Thus, this Web service needs writing permission to a local device, and the standard configuration uses drive `d:` for this purpose. This mechanism works at the SignatureCreation as well as at the SignatureCheck side and makes XML documents amenable to validation.

4.3.4.5 Web Service: SignatureCheck

The SignatureCheck Web service realizes a validation mechanism for XML signatures. Therefore, the SignatureCheck Web service extracts the signature node form the MetaBlock of the incoming StandardMessage, retrieves the public key of the signer from this XML signature and performs an XML signature check. The result of this check is returned as StandardMessagePlusBoolean which returns, in addition to the MetaBlock of the original StandardMessage, a boolean that indicates whether the check of the signature was successful (true) or not(false).

4.3.4.6 Web Service: UUID

The UUID Web service provides a Standard Message with an immutable universally unique identifier (UUID) which represents a 128-bit value. This value is inserted at the appropriate position in the MetaBlock. The Standard Message, which then has a UUID, is sent back as return value.

The UUID Web service is implemented as a single stateless session bean using the UUID class of the Java API. This implementation provides a static method `randomUUID()` which creates a pseudo randomly generated UUID by using a cryptographically strong pseudo random number generator. This UUID is inserted as a string into the `<UUID>` tag of the MetaBlock.

4.3.4.7 Web Service: XPathEvaluation

The XPathEvaluation (or ExpressionEvaluation) Web service does not realize any QoS-Features, but is necessary for every BPEL process that contains a decision. Every business document may contain entries which affect the further execution of the BPEL process. For example, an entry ”false“ at the field ”AcceptedIndicator“ in an OrderResponseSimple message can terminate the process or, if it is set to ”true“, the process can proceed.

As required by ebBP, this Web service is designed to evaluate expressions of eight different expression languages. An ExpressionEvaluationRequestMessage is expected as argument and the message's expression is passed on to the managing class for the respective language. Due to time constraints, the only supported expression languages are XPath1 and XPath2. The return value is a StandardMessagePlusBoolean which contains the MetaBlock and the boolean value of the evaluated expression.

The incoming message contains, as every message does, the MetaBlock as well as a body. In addition, this message type contains two tags to capture the expression to evaluate and the expression language of the expression. The expression language will be identified by the ExpressionEvaluator which hands over the expression to the corresponding evaluator of the determined expression language. This expression evaluator will evaluate the expression and return a boolean value or an exception. This boolean value is wrapped in a `StandardMessagePlusBoolean`. In case of an exception, the exception will be re-thrown.

4.3.4.8 Web Service: XSDValidation

The XSD validation service has a task quite similar to the Schematron validation service. The business document is validated against its `.xsd` file for checking conformance to its schema definition, i.e., whether all necessary elements and attributes exist and whether only permissible elements and attributes are contained. The result of the validation is sent back to the calling service.

The XSD validation service has not yet been implemented. Its functionality is similar to the Schematron Validation Service. It takes an object of type StandardMessageType as an argument. The business document contained within the message body needs to be cast to a separate object of the same type as the business message before the validation can be done. XSD validation may be accomplished using JAXP (Java API for XML Processing). A JAXP implementation is available standalone, e.g., Xerces[8] , but is also already included within Java SE. A SAXParserFactory object needs to be instantiated in order to validate a message. This instance will receive the schema to validate against. Then a new parser object is generated from the parser factory and finally, the business message can be parsed and simultaneously validated with the `parse()` method invoked on the parser object.

[8] `http://xerces.apache.org/`

Chapter 5

Related Work

There have been other studies in the field of mapping higher level languages to BPEL concerning the choreography to orchestration translation and vice versa. The following approaches are discussed and compared with this technical report.

- RosettaNet PIP Compositions to BPEL
- ebXML BPSS to BPEL
- WS-CDL to BPEL
- BPMN to BPEL
- BPEL to ebXML BPSS

RosettaNet PIP to BPEL: Andreas Schönberger, Guido Wirtz: "Realising RosettaNet PIP Compositions as Web Service Orchestrations - A Case Study" and Andreas Schönberger: "Modeling and Validating Business Collaborations: A Case Study on RosettaNet" In [SW06] and in the technical report [Sch06] the authors describe an B2Bi approach using RosettaNet Partner Interface Processes (PIPs).

RosettaNet is a non-profit organization with more than 500 members by now. The main target of RosettaNet is the support for electronic business-2-business exchange. The core of RosettaNet are Partner Interface Processes (PIPs). Each PIP describes how processes of two different partners can be integrated[1]. The different PIPs are organized in clusters which are divided into segments. For example, the cluster "Inventory Management" consists of five segments. A segment in this cluster is "Inventory Allocation" which includes the PIP "Notify of Consumption".

These PIPs can be combined to larger processes. The authors suggest to model these compositions in two steps: The first step is to model the business logic in a "centralized perspective"

[1]See: `http://www.rosettanet.org/cms/export/sites/default/RosettaNet/Downloads/RStandards/ClustersSegmentsPIPsOverview_10Oct2008.pdf`

(CP) and then take a look at the more technical side by modeling a "distributed perspective" (DP) which concentrates on the distributed implementation. The CP is modeled using a UML activity diagram while the DP is modeled using BPEL processes. This separation in two layers - an overall choreography and a more technical orchestration, which is executable, can be found in this case study as well.

The clearest contrast between the papers [SW06] and [Sch06] to this work is that for modeling the choreography two different standards are used (UML vs. ebBP). Moreover, the focus in the discussed papers is generally put on modeling the two different perspectives and checking these models automatically. In contrast to this, in our case study the focus is on the automated translation of a given choreography to executable BPEL processes (and respecting QoS aspects).

ebXML BPSS to BPEL: Ja-Hee Kim, Christian Huemer: "From an ebXML BPSS choreography to a BPEL-based implementation" In a first step, the paper [KH04] shows that the technologies BPEL/Web service and ebBP can coexist. Therefore, the following approach is proposed: Each party is searching for a Business Process Specification in a registry. With this as input, the company is implementing a BPEL process for the role, it plans to perform within this business process. Next, it describes its technical capabilities using ebXML's CPP and registers its profile together with the role it can perform within the referenced ebBP. Another party then can search and find matching business partners. After they agree upon technical details, they do business using the BPEL processes and their interfaces.

In a second step, the paper shows an approach for transformation from BPSS (version 1.1) to BPEL. The approach proposes to create a BPEL process for each Business Transaction of the BPSS. This is done for ensuring modular architecture and better reusability of the Business Transactions. For complex collaborations, the paper proposes to base the transformation on well known work flow patterns proposed in *Distributed and Parallel Databases*[2].

Transformation of theses patterns can be done quite simply, if both, BPSS and BPEL, support them. If BPEL does not support these patterns, the transformation can use work-arounds also proposed by Van der Aalst. For mapping QoS attributes of the BPSS, the paper proposes three possible strategies: direct mapping to BPEL attributes, reflection by the process structure and usage of other standards like WS-Security.

The general approach of the paper is similar to this use case. ebBP is chosen for modeling the choreography and BPEL is chosen for modeling the orchestrations but in this work the ebBP version 2.0 is used. The transformation concept differs because this work builds BPEL processes based on Business Collaborations and not based on Business Transactions. The transformation of this use case is not based on well known work flow patterns, instead, own transformation constructs are designed. [KH04] do not offer formal transformation rules and do not describe their translation approach in detail, but some transformations of BPSS constructs are similar to the mapping constructs described in the work at hand.

For the mapping of QoS attributes, the paper proposes three general realization strategies that

[2]Van der Aalst, A., ter Hofstede, A. T., Kiepuszewski, B., and Barros, A. 2003. Workflow patterns. Distributed and Parallel Databases 14, 5 - 51

are also identified in this work. But [KH04] only enumerate possible realizations, e.g., the usage of WS-Security and do not state whether the strategies work on a concrete platform. This work offers a realization strategy for each QoS attribute of ebBP and shows that these strategies work based on the GlassFish Platform.

WS-CDL to BPEL: Jan Mendling, Michael Hahr: "From WS-CDL choreography to BPEL process orchestration" Since ebBP defines a choreography between business parties, the Web Services Choreography Description Language (WS-CDL) is an alternative to it. In [MH06], WS-CDL is used to describe the choreography of collaborating participants and BPEL, to describe the behavior of a participant in this choreography. [MH06] propose the automated BPEL process generation from WS-CDL using XSLT[3] transformations. Instead of using NES profiles, [MH06] combine a use case and certain interaction patterns as an example of a choreography. Compared with ebBP, WS-CDL is technologically more closely related to Web services, while ebBP focuses the business part, especially business related QoS features. Considering that, Mendling's derivation is technically oriented towards the mapping of WS-CDL constructs to BPEL. This work takes the transformation a step further to get closer to real-life business needs, in particular security features.

WS-CDL to BPEL: Ingo Weber, Jochen Haller, Jutta A. Mülle: "Automated derivation of executable business processes from choreographies in virtual organizations" Similarly to [MH06], [WHM06] use WS-CDL as a basis to create executable BPEL processes. They take a different approach by utilizing a knowledge base to fill the gap between the local and global collaboration perspective. This is used for identifying parts of the choreography in WS-CDL which cannot be translated to BPEL directly. In this report the ebBP transformation was realized using mapping constructs as described in section 4.1.1.1. The restriction to binary relations in ebBP pointed out by [WHM06] are not applicable anymore in version 2.0 of the ebBP specification. Although this work only considers binary collaborations, as defined in the NES profiles, it is theoretically possible to have multi-party collaborations. Also, [WHM06] do not show the realization of QoS features which is one of the main goals of this contribution.

ebXML BPSS to BPEL: Bahareh Rahmanzadeh Heravi, Mohammadreza Razzazi: "Utilizing WS-BPEL business processes through ebXML BPSS" This paper [HR07] proposes an approach which is oppositional to the general approach of our use case. The aim is to model ebBP Business Collaborations that mimic the behavior and the characteristics of WS-BPEL processes. WS-BPEL as the de facto standard for processes at the level of Web services orchestration should be utilized by the B2B framework of ebXML. Therefor, registration and search of business processes is proposed to be based on ebBP. The execution should then be based on the WS-BPEL processes represented by the ebBP.

Although the direction of transformation is oppositional, there are some similarities between [HR07] and the work at hand. Both base the mapping of BPEL processes on Business Col-

[3] `http://www.w3.org/TR/xslt20/`

laborations. This means, each BPEL process is mapped to a ebBP Business Collaboration or vice versa. Mappings at lower levels are not comparable. This could be because of the different directions of the mapping.

The aim of [HR07] is to use the registry functionality of the ebXML framework. Such registry functionality is not considered in the work at hand, but constitutes a sensible extension point. Furthermore, [HR07] do not regard the realization and implementation of QoS features and attributes. Finally, a major difference is the intended use of ebBP models. In the work at hand, ebBP is used for describing choreographies, while in [HR07] it is rather used as an orchestration representation for using registry functionality.

Chapter 6

Conclusion and Future Work

This technical report contributes by showing the feasibility of mapping the high-level ebXML language ebBP to the executable BPEL markup. Particular attention has been paid to the QoS attributes defined in ebBP. These are realized with several WS-* standards, BPEL constructs and self-implemented Web services. A working prototype developed for this report proves the concept.

As mentioned initially, Quality of Service, i.e., the reliable, secure, non-manipulable, undeniable transmission of business documents, is essential for electronic Business-2-Business-Integration. The implemented NES business case profiles show that real-life scenarios are realizable with existing standards. Considering the experiences of this project, a full-featured product could be created from this prototype with manageable effort. However, since many of the used technologies are cutting-edge (some needed patches of the GlassFish Application Server were published at the end of the implementation phase), further development work is needed in this area to make the development of tools such as this project's prototype more convenient.

In near-future terms, the NES profiles with recursion should be fully tested and completely operational Web services for XSD and Schematron validation should be implemented which can be done in a timely fashion because most of the work is already done. The next step then would be to support not only binary collaborations and a broader range of ebBP constructs, in particular *Fork* or *Join*.

In the far future, the automated lookup and integration of business partners, from automatically generated business conditions to the execution of business transactions, is a desirable goal. Some of this functionality is already envisaged in the ebXML framework which should be reused accordingly.

Bibliography

[ACC+02] Selim Aissi, Arvola Chan, James Bryce Clark, David Fischer, Tony Fletcher, Brian Hayes, Neelakantan Kartha, Kevin Liu, Pallavi Malu, Dale Moberg, Himagiri Mukkamala, Peter Ogden, Marty Sachs, Yukinori Saito, David Smiley, Tony Weida, Pete Wenzel, and Jean Zheng. Collaboration-protocol profile and agreement specification version 2.0. Technical report, OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee, 2002.

[BBB+02] Ralph Berwanger, Dick Brooks, Doug Bunting, David Burdett, Zrvola Chan, Sanjay Cherian, Cliff Collins, Philippe DeSmedt, Colleen Evans, Chris Ferris, David Fischer, Jim Galvin, Brian Gibb, Scott Hinkelman, Jim Hughes, Kazunori Iwasa, Ian Jones, Brad Lund, Bob Miller, Dale Moberg, Himagiri Mukkamala, Bruce Pedretti, Yukinori Saito, Martin Sachs, Jeff Turpin, Aynur Unal, Cedrec Vessell, Daniel Weinreb, Pete Wenzel, Prasad Yendluri, and Sinisa Zimek. Message service specification version 2.0. Technical report, OASIS ebXML Messaging Services Technical Committee, 2002.

[BCC+02] Kathryn Breininger, Lisa Carnahan, Joseph M. Chiusano, Suresh Damodaran, Mike DeNicola, Anne Fischer, Sally Fuger Jong Kim, Kyu-Chul Lee, Joel Munter, Farrukh Najmi, Joel Neu, Sanjay Patil, Neal Smith, Nikola Stojanovic, Prasad Yendluri, and Yutaka Yoshida. Oasis/ebxml registry services specification v2.0. Technical report, OASIS/ebXML Registry Technical Committee, 2002.

[BCvR03] Tom Bellwood, Luc Clément, and Claus von Riegen. *UDDI Spec Technical Committee Specification.* OASIS, 3.0.1 edition, 10 2003.

[BHM+04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. Standard, W3C World Wide Web Consortium, http://www.w3.org/TR/ws-arch/, November 2004. Visited on October 30th 2008.

[CCMW02] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1.* World Wide Web Consortium, March 2002.

[DLS05] Glen Dobson, Russell Lock, and Ian Sommerville. Qosont: a qos ontology for service-centric systems. In *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 80–87, Washington, DC, USA, 2005. IEEE Computer Society.

[EGV04] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2004.

[ERI+02] Donald Eastlake, Joseph Reagle, Takeshi Imamura, Blair Dillaway, and Ed Simon. *XML Encryption Syntax and Processing*. W3C, December 2002.

[ERS+08] Donald Eastlake, Joseph Reagle, David Solo, Frederick Hirsch, Thomas Roessler, Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. *XML Signature Syntax and Processing (Second Edition)*. W3C, June 2008.

[FPD+07] Paul Fremantle, Sanjay Patil, Doug Davis, Anish Karmarkar, Gilbert Pilz, Steve Winkler, and Ümit Yalçinalp. Web services reliable messaging (ws-reliablemessaging) version 1.1. Standard, Oasis Open, http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01.html, June 2007. Available as PDF, HTML and DOC, visited on December 22nd 2008.

[GHR06] Martin Gudgin, Marc Hadley, and Tony Rogers. *Web Services Addressing 1.0 - Core*. W3C, May 2006.

[Gro07a] Northern European Subset Group. Nes information model architecture. Technical report, Northern European Subset Group, 2007.

[Gro07b] Northern European Subset Group. Profile overview version 2.0. Technical report, Northern European Subset Group, 2007.

[HR07] Bahareh Rahmanzadeh Heravi and Mohammadreza Razzazi. Utilizing ws-bpel business processes through ebxml bpss. Technical report, Coputer Engineering and Information Technology Department, Amirkabir University of Technology, Teheran, 2007.

[JEA+07] Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web services business process execution language version 2.0. Standard, Oasis Open, April 2007. Visited on November 3rd 2008.

[KH04] Ja-Hee Kim and Christian Huemer. From an ebxml bpss choreography to a bpel-based implementation. Technical report, University of Vienna and Research Studios Austria, 2004.

[LKN+07] Kelvin Lawrence, Chris Kaler, Anthony Nadalin, Marc Goodner, Martin Gudgin, and Abbie Barbirand Hans Granqvist. *WS-Trust 1.3*. OASIS, March 2007.

[MH06] Jan Mendling and Michael Hafner. From WS-CDL choreography to BPEL process orchestration. *Journal of Enterprise Information Management (JEIM). Special Issue on MIOS Best Papers*, 2006.

[Nor07] Northern European working group for development of a subset for UBL 2.0. *Northern European Subset Profiles*, 2 edition, July 2007.

[NRFJ07] Eric Newcomer, Ian Robinson, Max Feingold, and Ram Jeyaraman. Web services coordination (ws-coordination) 1.1. Standard, Oasis Open, December 2007. Available as PDF, visited on December 27th 2008.

[NRLF07] Eric Newcomer, Ian Robinson, Mark Little, and Tom Freund. Web services business activity (ws-businessactivity) version 1.1. Standard, Oasis Open, July 2007. Available as PDF, visited on December 27th 2008.

[NRLW07] Eric Newcomer, Ian Robinson, Mark Little, and Andrew Wilkinson. Web services atomic transaction (ws-atomictransaction) version 1.1. Standard, Oasis Open, July 2007. Available as PDF, visited on December 27th 2008.

[OAS06] OASIS. Web Services Security v1.1, February 2006.

[OM03] Ed Ort and Bhakti Mehta. Java architecture for xml binding (jaxb), March 2003.

[OMG08] OMG. *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification*. OMG, Object Management Group Headquarters 140 Kendrick Street Building A, Suite 300 Needham, MA 02494 USA, 1.1 edition, April 2008.

[Sch06] Andreas Schönberger. Modelling and validating business collaborations: A case study on rosettanet. Technical report, Universität Bamberg; Fakultät Wirtschaftsinformatik und Angewandte Informatik., 2006.

[SW06] Andreas Schönberger and Guido Wirtz. Realising rosettanet pip compositions as web service orchestrations - a case study. In Hamid R. Arabnia, editor, *CSREA EEE*, pages 141–147. CSREA Press, 2006.

[VOH+07] Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and Ümit Yalçinalp. *Web Services Policy 1.5 - Framework*. W3C, September 2007.

[WHM06] Ingo Weber, Jochen Haller, and Jutta A. Mülle. Automated derivation of executable business processes from choreograpies in virtual organizations. In *In: F. Lehner, H. Nösekabel, P. Kleinschmidt (eds.): Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006), Band 2, XML4BPM Track, GITO-Verlag Berlin*, pages 313–327, March 2006.

[WPA+01] James Whittle, Sue Probert, Mike Adcock, Gait Boxman, and Thomas Becker. Core component overview version 1.05. Technical report, UN/CEFACT and OASIS, 2001.

[YWM+06] John Yunker, David Webber, Dale Moberg, Kenji Nagahashi, Stephen Green, Sacha Schlegel, and Monica J. Martin. ebxml business process specification schema technical specification v2.0.4. Standard, Oasis Open, December 2006. Available as PDF, visited on November 14th 2008.

Appendix A

User Manual

A.1 Manual of the Translator

The following is a short manual for usage of the Translator and its file and folder structure.

Structure of Folders The following shows the folder structure of the sub folders of the Translator.

- `src`: folder containing all sources;
- `res`: folder containing additional resources, necessary for the Translator;
- `lib`: folder for external libraries, used by the Translator;
- `testfiles`: folder for the ebBP files to be interpreted; for each profile, there is a sub folder with role specific property files and the ebBP file, following the naming pattern `dsg_bamberg_profileX.xml`. `X` is placeholder for the profile number.
- `target`: folder containing the created BPEL, XSD and WSDL documents; there is a sub folder for each created profile with the name `profileX` if more than one profile has been created. If only one profile has been created, there is only one sub folder. Each profile specific sub folder contains further sub folders for customer and supplier. Each of these role specific sub folders contains all necessary files for deployment, separated in WSDL and BPEL files.
- `dist`: folder containing all necessary archives, libraries and resources for running the Translator and the Java Doc; this folder is created and filled after running Ant.
- `bin`: folder for the binary classes after compiling;

Settings before Starting the Translator It is recommended not to change the program specific properties within the `res` folder. The `program.properties` configures output and input paths of the Translator.

The `role.properties` files within the testfile folders of the profiles are preconfigured. The following logical groups of properties can be altered and set. The property files are preconfigured and need only to be changed if services, backend or processes are running on hosts different from the one configured in the file.

- `output:` specifies the output path for the generated files of the role;
- `IDs:` specification of the profile id, the integration party's role id, the id of the partner role and of the recursion;
- `keystore:` configuration of the keystore used by WS-Security;
- `truststore:` configuration of the truststore used by WS-Security;
- `bpel2backend:` specification of the URLs of the backend system for the different possible QoS feature combinations; used by the BPEL process to communicate with its backend; the naming of the properties follows the pattern `bpel2backend.general.`*qosfeatures*`.url`.
- `backend2bpel:` specification of the URLs of the BPEL process, necessary for communication from backend system to BPEL process with regard to the particular QoS feature combination; the naming of the properties follows the pattern `backend2bpel.general.`*qosfeatures*`.url`.
- `bpel2bpel:` specification of the URLs of the BPEL processes for communication between the BPEL processes; the values for the own integration party's role have to be set; for URLs of the partner processes, please see paragraph „Settings after Running the Translator"; the naming of the properties follows the pattern `bpel2bpel.general.`*qosfeatures*`.url`.
- `bpel2service:` specification of the URLs of the different used Web services for all possible QoS feature combinations; the naming of the properties follows the pattern `bpel2service.`*servicename.qosfeatures*`.url`.
- other: these properties determine some miscellaneous parameters.

Starting the Translator For running the Translator, the Ant-Build-Script[1] has to be executed. The main folder of the Translator contains the `build.xml` file. To start the Ant-Build-Script, go to the main folder of the Translator and run Ant. There are two options for running it:

- `ant`: compiles the Translator;
- `ant profileX`: compiles the Translator and runs it for the specified profile. `X` has to be replaced by the profile number.

[1] http://ant.apache.org/

The output of the Translator is put to the sub folder `target`.

Settings after Running the Translator After a successful ending of the generation process, some changes have to be made to the WSDL files responsible for the communication between the BPEL processes. These WSDL files can be identified by the name prefix `BPEL2BPEL`. Note that, because of different QoS combinations, there can be multiple BPEL2BPEL WSDL files. Listing A.1 shows the code to be changed. The role of customer has to change the host name `localhost` within the `soap:address location` to the host name of the partner process and vice versa. The changes have to be the same to all WSDL files of the same role. If the partner process runs on localhost, no changes have to be made. This change is necessary, because the Translator fills the WSDLs of each role with host name localhost.

```
1 ...
2 <service name="BPEL2BPELNoQosRMcb_profile3global_role_customerServiceService">
3                 <port name="BPEL2BPELNoQosRMcb_profile3global_role_customerServicePort"
4                  binding ="
                       tns:BPEL2BPELNoQosRMcb_profile3global_role_customerServiceBinding">
5                     <soap:address location="http://localhost:50002/
                          BPEL2BPELNoQosRMServiceService/BPEL2BPELNoQosRMService"/>
6                 </port>
7 </service>
8 ...
```

Listing A.1: Extract of the WSDL file to be changed

Composite Application and Deployment In order to deploy the processes the files for each role have to be packed into one BPEL module and then added to a JBI Composite Application for each role[2]. The so created Composite Applications can then be deployed on a GlassFish Application Server; the two Composite Applications have to be deployed on different servers.

[2]More information about how to generate BPEL modules and Composite Applications with NetBeans IDE can be found in the "SOA Application Learning Trail" of NetBeans at `http://www.netbeans.org/kb/trails/soa.html`

A.2 Manual of the Backend System

GlassFish Configuration After installation and before the dummy backend can be deployed, some configuration of the GlassFish Application Server has to be done. The following list contains all important tasks to enable GlassFish to run the backend system:

- **Enabling SSL:** To enable SSL for Web service communication see the Java EE 5 Tutorial.[3]
- **Enabling Log4j:** Log4j is used as logging tool within the system. In order to enable Log4j for GlassFish an up-to-date Log4j Jar and a Log4j property file has to be referenced in the "Path settings" of GlassFish. To change the "Path Settings" use the Admin Console of GlassFish; the required settings can be found in the in the tab "JVM Settings". To enable Log4j the absolute path to the jar file has to be entered in the text box "System Classpath", e.g., "`D:\glassfish\domains\domain1\lib\`". If the Log4j property file also is in the same path you are done; if not it is necessary to add this path as well. An example of a preconfigured Log4j configuration file can be found within the project distribution.
- **JMS Queue creation:** A JMS Queue is needed for temporarily storing messages. First create a "Physical Destination" in "Configuration" - "Java Message Service" with any name and type "javax.jms.Queue". Then go to "Resources" - "JMS Resources" - "Destination Resources" and create a new JMS resource with the name "jms/pi3b2biWaitingMessages Queue" using the physical destination you just created and the type "javax.jms.Queue". Finally, go to "JMS Resources" - "Connection Factories" and create a connection factory with the name "jms/pi3b2biCF" and type "javax.jms.QueueConnectionFactory".
- **JDBC Pool configuration:** As mentioned above the backend system needs a database to save some information. This database has to be registered in GlassFish. The required settings have to be done under the path "Resources" - "JDBC" - "JDBC Resources". Create a new JDBC resource there with JNDI-Name "`jdbc/sample`". If no DB pool already exists, a pool has to be created first.
- **Web service property file:** Furthermore the file `webservice.properties` has to be referenced by the server system classpath - therefore the same procedure as for "Enabling Log4j" has to be done. The file `webservice.properties` contains the addresses of all needed Web services. These Web services are the SignatureCreation service and the profile specific addresses of the BPEL processes. For each profile four Web service addresses are needed to cover the four possible QoS combinations which are realized with WS-* standards. The default addresses reference BPEL processes deployed at the localhost. Also, there is a variable named "jms.maxRetryCount" which sets the maximum number a message should be tried to deliver to the backend's state machine.

Deployment The backend system needs to be deployed on two different hosts; both configured as described above. Before running the application the needed BPEL processes and the

[3] Available at: `http://java.sun.com/javaee/5/docs/tutorial/doc/bnbxw.html`

QoS realizing Web services have to be installed. See sections A.1 and A.3 for information on how to achieve this.

Usage of the Backend System The backend system is generic for all profiles and roles - so first you have to choose which profile should be tested (See Fig. A.1). In the next step the performing role is determined (See Fig. A.2). Depending on this decisions you are able to start a new process or not. In each profile only one role is able to initiate a new process. For example, in profile 3 "Basic Order Only" a customer sends an order to a supplier. Hence, the customer begins the process. After clicking the button "Start new process" a message is sent to the corresponding BPEL process which then controls the process by sending messages or requests of messages ("`ReadyToReceiveXXX`") to the backend. Simply "handle" all incoming messages in the order they appear at the web interface. The message overview screen is shown in figure A.3.

The backend system does not know which QoS features are actually accepted by the BPEL processes for incoming messages. Thus, the user must know which QoS-Features are activated and must therefore activate the right options while sending a business document such as an order or an invoice.

Logging the Exchanged Messages To see the different QoS realizations in action it is possible to trace the exchanged messages using various possibilities. One possibility is to enable the Web service monitoring feature of GlassFish. This is useful to see the incoming message for the QoS features which are realized by Web services. To enable monitoring click on the menu item "Web Services" and choose the Web service you want to monitor by clicking on the service name. Click "Monitor" and enable the monitoring under the tab "Configuration". There, set the Monitoring level to "HIGH". The logged message can be found in the Tab "Messages". Unfortunately the messages do not contain any of the WS-* information, e.g., a WS-Security header because the logging is on a too high level of the service stack. To see this information, logging on the network layer is needed. For logging on the network layer various tools such as TCPmon[4] and Wireshark[5] exist.

[4]Available at: `https://tcpmon.dev.java.net/`
[5]Available at: `http://www.wireshark.org/`

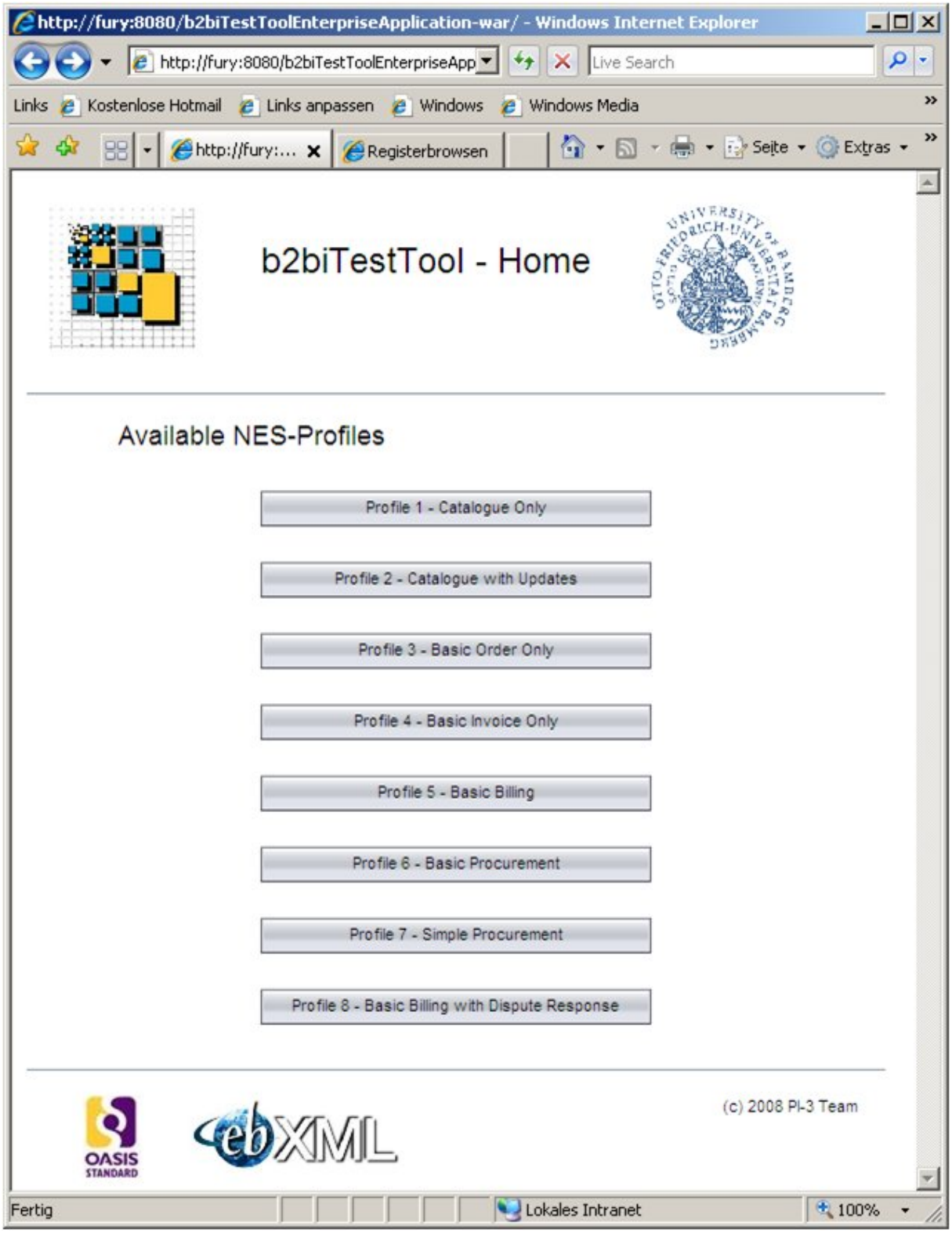


Figure A.1: Start page of the backend system

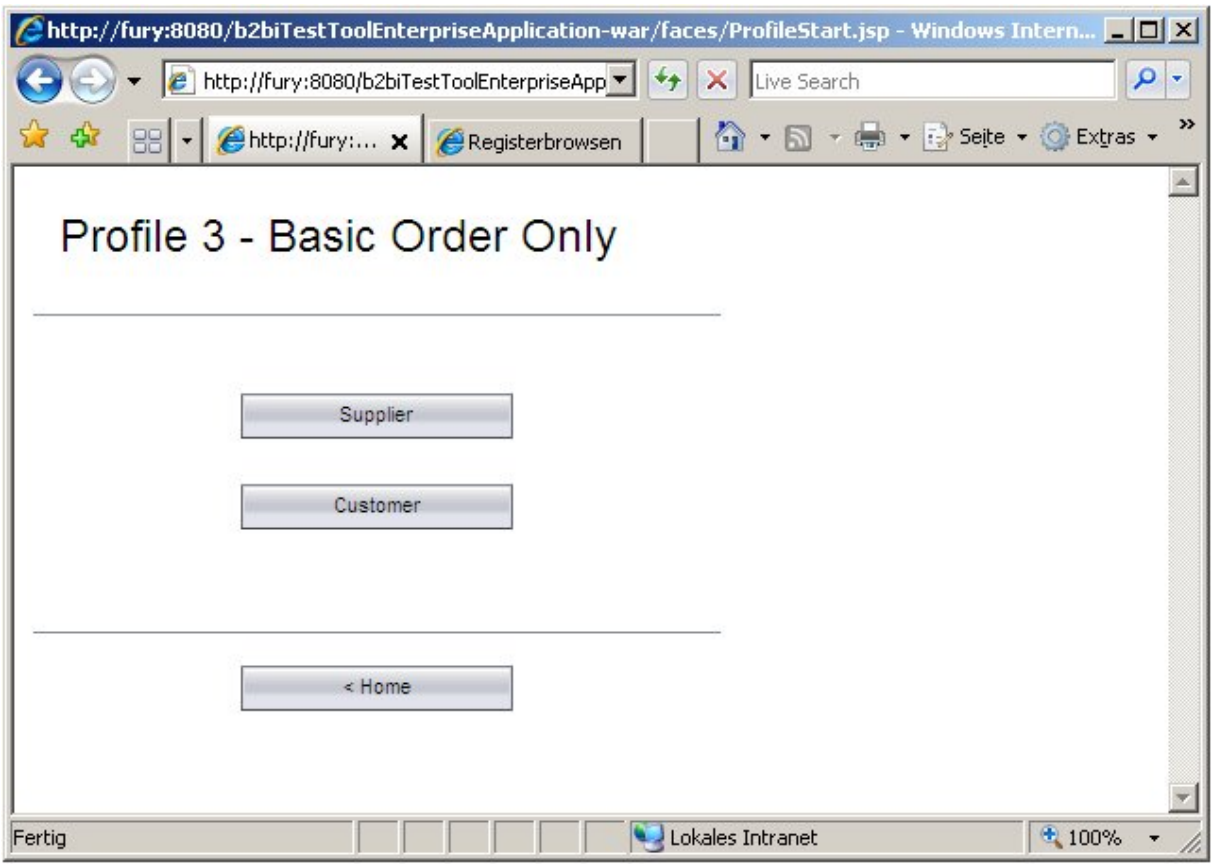


Figure A.2: Selection of the performing role

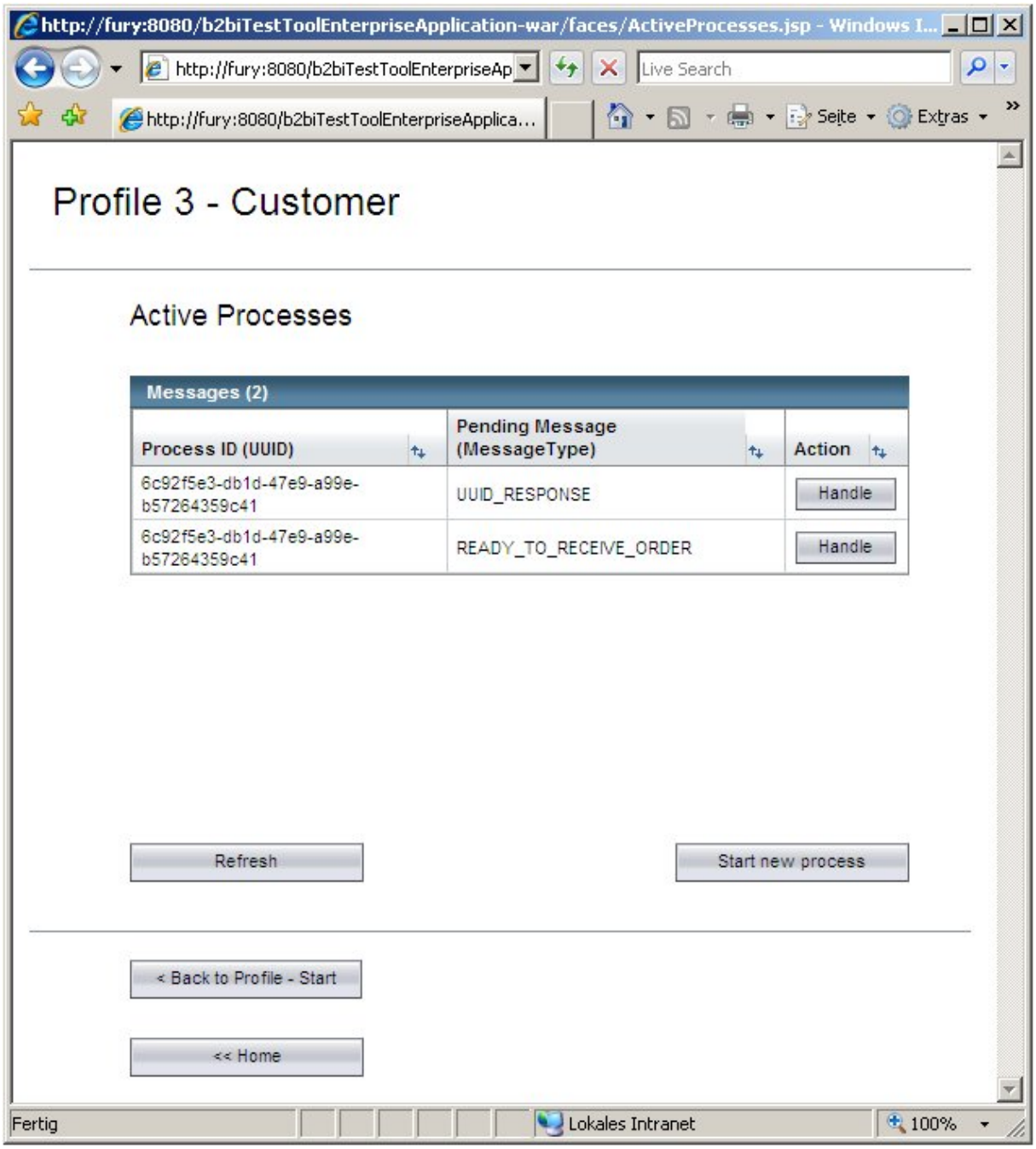


Figure A.3: Screen showing the pending messages

A.3 Manual of the Web services

Building the Web services Each Web service includes an Ant build file which generates a deployable `.jar` file in the folder dist of the according Web service. To build a Web service, check the path of the GlassFish server in the `build.xml` and alter this path if necessary. Afterwards, execute Ant in the directory of the Web service to be built by typing in `ant` in the root directory of a Web service. This procedure is the same for every Web service.

Deploying the Web services The deployment is also almost the same for every Web service. Deploy the `.jar` file of the Web service to be deployed to GlassFish. This `.jar` file is located in the folder `dist` of the Web service root folder.

Configuration of the Web services Most of the required Web services work without configuration, however the AuthorizationCheck Web service as well as the two Signature Web services need some configuration.

For AuthorizationCheck Web service authorizations users have to be added. Therefor, open the `User2ServiceAuthorizationManagementNoQosService`, use the method `addOperation` and type in the user name in the first field, the profile id in the second field and the operation in the third field. By using the keyword “all” for the operation field the specified user gets any rights in the specified profile. In order to cancel authorizations, use the method `deleteOperation`, which works exactly like `addOperation`, only the other way round.

Both Web services concerning the signature need writing permission to a local device, e.g., D:, in order to store the signed and normalized documents temporarily. For further information, see section 4.3.4.4.

Appendix B

Used Tools

B.1 IDEs

- Eclipse 3.4 http://www.eclipse.org/
- NetBeans 6.1 Patch 4 Candidate (200810140114) + OpenESB addons http://www.netbeans.org
- IBM Rational Software Architect v7 http://www.ibm.com/software/awdtools/architect/swarchitect/

B.2 Test

- soapUI 2.0 http://www.soapui.org/
- Wireshark 1.0.5 http://www.wireshark.org/

B.3 Runtime

- Java SE Development Kit (JDK) 6 Update 11 http://java.sun.com/
- GlassFish ESB Release Candidate 1 https://open-esb.dev.java.net/

B.4 Documentation

- TeXnicCenter 1.0 RC1 http://www.texniccenter.org/
- MikTex 2.7 http://www.miktex.org/

- MS Office 2003 `http://office.microsoft.com`

Appendix C

ebBP Modeling Naming Conventions

These naming conventions are the basis for the used naming expressions in the ebBP files. These conventions may differ slightly in special situations for uniqueness reasons.

The attributes are depicted in red, the strings are depicted in green.

Business Signals

name = **name** of the tags specification
nameId = capital letters of the **name**s in lower case letters + **2**

Specification

nameId = capital letters of the **name**s in lower case letters + **bpss2**

Business Documents

nameId = **bd_** + **name**

Specification

name = **nameId**
name = {**common** | **basic**} + **NES2_0** + **name** of the business document with a capital letter at the beginning

For custom XSD files only the name of XSD type will be used!

Business Transactions

name = the purpose/function of the business transaction
nameId = **bt_** + **name** (abbreviations of long words using the abbreviation dictionary)

RequestingRole / RespondingRole

name = {**initiator** | **responder**}
nameId = **nameId** of the business transaction + **_role_** + **name**

RequestingBusinessActivity / RespondingBusinessActivity

name = description of the function of the activity
nameId = **nameId** of the business transaction + **_ba_** + {**req** | **resp**}
DocumentEnvelope
name = **name** of the sent document
nameId = **name** of the business transaction + **_doc_** + **name**
ReceiptAcknowledgement
name = **ra**
nameId = **name** of the business transaction + **_ack_** + **name**
ReceiptAcknowledgementException
name = **rae**
nameId = **name** of the business transaction + **_ack_** + **name**
AcceptanceAcknowledgement
name = **aa**
nameId = **name** of the business transaction + **_ack_** + **name**
AcceptanceAcknowledgementException
name = **aae**
nameId = **name** of the business transaction + **_ack_** + **name**

Notification

name = **notify** + the **name** of the sent document

Commercial Transaction

name = **issue** + the **name** of the sent document

Information Distribution

name = **distribute** + the **name** of the sent document

Business Collaborations

Of a profile:
nameId = **cb_probile**{ **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** }**global**
Of an inner collaboration:
nameId = **cb_inner_** + function of this transaction

Role Supplier / Role Customer

name = {Supplier | Customer}
nameId = **nameId** of the business collaboration + **_role_** + **name** (lower case only)

For all the BTAs, CAs, Decs, following rule is applied when used in an inner collaboration:
After the first _-Sign an additional information field is inserted into the **nameId** field:
{**cbsi** | **cbesi** | **cbscn** | **cbescn**}
The abbreviations are explained in the abbreviation dictionary.

For all Successes and Failures, following rule is applied when used in an inner collaboration:
Before the **nameId** one of the following expressions is inserted:
{**cbsi** | **cbesi** | **cbscn** | **cbescn**}
The abbreviations are explained in the abbreviation dictionary.

Start

name = **start of the** + **name** of the business collaboration
nameId = **start_** + **nameId** of the business collaboration

Business Transaction Acivity

name = the purpose of the activity, often derived from the **name** of the business transaction
nameId = **bta_** + often derived from the **nameId** of the business transaction

Collaboration Activity

name = the purpose of the activity, often derived from the **name** of the business transaction
nameId = **ca_** + often derived from the **nameId** of the business transaction

Decision

name = description after which action this decision occurs
nameId = **dec_** + part after the _ sign of the **name** of the action before the decision

Success

nameId = **success**

Failure

nameId = {**techFail** | **businessFail**}

Abbreviation Dictionary

App	-	Application
Resp	-	Responding
cbsi	-	collaboration send invoice
cbesi	-	collaboration external send invoice
cbscn	-	collaboration send credit note
cbescn	-	collaboration external send credit note

Appendix D

List of previous University of Bamberg reports

Bamberger Beiträge zur Wirtschaftsinformatik

Stand August 26, 2009

Nr. 1 (1989) Augsburger W., Bartmann D., Sinz E.J.: Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)

Nr. 2 (1990) Esswein W.: Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG

Nr. 3 (1990) Augsburger W., Rieder H., Schwab J.: Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen

Nr. 4 (1990) Ferstl O.K., Sinz E.J.: Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)

Nr. 5 (1990) Ferstl O.K., Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)

Nr. 6 (1991) Augsburger W., Rieder H., Schwab J.: Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten

Nr. 7 (1991) Augsburger W., Rieder H., Schwab J.: Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU / Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten für den Endanwender

Nr. 8 (1991) Schwab J.: Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung

Nr. 9 (1992) Gross H.-P.: Eine semantiktreue Transformation vom Entity-Relationship-Modell in das Strukturierte Entity-Relationship-Modell

Nr. 10 (1992) Sinz E.J.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)

Nr. 11 (1992) Ferstl O.K., Sinz E. J.: Glossar zum Begriffsystem des Semantischen Objektmodells

Nr. 12 (1992) Sinz E. J., Popp K.M.: Zur Ableitung der Grobstruktur des konzeptuellen Schemas aus dem Modell der betrieblichen Diskurswelt

Nr. 13 (1992) Esswein W., Locarek H.: Objektorientierte Programmierung mit dem Objekt-Rollenmodell

Nr. 14 (1992) Esswein W.: Das Rollenmodell der Organsiation: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen

Nr. 15 (1992) Schwab H. J.: EISREVU-Modellierungssystem. Benutzerhandbuch

Nr. 16 (1992) Schwab K.: Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip

Nr. 17 (1993) Schwab K.: Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystems zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen. Dissertation

Nr. 18 (1993) Ferstl O.K., Sinz E.J.: Der Modellierungsansatz des Semantischen Objektmodells

Nr. 19 (1994) Ferstl O.K., Sinz E.J., Amberg M., Hagemann U., Malischewski C.: Tool-Based Business Process Modeling Using the SOM Approach

Nr. 20 (1994) Ferstl O.K., Sinz E.J.: From Business Process Modeling to the Specification of Distributed Business Application Systems - An Object-Oriented Approach -. 1st edition, June 1994

Ferstl O.K., Sinz E.J. : Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach -. 2nd edition, November 1994

Nr. 21 (1994) Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells zur Modellierung von Geschäftsprozessen

Nr. 22 (1994) Augsburger W., Schwab K.: Using Formalism and Semi-Formal Constructs for Modeling Information Systems

Nr. 23 (1994) Ferstl O.K., Hagemann U.: Simulation hierarischer objekt- und transaktionsorientierter Modelle

Nr. 24 (1994) Sinz E.J.: Das Informationssystem der Universität als Instrument zur zielgerichteten Lenkung von Universitätsprozessen

Nr. 25 (1994) Wittke M., Mekinic, G.: Kooperierende Informationsräume. Ein Ansatz für verteilte Führungsinformationssysteme

Nr. 26 (1995) Ferstl O.K., Sinz E.J.: Re-Engineering von Geschäftsprozessen auf der Grundlage des SOM-Ansatzes

Nr. 27 (1995) Ferstl, O.K., Mannmeusel, Th.: Dezentrale Produktionslenkung. Erscheint in CIM-Management 3/1995

Nr. 28 (1995) Ludwig, H., Schwab, K.: Integrating cooperation systems: an event-based approach

Nr. 30 (1995) Augsburger W., Ludwig H., Schwab K.: Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit

Nr. 31 (1995) Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse

Nr. 32 (1995) Gunzenhäuser R., Duske A., Ferstl O.K., Ludwig H., Mekinic G., Rieder H., Schwab H.-J., Schwab K., Sinz E.J., Wittke M: Festschrift zum 60. Geburtstag von Walter Augsburger

Nr. 33 (1995) Sinz, E.J.: Kann das Geschäftsprozeßmodell der Unternehmung das unternehmensweite Datenschema ablösen?

Nr. 34 (1995) Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -

Nr. 35 (1995) Sinz E.J.: Serviceorientierung der Hochschulverwaltung und ihre Unterstützung durch workflow-orientierte Anwendungssysteme

Nr. 36 (1996) Ferstl O.K., Sinz, E.J., Amberg M.: Stichwörter zum Fachgebiet Wirtschaftsinformatik. Erscheint in: Broy M., Spaniol O. (Hrsg.): Lexikon Informatik und Kommunikationstechnik, 2. Auflage, VDI-Verlag, Düsseldorf 1996

Nr. 37 (1996) Ferstl O.K., Sinz E.J.: Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems, July 1996

Nr. 38 (1996) Ferstl O.K., Schäfer R.: Eine Lernumgebung für die betriebliche Aus- und Weiterbildung on demand, Juli 1996

Nr. 39 (1996) Hazebrouck J.-P.: Einsatzpotentiale von Fuzzy-Logic im Strategischen Management dargestellt an Fuzzy-System-Konzepten für Portfolio-Ansätze

Nr. 40 (1997) Sinz E.J.: Architektur betrieblicher Informationssysteme. In: Rechenberg P., Pomberger G. (Hrsg.): Handbuch der Informatik, Hanser-Verlag, München 1997

Nr. 41 (1997) Sinz E.J.: Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Angenommen für: Informatik '97. Informatik als Innovationsmotor. 27. Jahrestagung der Gesellschaft für Informatik, Aachen 24.-26.9.1997

Nr. 42 (1997) Ferstl O.K., Sinz E.J., Hammel C., Schlitt M., Wolf S.: Application Objects – fachliche Bausteine für die Entwicklung komponentenbasierter Anwendungssysteme. Angenommen für: HMD – Theorie und Praxis der Wirtschaftsinformatik. Schwerpunkheft ComponentWare, 1997

Nr. 43 (1997): Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework - . Accepted for: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1997

Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using (SOM), 2nd Edition. Appears in: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1998

Nr. 44 (1997) Ferstl O.K., Schmitz K.: Zur Nutzung von Hypertextkonzepten in Lernumgebungen. In: Conradi H., Kreutz R., Spitzer K. (Hrsg.): CBT in der Medizin – Methoden, Techniken, Anwendungen -. Proceedings zum Workshop in Aachen 6. – 7. Juni 1997. 1. Auflage Aachen: Verlag der Augustinus Buchhandlung

Nr. 45 (1998) Ferstl O.K.: Datenkommunikation. In. Schulte Ch. (Hrsg.): Lexikon der Logistik, Oldenbourg-Verlag, München 1998

Nr. 46 (1998) Sinz E.J.: Prozeßgestaltung und Prozeßunterstützung im Prüfungswesen. Erschienen in: Proceedings Workshop „Informationssysteme für das Hochschulmanagement". Aachen, September 1997

Nr. 47 (1998) Sinz, E.J.:, Wismans B.: Das „Elektronische Prüfungsamt". Erscheint in: Wirtschaftswissenschaftliches Studium WiSt, 1998

Nr. 48 (1998) Haase, O., Henrich, A.: A Hybrid Respresentation of Vague Collections for Distributed Object Management Systems. Erscheint in: IEEE Transactions on Knowledge and Data Engineering

Nr. 49 (1998) Henrich, A.: Applying Document Retrieval Techniques in Software Engineering Environments. In: Proc. International Conference on Database and Expert Systems

Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460

Nr. 50 (1999) Henrich, A., Jamin, S.: On the Optimization of Queries containing Regular Path Expressions. Erscheint in: Proceedings of the Fourth Workshop on Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel, July, 1999 (Springer, Lecture Notes)

Nr. 51 (1999) Haase O., Henrich, A.: A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases. Erscheint in: Proceedings of the Third East-European Conference on Advances in Databases and Information Systems – ADBIS'99, Maribor, Slovenia, September 1999 (Springer, Lecture Notes in Computer Science)

Nr. 52 (1999) Sinz E.J., Böhnlein M., Ulbrich-vom Ende A.: Konzeption eines Data Warehouse-Systems für Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule" im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 6. Oktober 1999

Nr. 53 (1999) Sinz E.J.: Konstruktion von Informationssystemen. Der Beitrag wurde in geringfügig modifizierter Fassung angenommen für: Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, Hanser, München 1999

Nr. 54 (1999) Herda N., Janson A., Reif M., Schindler T., Augsburger W.: Entwicklung des Intranets SPICE: Erfahrungsbericht einer Praxiskooperation.

Nr. 55 (2000) Böhnlein M., Ulbrich-vom Ende A.: Grundlagen des Data Warehousing. Modellierung und Architektur

Nr. 56 (2000) Freitag B, Sinz E.J., Wismans B.: Die informationstechnische Infrastruktur der Virtuellen Hochschule Bayern (vhb). Angenommen für Workshop "Unternehmen Hochschule 2000" im Rahmen der Jahrestagung der Gesellschaft f. Informatik, Berlin 19. - 22. September 2000

Nr. 57 (2000) Böhnlein M., Ulbrich-vom Ende A.: Developing Data Warehouse Structures from Business Process Models.

Nr. 58 (2000) Knobloch B.: Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten.

Nr. 59 (2001) Sinz E.J., Böhnlein M., Plaha M., Ulbrich-vom Ende A.: Architekturkonzept eines verteilten Data-Warehouse-Systems für das Hochschulwesen. Angenommen für: WI-IF 2001, Augsburg, 19.-21. September 2001

Nr. 60 (2001) Sinz E.J., Wismans B.: Anforderungen an die IV-Infrastruktur von Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule 2001" im Rahmen der Jahrestagung der Gesellschaft für Informatik, Wien 25. – 28. September 2001

Änderung des Titels der Schriftenreihe *Bamberger Beiträge zur Wirtschaftsinformatik* in *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* ab Nr. 61

Note: The title of our technical report series has been changed from *Bamberger Beiträge zur Wirtschaftsinformatik* to *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* starting with TR No. 61

Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik

Nr. 61 (2002) Goré R., Mendler M., de Paiva V. (Hrsg.): Proceedings of the International Workshop on Intuitionistic Modal Logic and Applications (IMLA 2002), Copenhagen, July 2002.

Nr. 62 (2002) Sinz E.J., Plaha M., Ulbrich-vom Ende A.: Datenschutz und Datensicherheit in einem landesweiten Data-Warehouse-System für das Hochschulwesen. Erscheint in: Beiträge zur Hochschulforschung, Heft 4-2002, Bayerisches Staatsinstitut für Hochschulforschung und Hochschulplanung, München 2002

Nr. 63 (2005) Aguado, J., Mendler, M.: Constructive Semantics for Instantaneous Reactions

Nr. 64 (2005) Ferstl, O.K.: Lebenslanges Lernen und virtuelle Lehre: globale und lokale Verbesserungspotenziale. Erschienen in: Kerres, Michael; Keil-Slawik, Reinhard (Hrsg.); Hochschulen im digitalen Zeitalter: Innovationspotenziale und Strukturwandel, S. 247 – 263; Reihe education quality forum, herausgegeben durch das Centrum für eCompetence in Hochschulen NRW, Band 2, Münster/New York/München/Berlin: Waxmann 2005

Nr. 65 (2006) Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet

Nr. 66 (2006) Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, Karsten Loesing, and Guido Wirtz: Concealing Presence Information in Instant Messaging Systems, April 2006

Nr. 67 (2006) Marco Fischer, Andreas Grünert, Sebastian Hudert, Stefan König, Kira Lenskaya, Gregor Scheithauer, Sven Kaffille, and Guido Wirtz: Decentralized Reputation Management for Cooperating Software Agents in Open Multi-Agent Systems, April 2006

Nr. 68 (2006) Michael Mendler, Thomas R. Shiple, Gérard Berry: Constructive Circuits and the Exactness of Ternary Simulation

Nr. 69 (2007) Sebastian Hudert: A Proposal for a Web Services Agreement Negotiation Protocol Framework . February 2007

Nr. 70 (2007) Thomas Meins: Integration eines allgemeinen Service-Centers für PC-und Medientechnik an der Universität Bamberg – Analyse und Realisierungs-Szenarien. Februar 2007

Nr. 71 (2007) Andreas Grünert: Life-cycle assistance capabilities of cooperating Software Agents for Virtual Enterprises. März 2007

Nr. 72 (2007) Michael Mendler, Gerald Lüttgen: Is Observational Congruence on μ-Expressions Axiomatisable in Equational Horn Logic?

Nr. 73 (2007) Martin Schissler: to be announced

Nr. 74 (2007) Sven Kaffille, Karsten Loesing: Open chord version 1.0.4 User's Manual. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 74, Bamberg University, October 2007. ISSN 0937-3349.

Nr. 75 (2008) Karsten Loesing (Hrsg.): Extended Abstracts of the Second Privacy Enhancing Technologies Convention (PET-CON 2008.1). Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 75, Bamberg University, April 2008. ISSN 0937-3349.

Nr. 76 (2008) G. Scheithauer and G. Wirtz: Applying Business Process Management Systems? A Case Study. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 76, Bamberg University, May 2008. ISSN 0937-3349.

Nr. 77 (2008) Michael Mendler, Stephan Scheele: Towards Constructive Description Logics for Abstraction and Refinement. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 77, Bamberg University, September 2008. ISSN 0937-3349.

Nr. 78 (2008) Gregor Scheithauer and Matthias Winkler: A Service Description Framework for Service Ecosystems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 78, Bamberg University, October 2008. ISSN 0937-3349.

Nr. 79 (2008) Christian Wilms: Improving the Tor Hidden Service Protocol Aiming at Better Performancs. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 79, Bamberg University, November 2008. ISSN 0937-3349.

Nr. 80 (2009) Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger, Guido Wirtz: QoS Enabled B2B Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 80, Bamberg University, May 2009. ISSN 0937-3349.