MAGISTERARBEIT

Probing and Monitoring of WSBPEL Processes with Web Services

Ausgeführt am

Institut für Softwaretechnik und interaktive Systeme
der Technischen Universität Wien

Unter der Anleitung von

o. Univ. Prof. Dr. A Min Tjoa
und Dr. Josef Schiefer

durch

Heinz Roth

Lohorn 6
A-6911 Lochau

24. April 2006

Abstract

In the battle for prices, market shares and always shorter product lifecycles, the automation of business processes to optimise the daily business of a company receives ever-growing attention. Nevertheless, it still exhibits great potentials for further promising developments in application support using information technology.

On the other hand, tracking the achievement of business goals, objectives and strategies in terms of controlling is increasingly used to measure and adjust the outcome of these processes.
Interestingly enough, the former can facilitate the realisation of the latter by providing unaltered audit information in real-time, thus, allowing for ex-post evaluation and, moreover, real-time analysis of a process, leaving the door open for process-intervention at runtime.

The Web Services Business Process Execution Language (WSBPEL) already is one of the widely accepted technologies in this domain of business process automation and will be studied in depth together with Business Process Management (BPM) and controlling in general.

One of the main contributions of this thesis is an approach for the transformation of a WSBPEL model into an auditable model which can be used for process monitoring purposes. The empirical part presents a platform independent solution with the ability to extract audit information of any WSBPEL compliant process – a powerful feature of business process engines still in the need of standardisation.

Acknowledgements

Zunächst möchte ich mich für die fachliche Unterstützung bei der Erstellung der Magisterarbeit bei meinen Betreuern O. Univ. Prof. Dipl. Ing. Dr. A Min Tjoa und bei Dr. Josef Schiefer bedanken, die es verstanden mir das Thema auf interessante Art und Weise näher zu bringen und mir jederzeit mit Rat und Tat zur Seite standen.

Weiters danke ich meinen Eltern Josef und Ida sowie meiner gesamten Familie da mein Studium erst durch sie möglich wurde.

Von ganzem Herzen bedanke ich mich auch bei dir Natascha. Während den sämtlichen Jahren die wir uns durch unsere Studienzeit begleitet haben hast du mir die Motivation und Kraft gespendet die ich brauchte.

Contents

List of Figures

List of Tables

List of Sources

1 Introduction

"A wealth of information creates a poverty of attention."
Herbert Simon

Controlling is an essential element in today's corporate culture. It facilitates the decision making on and the execution of sound business strategies.

The move away from traditional business process measurement instruments fulfilling the tasks of controlling, as argued by NEELY as well as BROADBENT, requires new methods for and perceptions of the integration of controlling within the business structure.

To some extent, the radical change of the company operational and organization structure of most businesses from functional divisions to process oriented teams during the last few decades can be regarded as a main reason for this change.

The manifold developments in information technology accelerate the progress in business management even more. At the same time, those advancements feature some of these newly required and sophisticated methods.

One of these new technologies is the Web Services Business Process Execution Language (WSBPEL), an OASIS standard used to model business processes in a formal manner. Afterwards, the devised and platform independent process definition can be executed by any so called WSBPEL process engine. Even though WSBPEL is an open standard, the various existing process engines differ greatly in terms of additional included features, among which auditing plays an important role.

To familiarise the reader with the different topics, the first chapters introduce the subjects mentioned above.

Business Process Management (BPM) together with a characterization as well as a lifecycle model of business processes are presented in chapter 2.

In chapter 3, WSBPEL is discussed as one representative of an emerging technology for business process execution. A brief overview of the web service stack on top of which WSBPEL is operating is included as well.

Chapter 4 points out the reasons and importance of controlling within businesses and studies how controlling is integrated within the business strategy.

Finally, the empirical part of this thesis presents a platform independent and WSBPEL standard compliant solution for auditing of WSBPEL processes in chapter 5. A simple prototype is used to extend a given WSBPEL process definition, taking advantage of an auditing web service which will accept audit information about each monitored process activity.

2 Business Process Management (BPM)

This chapter aims at giving a short introduction to the basic ideas of BPM although it is assumed that the reader is already familiar with the history and the basic concepts of BPM. Therefore, the following sections do not seek any completeness in describing BPM, but try to emphasize on the auditing aspect in this domain.

The main focus lies on a characterization of business processes and each phase of a business process lifecycle. A brief introduction on the origin of process organizations will be given beforehand.

2.1 Process Organizations

For decades, businesses tried to optimize the production of their goods by increasing worker productivity. Industrialization not only achieved this goal through the growing use of machines, but also by fragmenting work into simple pieces, the division of labour. ADAM SMITH gave three important advantages of his ideas resulting in significant economies of scale which he already presented in 1776:

- Increase dexterity of workers
- Save the time needed to move from one working step to another
- Vast possibilities to use machines for a single piece of work

Henry Ford was one of the most prominent entrepreneurs to apply this approach in form of the moving assembly line. Employees in his factory specialized to such an extent, that they only were trained to install exactly one part. The simplicity of their jobs made it easy to train new personnel in a short time and to optimize every step of the overall manufacturing process.

Unfortunately, the task to coordinate the production process as a whole became more complex with every decomposition made. Therefore, Alfred Sloan used the same approach, i.e. division of labour, to change the way managers did their jobs. He created smaller decentralized divisions supervised and controlled by a single manager, thus decreasing the complexity of their tasks.

The reason why the organizational structure of many companies only recently changed rapidly from these functional divisions to process oriented teams has many facets.

HAMMER & CHAMPY tried to persuade especially the american industry to recognize this change when they published their manifesto *Reengineering the Corporation* which called for a *business revolution* in the US. They named three main factors that required this new approach: customers, competition and change.

HAMMER & CHAMPY wanted companies to become more flexible and innovative to overcome their shortcomings and adjust to the changed environment. They argued that the huge bureaucracy, which has been created through decentralization of management, although it was needed to bridge the various gaps between the different divisions, resulted in a vacuum between the customer and the company because there simply was no connection between decision makers of the company and their customers anymore.

This customer focus also reflects in their definition of a business process:

> *"We define a business process as a collection of activities that takes one or more kinds of input and creates an output that is of* ***value to the customer.****"* [1]

DAVENPORT offered a more detailed definition for business processes:

> *"[…] a process is simply a structured, measured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what.*
> *A process is thus a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs: a structure for action."* [2]

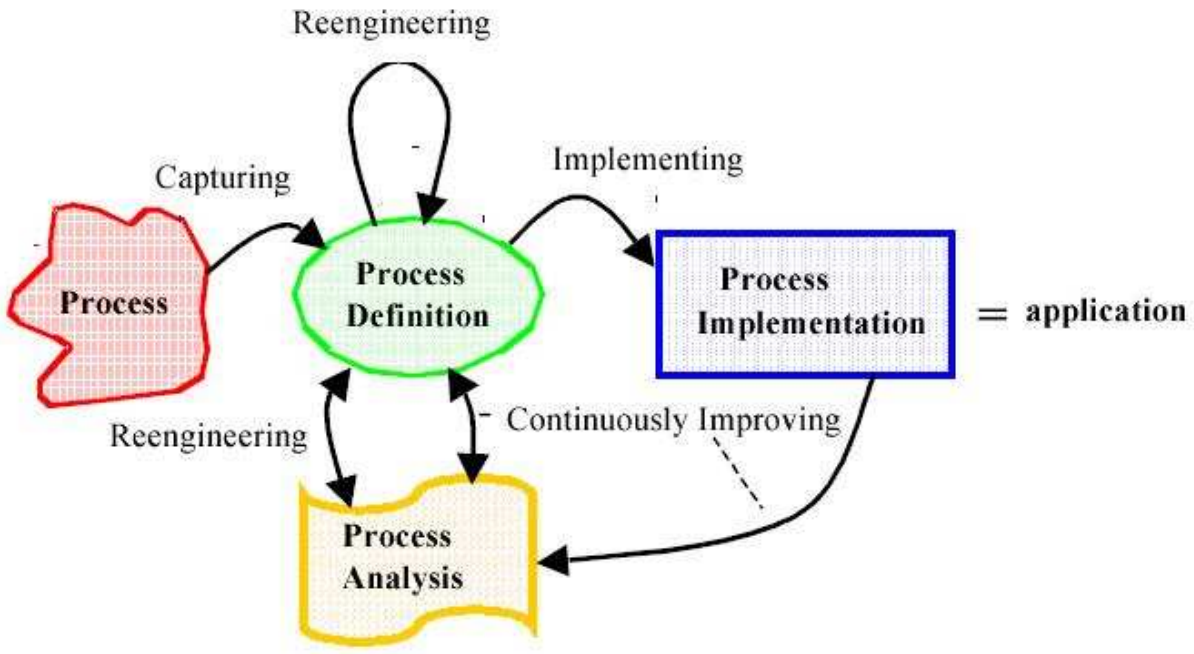
For a company, to apply Business Process Reengineering (BPR) means to change its way of doing business. The driving force behind every process should always be the goals of the process and its value to the customer.

2.2 Business Process Lifecycle

Companies devoted themselves to BPM for over two decades now to improve the quality of their products and satisfy their customers' needs. Since then, several lifecycle models have been developed to describe and control these efforts.

Traditional approaches to business process lifecycle management like the one proposed by GEORGAKOPOULOS & TSALGATIDOU distinguish four phases as shown in Figure 1:

- Capturing: To capture a process one has to understand it (interviews, experts, etc.)
- Reengineering: Radical top down process redesign guided by business objectives
- Implementing: Realizing & automating a process using information systems
- Continuous Improvement: Measure & analyse process execution, make small corrections



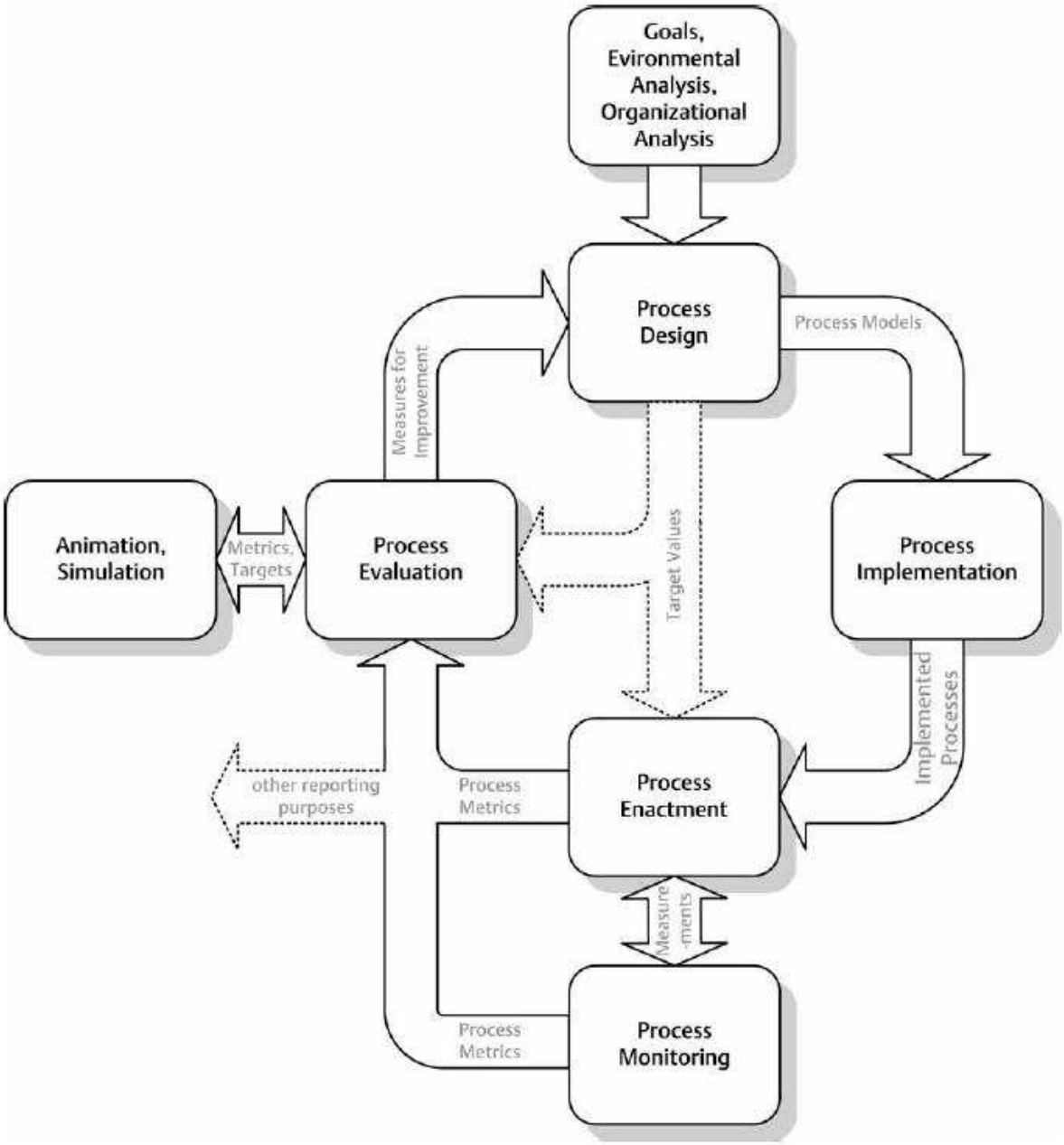
Source: Georgakopoulos & Tsalgatidou, 1998, Figure 3.

Figure 1: Traditional Business Process Lifecycle Management.

[1] Compare Hammer & Champy, 1993, p. 35.
[2] Compare Davenport, 1993, p. 5.

ZUR MÜHLEN criticizes the lack of separation between the technical implementation of the required infrastructure and the conceptual design of designed models in these traditional approaches. Furthermore, he is missing monitoring aspects and the possibility to manipulate and adjust already running process instances.

Therefore, ZUR MÜHLEN presents a new process management lifecycle shown in Figure 2 which combines and extends earlier models. It outlines the importance of a monitoring phase during process enactment and takes gained information into consideration.



Source: zur Mühlen, 2004, Figure 2-17.

Figure 2: Process Life Cycle

In the following, the various phases of the business process lifecycle will be presented briefly to provide a framework in which the topics discussed in the remaining chapters can be positioned to give the reader a better overview.

2.2.1 Process Analysis

If a company is about to capture its business processes for the first time, or it has moved away from an original process design too far, that it is preferable to start from scratch than to work with the already defined process model, the first step is to understand and analyse the process behind the business.

This analysis usually works with a bottom up approach where the people executing the *process* are interviewed and/or studied at work or (external) experts are questioned to gain information about how the process is working and what alternatives exist to reach the goals of this specific process.

Furthermore, it's very important to define the goals and the value for the customer associated with the observed efforts as well as possible changes to these targets in the near future.

Finally the company's organizational structure and possibilities to change it need to be examined as well as existing external influences which need to be taken into consideration before the way of doing business can be changed.

2.2.2 Process Design

After the goals and circumstances under which a process can be performed have been analyzed, the *process model*, which is an abstraction of the real world business process, can be designed. During the design phase, the resources, their responsibilities and the process structure are defined.

GEORGAKOPOULOS & TSALGATIDOU, who compared technologies and tools for business process lifecycle management, distinguished different levels of abstraction depending on the intended use of the captured and/or designed process. If a process is to be automated and executed by an information system, the required granularity and completeness of the process description will be very different from one used on management level for business process reengineering.

> *"The process abstraction level in a definition depends on the intended use of the definition. For example, a definition may describe a process at the highest conceptual level necessary for understanding, evaluating, and redesigning the process. On the other hand, another definition may describe the same process at a lower-level of detail required for performing process implementation."* [3]

The result of the design phase is a process model of the captured process. There exist several tools providing support during this stage, for instance the *Architecture of Integrated Information Systems* (ARIS) model, a framework for analysing and designing information systems, which can be used among other things to model business processes.

2.2.3 Process Implementation

At this point, the process has been fully captured and designed. The goals, the participants, the abstract structure of the process and the targeted values during its execution have been defined.

During this next phase, the required infrastructure together with the so called *process engine* is chosen and the designed process is implemented to work in this runtime environment.

---

[3] Compare Georgakopoulos & Tsalgatidou, 1998, section 3.1.

One of the languages to describe businesses processes in a formal manner which is evolving as a business standard in this domain is the Web Services Business Process Execution Language (WSBPEL). For now, it's sufficient to note that a process described with WSBPEL consists of activities which can be arranged in sequential, parallel or conditional order. Process models described with WSBPEL can be executed on any compliant engine[4].

Please refer to chapter 3 for more details on WSBPEL and to section 5.1 for an overview of a commercial WSBPEL engine.

2.2.4 Process Enactment

Whenever a specific business process is required, an individual *process instance* is derived from the deployed process model. The process engine manages the execution of the process instance, notifying the process participants following the defined process structure and providing them with the necessary information units until the process completes.

Of course, exceptional situations may occur during process enactment. Processes can fault, terminate abnormally or remain in a deadlock, although perfectly designed process model already covers these behaviours as good as the process enacting infrastructure allows it and reacts accordingly.

Whether the process instance finishes smoothly or with an error, *process metrics* collected from executed processes provide useful information for *process owners* which can be used during process evaluation.

2.2.5 Process Monitoring

In contrary to *process metrics* collected after a process instance executed, the data derived during this phase represents real-time information on the state of process instances and the process engine itself.

A lot of possibilities arise from process monitoring such as

- Alert business users in real-time and provide context [5]
- Real-time indicators about the current process performance
- Automatic discovery and analyses processes situations or exceptions and creation of reactive and proactive responses such as generating early warnings, preventing damage, loss or extensive cost, exploiting time-critical business opportunities, or adapting the business process with minimal latency.
- …

Unlike the approach used by ZUR MÜHLEN, which deals with the ex post analysis of process metrics, the empirical part presented in chapter 5 focuses exactly on these possibilities and how this monitoring aspect can be added to WSBPEL without relying on proprietary extensions to this language.

Please refer to section 4.3 for more information on application support for business process auditing.

---

[4] Compare Wikipedia contributors, 2005, chapter 3.

[5] Compare Hellinger & Fingerhut, 2002, The Simple Anatomy of the Decision.

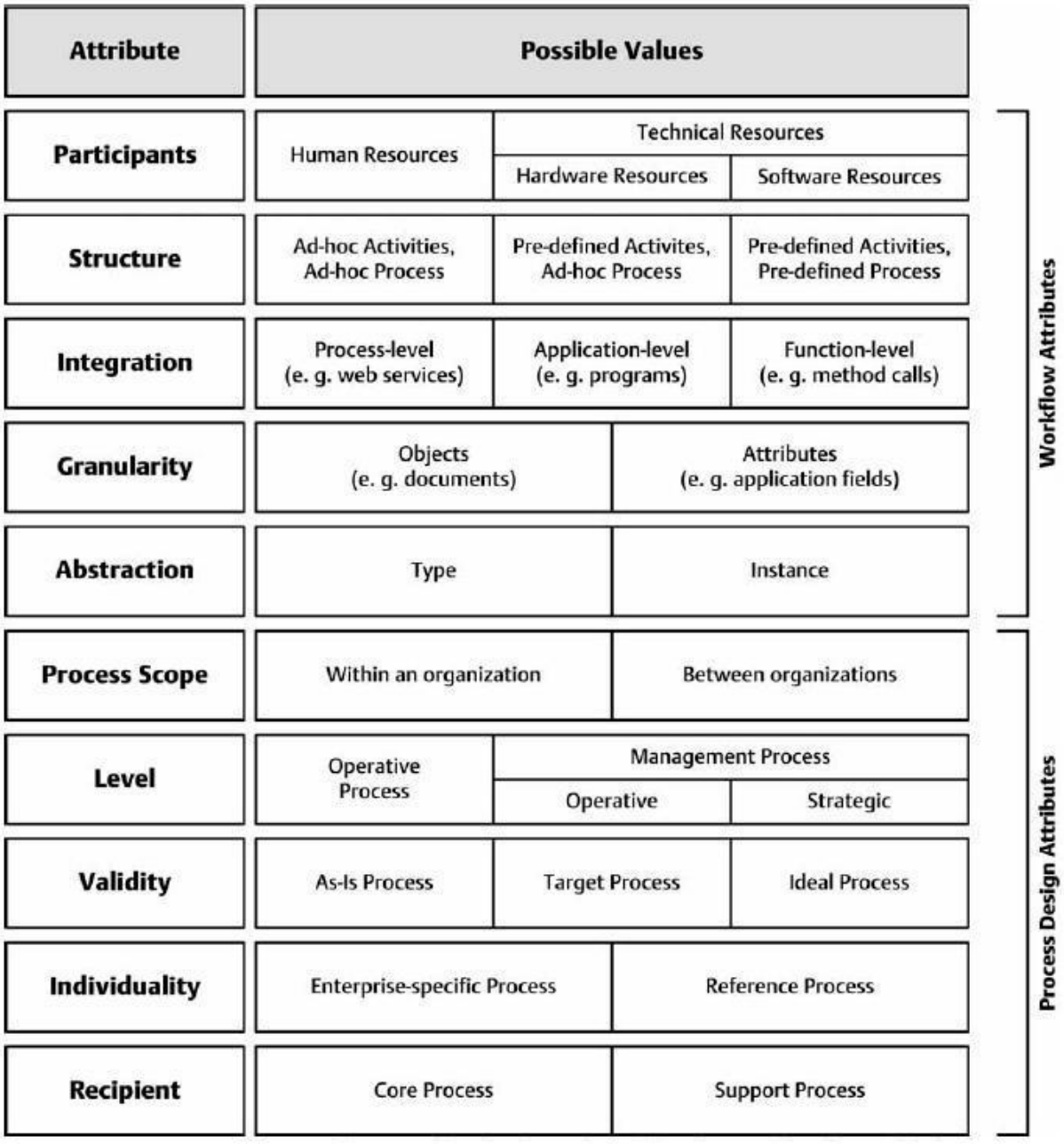
2.2.6 Process Evaluation

The ex-post evaluation of a number of executed process instances derived from one process model closes the BPM lifecycle. The statistics generated by this analysis provide the foundation for business decisions to be made as well as for possible improvements of the process model itself.

As it was stated in the last section, ZUR MÜHLEN deals in great detail with the controlling aspect during this phase.[6]

2.3 Characterizing Business Processes

A detailed characterization of a business process itself is given by ZUR MÜHLEN which is shown in Figure 3 and will be discussed in the following sections. It suggests that processes with the same attributes can be grouped together to analyze their requirements more efficiently.



[6] Compare zur Mühlen, 2004.

Source: zur Mühlen, 2004, Figure 2-6.

Figure 3: Classification of Process Attributes.

2.3.1 Process Participants

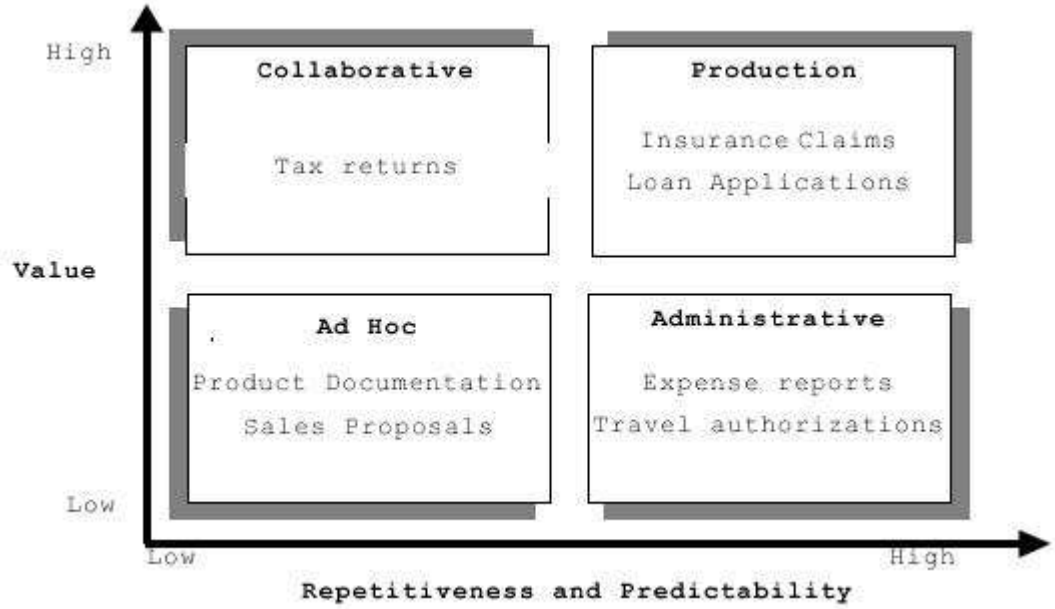
Two different types of process participants are distinguished: humans and technical resources. From the point of view of business process, the difference lies within the possible communication methods.
While it can assume to get an immediate answer from an information system most of the time (*push*), it may need to wait for humans to actively request information about the process (*pull*).

2.3.2 Structure

Capturing every aspect of the work done by a company to produce its goods as a business process could quickly result in a lot of overhead for processes which do not occur on a frequent basis or if the structure of the process is unknown until it is executed.

GEORGAKOPOULOS & TSALGATIDOU distinguish four kinds of processes which are compared in Figure 4.

While ad hoc processes are negligible and not worth the effort to be automated, collaborative processes are performed by an expert team where the ordering of activities is controlled and coordinated by humans. [7] Administrative and production processes on the other hand offer important attributes like predictability and repetitiveness. Therefore, processes belonging to these types will be more likely to be successfully captured and automated.



Source: Georgakopoulos & Tsalgatidou, 1998, Figure 2.

Figure 4: Characterization of Processes.

2.3.3 Integration

Similar to the concept of loose and tight coupling, the integrity dimension refers to the extent surrounding systems, applications or methods are accessed by the process. Depending on the knowledge about and the permissions for these systems of the process itself, it may only be able

[7] Compare Georgakopoulos & Tsalgatidou, 1998, chapter 2.

to nudge another process in an interorganizational environment whereas it may be able to make explicit function calls within the boundaries of its own business.

ZUR MÜHLEN distinguishes between *process-level integration* on the one hand, *application-level integration* if entire applications are triggered and *function-level* integration on the other hand, mostly used in technical processes.

2.3.4 Granularity

The granularity of a process is actually an attribute of the (data) objects which are passed between the process participants, ranging from a simple data type like a customer id to a solid product. Thus, the granularity also relates to the participants (human or application) and the abstraction level of the process.

2.3.5 Abstraction

A *process type* refers to the general definition of a process which serves as a template for a particular *process instance* of such a process type.

2.3.6 Process Scope

This attribute is very important when capturing a process. If a process is interorganizational, part of the work when capturing the process is to define the interfaces and to understand, that there may only be limited control on the execution and sometimes even the design of the process.

2.3.7 Level

The hierarchy level of a process indicates whether the goals of a process are found in *management* issues or in the *operative business*. Management processes can further be distinguished in *operative* and *strategic processes*, depending on their time horizon.

2.3.8 Validity

The validity of a process indicates the *quality* of a process. ZUR MÜHLEN distinguishes three different qualities.
An *as-is process* is a representation of an existing process within an organization. The opposite would be an *ideal process*, which corresponds to the ultimate design of a process. Unfortunately cost issues, organizational boundaries or other constraints can be obstacles for the implementation of a process. If processes within an organization are going to be changed or implemented, a *target process* is used as a draft to guide the efforts of changing the current process.

2.3.9 Individuality

The individuality of a process indicates the application area of a process design. The lowest degree of individuality is given by *reference processes* which address a problem domain shared between many different companies. By using and adjusting these kinds of processes, an *enterprise-specific process* may be created. Enterprise-specific processes are only applicable within the specific company they were designed for.

2.3.10 Recipient

The simple difference between the *core processes* and *support processes* shown in Figure 3 is the fact, that core processes create value (to the customer) while support processes are needed to successfully execute the core processes.

3 Web Services Business Process Execution Language (WSBPEL)

The term WSBPEL originates from the efforts of the *Organization for the Advancement of Structured Information Standards* (OASIS) which continued the work of IBM and Microsoft and several other parties to combine their proprietary languages for executing business processes. Those companies also created the current version of the WSBPEL specification. At the time of writing, the term *Business Process Execution Language for Web Services* (*BPEL4WS*) was used and sometimes still is used together with its abbreviation *BPEL*.

The term WSBPEL will be used generally in this thesis although it should be noted that the OASIS technical committee, which decided on this new term during a vote in 2004, indicated significant changes between BPEL4WS 1.1 and the WS-BPEL 2.0 specification that will sooner or later become the new standard.

Basically, WSBPEL is a block structured XML based flow language to describe business processes that consists of activities as its basic components. Activities can be arranged in a sequential order as well as in a parallel, synchronized way and their execution can result in a nondeterministic outcome.
As indicated by the name WSBPEL itself, web services play the most important role as far as interactivity with other systems is concerned, i.e. the communication to and from other entities is done by web services.

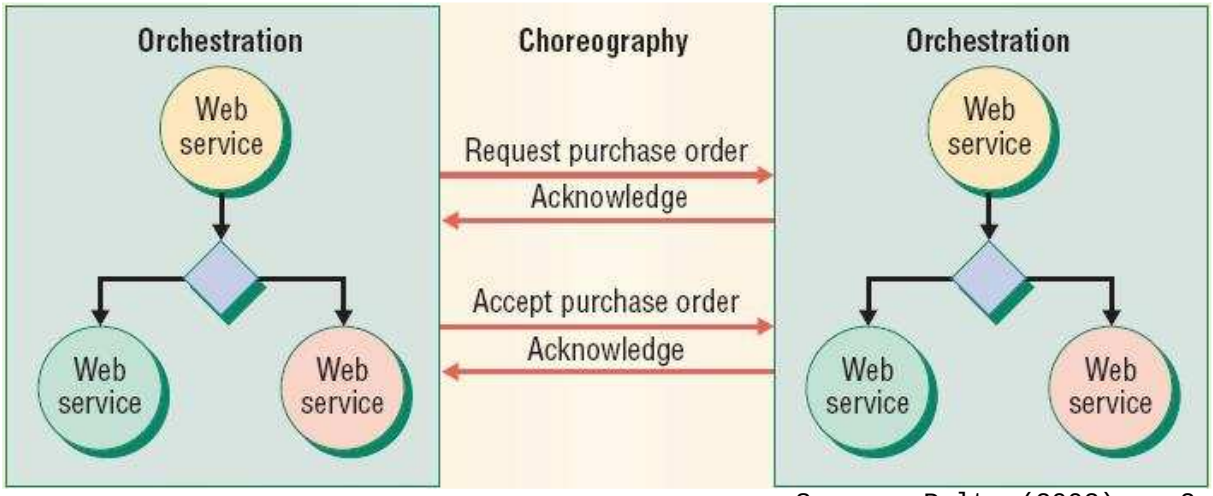
A short but meaningful definition by JORDAN & EVDEMON describes the main use of WSBPEL to be:

> *"Enabling users to describe business process activities as web services and define how they can be connected to accomplish specific tasks"* [8]

More technical speaking, WSBPEL can be used to specify the chronological order of web service operations in a long running business transaction, opposed to tasks that only require single web service requests.

The WSBPEL specification includes *abstract* and *executable processes*, where abstract processes are referred to as business protocols, describing the visible behaviour and exchange of messages between two or more business process participants (*choreography*) whereas ANDREWS ET AL. describe an executable processes as a model for the actual behaviour of one of those participants (*orchestration*).[9] Thus, orchestration means an executable process that is described and controlled by a single endpoint which is then known as a *workflow*, while choreography involves multiple parties and describes the publicly observable exchange of messages, as indicated in Figure 5.

[8] Compare Jordan & Evdemon, 2005.
[9] Compare Andrews et al., 2003, abstract.



Source: Peltz (2003), p.2.

Figure 5: Orchestration vs. Choreography

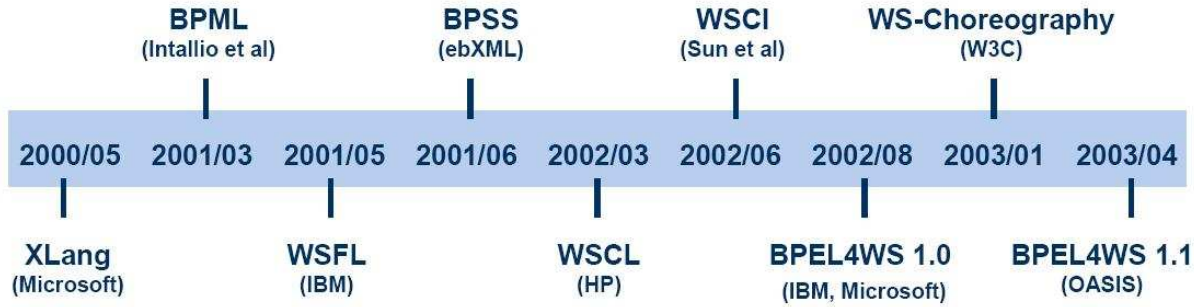
One important aspect of WSBPEL is the fact that business processes defined in WSBPEL should be fully executable and portable between WSBPEL- compliant engines.

In this subchapter the various aspects and concepts of the Web Services Business Process Execution Language (WSBPEL) will be described. First, the recent history of the WSBPEL specification[10] (version 1.1) is examined. In the following sections, a detailed presentation of core concepts and basic elements as well as an overview of the basic structure of a WSBPEL process will be given.

3.1 History

As mentioned earlier, WSBPEL emerged from a fusion of various other existing specifications and standards which addressed the domain of web service orchestrations. According to ANDREWS ET AL., two standards from Microsoft and IBM, namely XLANG and WSFL, exerted most influence on the creation of WSBPEL.

Both, XLANG and WSFL, are similar as both are built on top of WSDL. According to KRAMLER & RETSCHITZEGGER, there are a number of major differences to WSFL concerning behavioural and transactional, functional, and organizational characteristics.



SOURCE: Kavantzas & Lehmannn (2003).

Figure 6: Recent History of Business Process Standards.

Microsoft started to develop XLANG to describe the flow of a business process using sequential, parallel and conditional activities back in the year 2000. XLANG was designed to work with and

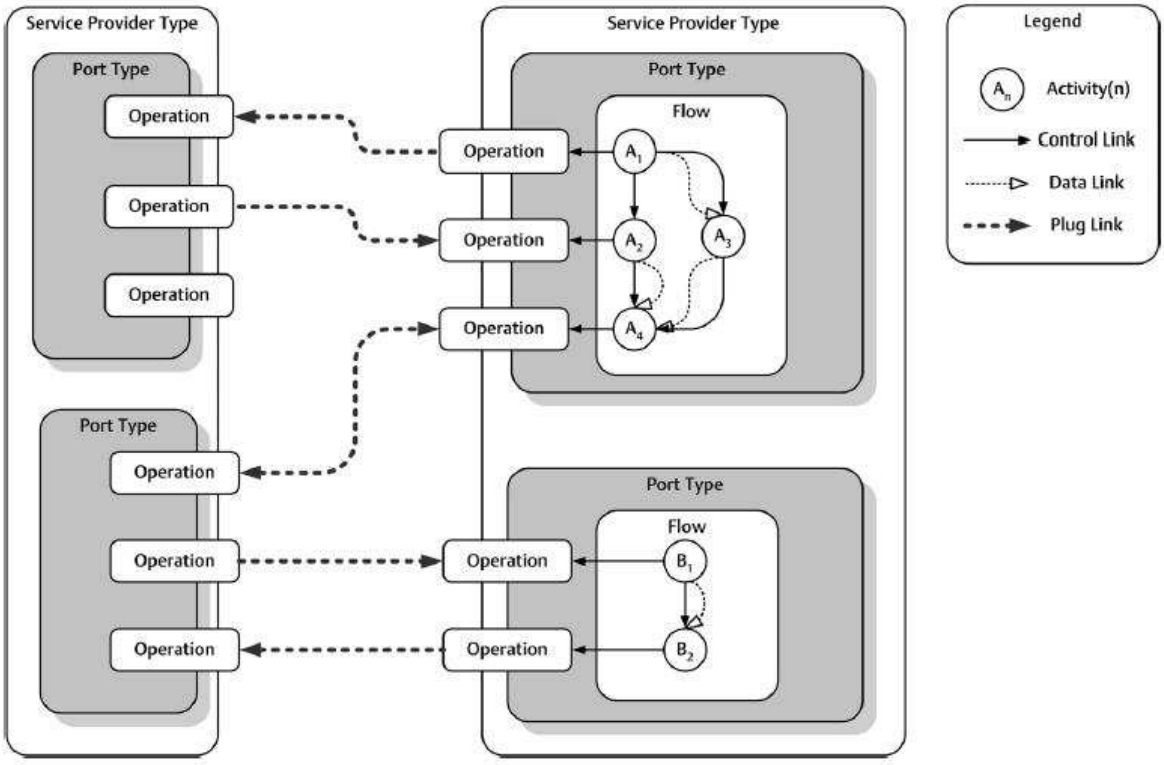
[10] Compare Andrews et al., 2003.

is stilled used by their BizTalk Server Series. THATTE outlined the key factors for this initiative and the difference to workflow management systems as follows:

> *"Enterprise workflow systems today support the definition, execution and monitoring of long-running processes that coordinate the activities of multiple business applications. But they do not separate internal implementation from external protocol description. When processes span business boundaries, loose coupling based on precise external protocols is required because the parties involved do not share application and workflow implementation technologies, and will not allow external control over the use of their backend applications. Such protocols are by necessity message-centric […] "*[11]

The XLANG *schedule* is the main element and is composed of several *taks*. Similar to the concept of compensation in WSBPEL (see section 3.2.3.14) it is possible to group multiple activities to a *transaction*, for which compensating activities can be defined in case of an error during the transaction.

IBM presented the Web Services Flow Language (WSFL)[12] in 2001 which is used to describe both the private (referred to as *usage pattern* or *flow model*) and the public elements of a process (referred to as the *global model*).[13]



Source: Zur Mühlen (2004), p.155.

Figure 7: Schematic WSFL Flow.

A so called WSFL *flow* is defined by *data links* and *control links* while information about the global model is available through *plug links* as shown in Figure 7.

[11] Compare Thatte, 2001, chapter 3.
[12] Compare Leymann, 2001.
[13] Compare zur Mühlen, 2004, p.154.

In 2002, Microsoft, IBM, and BEA superseded XLANG and WSFL with their release of the BPEL4WS specification version 1.0; in May 2003 they released the current version 1.1 together with SAP and SIEBEL.

In 2003, OASIS picked up the development after the involved companies agreed to hand over the last specification under royalty-free terms to a new OASIS technical committee, whose members are still working on a final draft.

3.2 How WSBPEL Works

This section introduces the basic concepts and the enabling technologies of WSBPEL. Web services play an important role in the underlying layers on top of which WSBPEL is located, as shown in Figure 8, whereas WSBPEL provides the constructs to capture a business process in a formal way, resulting in a process definition where each activity is implemented as a web service that conducts the actual work.

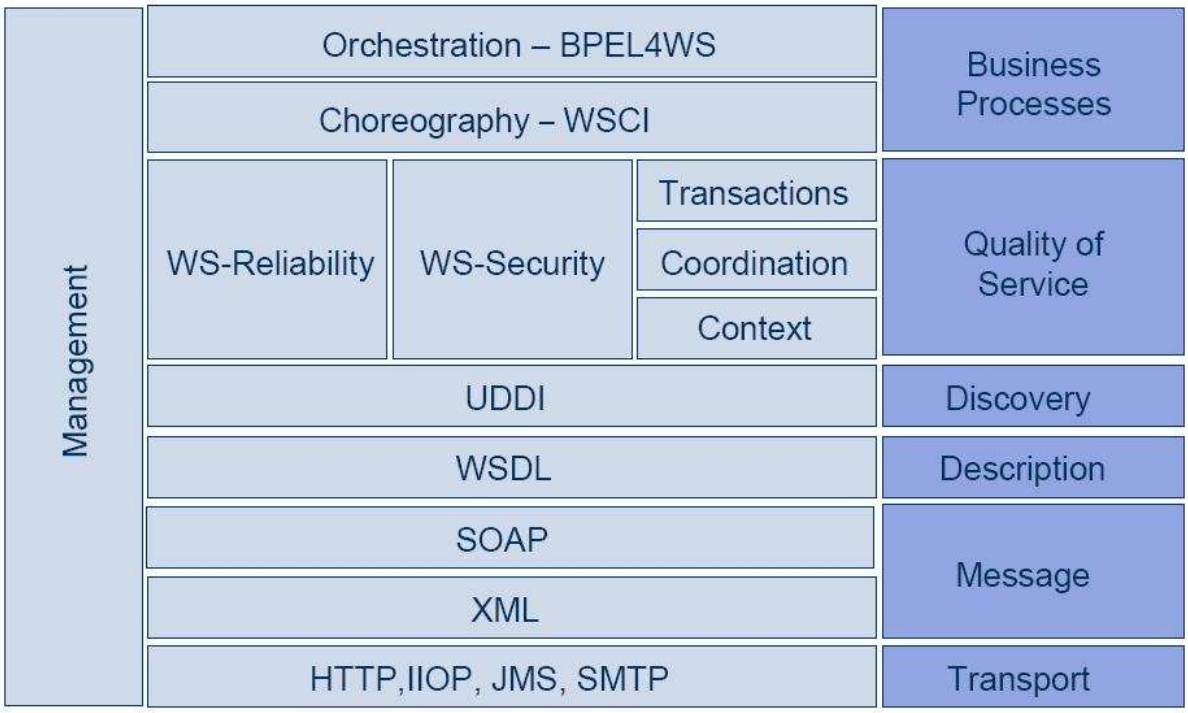


Figure 8: WSBPEL on top of the web service stack.

Therefore a short overview of the web service stack is given in section 3.2.1. Afterwards, the basic concepts and notions of WSBPEL are described in section 3.2.2. Examples of actual WBSPEL processes can be found in the chapter 5.

3.2.1 How Web Services Support WSBPEL

3.2.1.1 Web Service Choreography Interface (WSCI)

Figure 8 should not be misunderstood in that way, that WSCI is always needed by WBSPEL to orchestrate web services, although WSCI may still be used in a complementary way to describe choreographic aspects of the related web services on a lower layer.

To understand the relationship between WSCI and WSBPEL one has to look at it from two different perspectives, from a technical and a political one.

On the one hand, the scope of these two technologies can be explored. WSCI focuses on defining the observable behaviour of a web service offered by a single participant, called *WSCI interface*, as well as describing the collective message exchange among interacting web services[14], which is referred to as the *global model*. WSCI explicitly does not address the implementation of these internal processes.

WSBPEL mainly focuses on executable business processes, thus on the orchestration of web services as it is seen from one of the participant's perspective. It was already mentioned in the introducing paragraphs of this chapter that WSBPEL additionally supports, besides executable processes, another type of processes called abstract processes. Abstract processes can be used to define the choreography of web services with WSBPEL which is then known as a business protocol.

The overlapping of these two competing technologies resulted in a tension between the supporters of each one of these, with BEA Systems, IBM, Microsoft, SAP and Siebel Systems on the side of WSBPEL and BEA Systems, Intalio, SAP and Sun Microsystems on the side of WSCI, which is still not resolved.

3.2.1.2 Web Services Reliability (WS-Reliability)

Because web services are based upon unreliable transport protocols such as HTTP, a new layer was needed to cope with the problems of failed message deliveries. The first version of the WS-Reliability specification was released in 2003 by a group of major IT vendors and later on transferred to the same organizational body that works on the advancement of WSBPEL, namely OASIS.

In 2004, OASIS ratified the current version 1.1 of the WS-Reliability specification[15], which supports one or more of the following core features

- Guaranteed delivery at least once
- Duplicate elimination
- Guaranteed message ordering

WS-Reliability is a necessary and important attribute of web service communication since it is absolutely mandatory to discover and recover from such messaging errors in business applications.

3.2.1.3 Web Services Security (WS-Security)

Since web services offer a comfortable way to exchange data via the public internet, security aspects have to be taken into consideration. Per se, web services offer no possibility to cipher messages or exchanging the necessary security tokens. Therefore, each application with such security needs that wants to use web services for communication would have to cope with these issues again and again.

In October 2001 Microsoft announced their first version of a WS-Security specification which was a SOAP extension in order to provide three independent features:

- Credential exchange
- Message integrity

[14] Compare Arkin et al., 2002, Abstract.
[15] Compare Iwasa et al., 2004.

- Message confidentiality

After combining their efforts, Microsoft, IBM and VeriSign released another version of the WS-Security specification, which was submitted to OASIS for further development and became a new OASIS standard in 2004.

3.2.1.4 WS-Coordination, WS-Transactions, and WS-Context

As the term *coordination* already suggests, WS-Coordination is about one coordinator supervising the actions of a distributed system. The WS-Coordination specification proposed by Cabrera et al. describe the main idea of it to be

> *"[…] an extensible framework for providing protocols that coordinate the actions of distributed applications. Such coordination protocols are used to support a number of applications, including those that need to reach consistent agreement on the outcome of distributed activities."* [16]

The framework is based on the SOAP and WSDL extensibility model and consists of three components

- An activation service to create a coordination instance
- A registration service for applications to register for coordination
- A coordination type

Coordinated applications share a common context known as the *coordination context* to create a logical connection between the different applications and exchange information on activities. Because various domains exists that have a need for coordination of the distributed execution of their activities the required context model can vary enormously. WS-Coordination aims at being a generic, reusable framework through the use of coordination types which define the format of the WS-Coordination context. These context types are used to adapt the whole coordination protocol to meet the needs of a specific problem domain. According to According to LITTLE & WEBBER[17], they are usually an extension provided by third-party vendors and consist of the following parts:

- A coordination identifier with guaranteed global uniqueness for an individual coordinator in the form of a URI
- An address of a registration service endpoint where parties receiving a context can register participants into the protocol
- A time-to-live value that indicates for how long the context should be considered valid
- Extensible protocol-specific information particular to the actual coordination protocol supported by the coordinator

Since WS-Coordination is a generic framework, it also means that it is useless without proper adaptation, which happens in the fourth of these four points.

3.2.1.5 Universal Description, Discovery and Integration (UDDI)

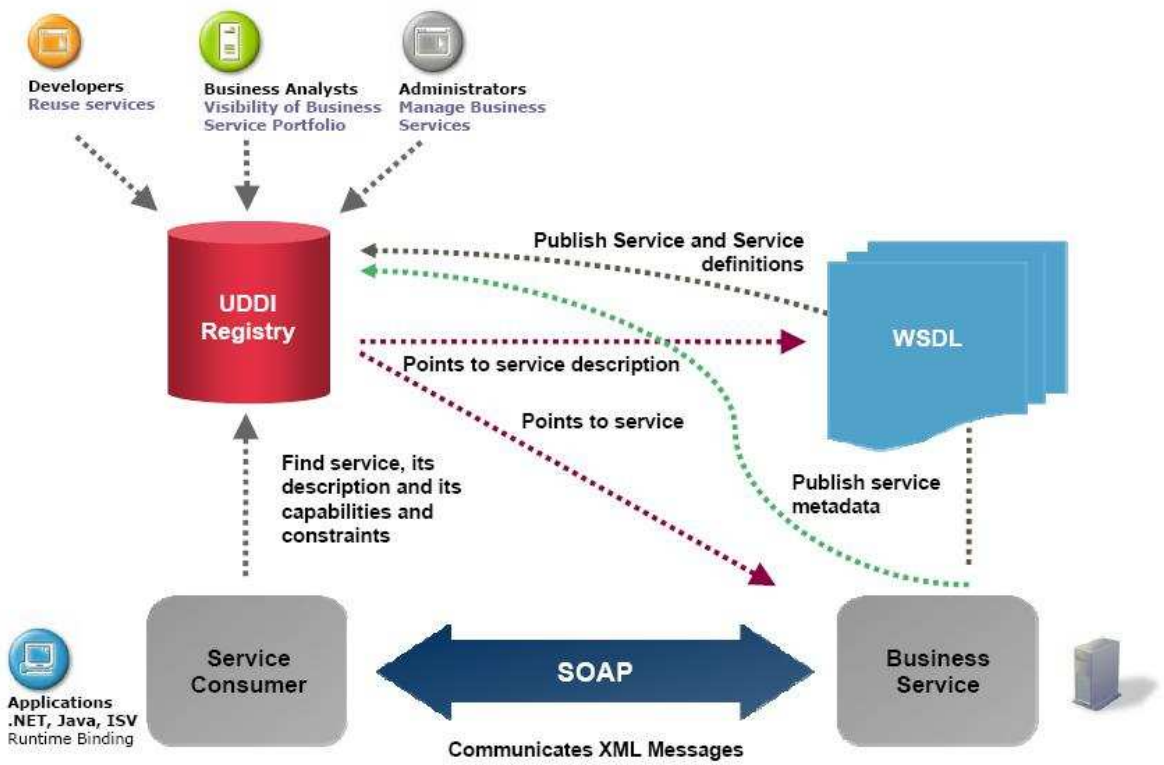
Although the dream of the internet industry which thought about UDDI as a all-in-one device suitable for every purpose when it comes to dynamically finding the perfect web service didn't become reality, UDDI still plays a major role among web service supporting technologies.

---

[16] Compare Cabrera et al., 2005, abstract.
[17] Compare Little & Webber, 2003, Context.

In the beginning, UDDI was praised to be become sort of an *e-business directory*[18] where businesses could advertise their services and clients would find them just like in a telephone directory with which it has often been compared. Only recently, OASIS has approved UDDI version 3.0 as an OASIS standard which is nowadays more down-to-earth referred to as a web service infrastructure.

The two most important use cases of UDDI are publishing and discovering services, as it can be seen in Figure 9. Thus, the main artefact is the so called UDDI registry, which represents a directory for published web services containing data and meta-data about them which is used to fulfil client queries.



SOURCE: Bryce (2005).

Figure 9: Using a UDDI Registry.

The best-known registry still is the UDDI Business Registry (UBR), a public registry that serves as some kind of master directory similar to DNS root servers, although the biggest field of application are private registries either within a single company or shared between partner businesses.

Besides UDDI's main purpose of describing and discovering single web services, it is worth mentioning that it is also possible to publish abstract WSBPEL process definitions in the same manner.[19]

3.2.1.6 Web Services Description Language (WSDL)

The next layer in the web services stack is used to describe network services in a XML document for the client. The current WSDL Note Version 1.1 was provided by the W3C and co-authored by Ariba, IBM and Microsoft.

[18] Compare Sleeper, 2002, Introduction.
[19] Compare Riegen & Trickovic, 2004.

The basic elements of a WSDL description can be seen in Table 1. First of all, types are defined which will later be used to put together the messages exchanged between the client and the server.

Port types describe a named set of operations and the messages involved, supporting the following four exchange patterns: [20]

- 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.

So far, types, messages and the provided operations in terms of port types are defined abstractly and provide sort of an interface to the parties involved in creating a distributed application that needs to exchange messages.

Element Name	Description
types	A container for data type definitions using some type system (such as XSD)
message	An abstract, typed definition of the data being communicated
operation	An abstract description of an action supported by the service
portType	An abstract set of operations supported by one or more endpoints
binding	A concrete protocol and data format specification for a particular port type
port	A single endpoint defined as a combination of a binding and a network address.
service	A collection of related endpoints.

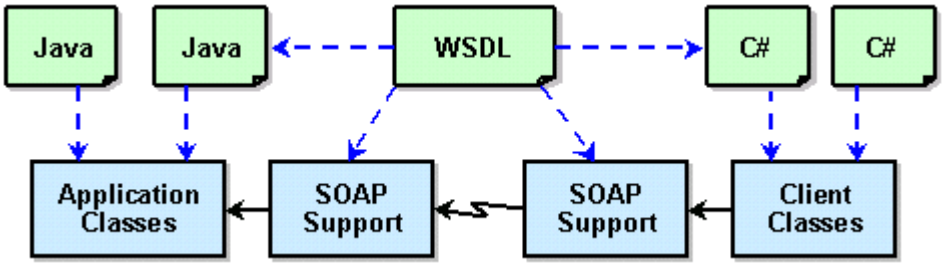
SOURCE: Christensen et al. (2001), section 1.

Table 1: WSDL Elements.

The data format and the protocol used by one port type are defined in a named binding and finally, services group together one or several ports which contain the actual address where the service can be reached.

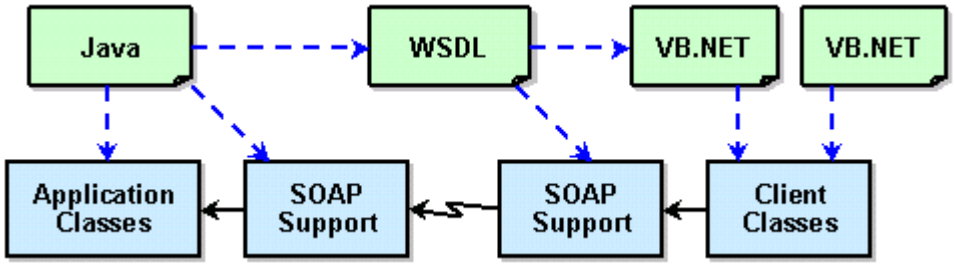
Thus, WSDL breaks down the description of a web service in an abstract part containing the definition for the endpoints and messages and another part describing their concrete network deployment.

Last but not least, this features the reusability of the description as well as urges developers to use a better way to implement their applications as shown in Figure 10.



INSTEAD OF

[20] Compare Christensen et al., 2001, section 2.4.



SOURCE: Provost (2003).

Figure 10: Development paths.

3.2.2 Basic WSBPEL Concepts

3.2.2.1 Business Processes

In the preceding part of the thesis the business process lifecycle was discussed as part of the overall domain of business process management. As stated by GEORGAKOPOULOS & TSALGATIDOU, the lifecycle of a business process covers many tasks, starting with capturing the process in a computerized version, improving and automating the process and finally monitoring the executed instances of that process to name a few.

This section deals with the step of creating an executable representation of a given process with the possibilities offered by WSBPEL.

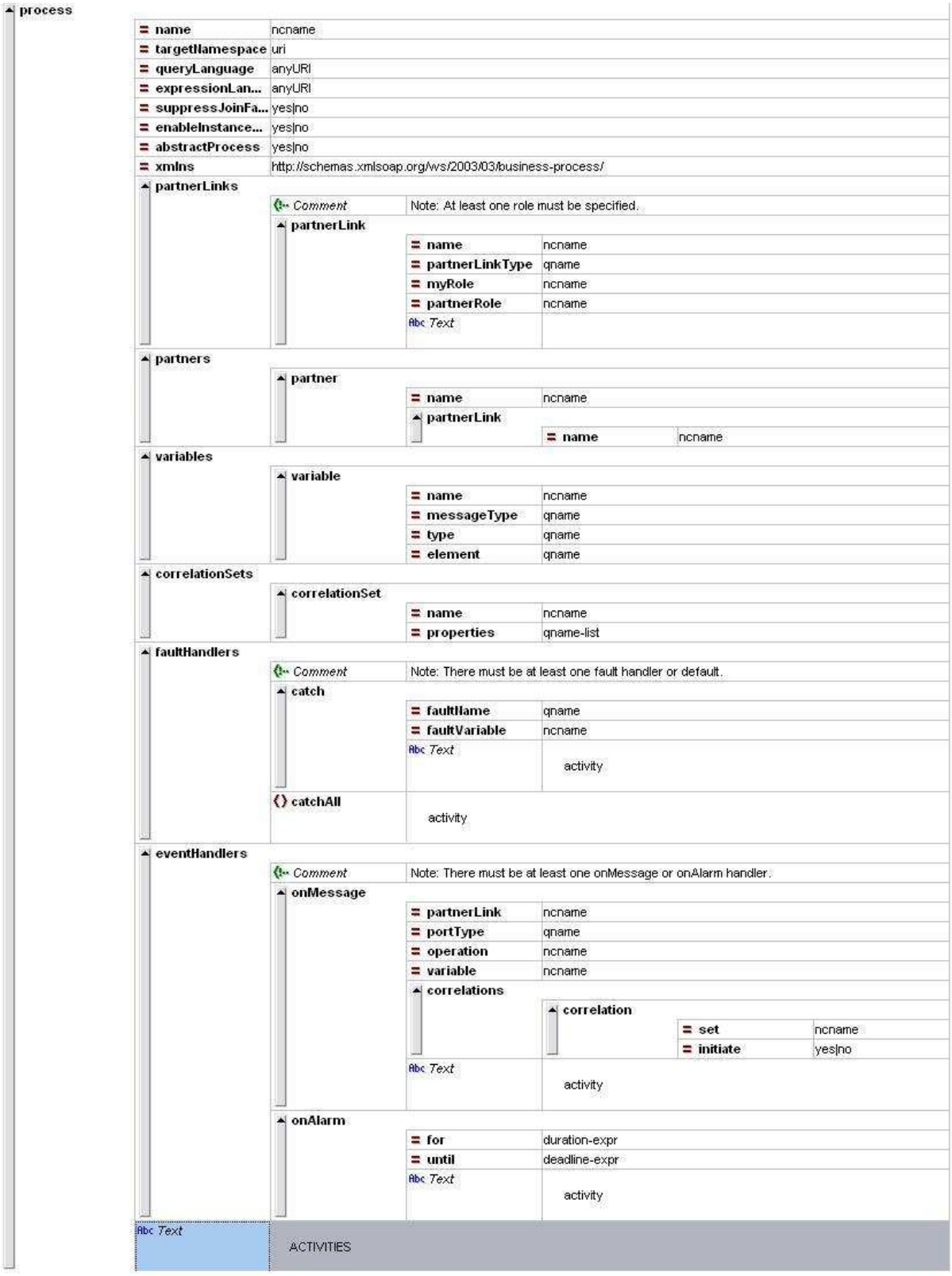
> *"BPELAWS defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web Service interfaces […]"* [21]

These possibilities include the interaction with so called partners by calling their web services or offer web services to them, using variables to receive, transform and send information as well design the control flow of the process based on decisions and sequential or parallel processing.

To create an executable WSBPEL process, the interaction with the partners, the web services used and offered and the types of message that are exchanged have to be described. Then, like in ordinary programming languages, some prerequisites have to be taken care of, e.g. variable declaration. Finally, the process itself has to be described using the supported activities in the appropriate order.

Figure 11 shows the elements that make up a WSBPEL process. Detailed information on partner links is missing in the process description itself, but is described in another document using WSDL.

[21] See Andrews et al., 2003, chapter 1.



Source: Andrews et al. (2003), section 6.2.

Figure 11: Basic structure of a business process in WSBPEL.

The activity in the second last row is called the *primary activity*[22] of the process and contains all the other activities that make up the process. Most of the times a process will be instantiated because

[22] Compare Andrews et al., 2003, chapter 13.

an incoming message is received by a starting *receive* activity[23] in its definition and end, after all activities have successfully been processed.

Depending on the business process the time between instantiation and termination of a process can vary enormously, from seconds to weeks. Therefore, two kinds of processes can be defined to adjust to these deviating durations as shown in Figure 12.

A synchronous process will force the client to wait and therefore block its operations on the client side until the process sends its reply. Thus this kind of process will most likely be used for short running tasks.

An asynchronous process on the other hand closes the connection after it receives the client's request. As soon as it is ready to send its response, it can call back the client and transmit the results. In this case however, the client has to offer a web service itself for the process to be able to call back.

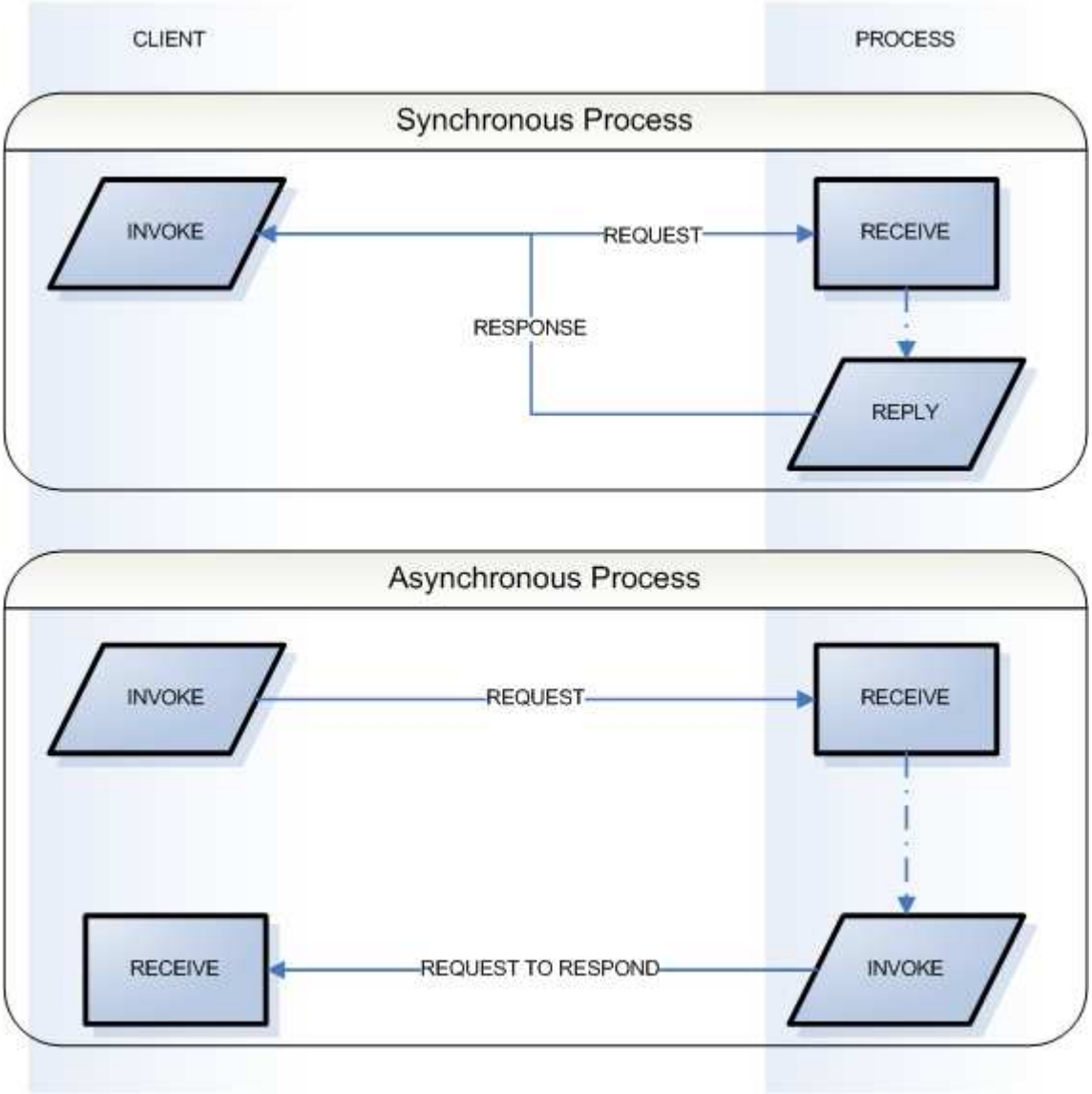


Figure 12: Synchronous vs. Asynchronous Processes.

[23] Refer to chapter 3.2.3.1 and 3.2.3.4 for details on when a process gets instantiated.

While the WSPEL process is executed, it will assign values to variables and call other web services until the last activity is processed. During this enactment, unfavourable states or even failures can occur. In the complex and changing environment in which business processes are executed this is even more the case than with ordinary programming languages where the concept of exceptions is used to handle these situations. WSBPEL deals with this problem using compensation and fault handlers (see section 3.2.3.14 and 3.2.3.15).

3.2.2.2 Partners

Most of the time WSPEL will be used to implement interorganizational processes and therefore the underlying concept of a partner is a very important one[24]. According to ANDREWS ET AL., the term *partner* makes perfect sense.

> *"The relationship of a business process to a partner is typically peer-to-peer, requiring a two-way dependency at the service level. In other words, a partner represents both a consumer of a service provided by the business process and a provider of a service to the business process. This is especially the case when the interactions are based on asynchronous messaging […]"*[25]

Prior to the description of the business process itself, the partners participating, the possible interactions between them as well as the structure of the exchanged information have to be declared. This is done sequentially by first defining the exchanged information and how a *message* containing this information looks like. The identified messages can then be used to model interactions in terms of *port types* where several *operations* demand a message as an input variable and in case of a synchronous invocation also a message as an output variable.

After the message transfer options have been declared, the roles of two services communicating with each other are defined through *partner link types*. Usually there are two roles involved in one partner link type, e.g. Buyer and Seller, although it is also possible to assign only one role to a partner link type if the specific service does not ask for any prerequisites for the calling service.
For a single business process, specific partner links are defined by the declared partner link types, whereby the role of the business process and, if necessary, the role of the partner are explicitly stated. These partner links are used by basic interaction activities described in 3.2.3 (*invoke*, *receive* and *reply*).

3.2.2.2.1 Partner Link Types

The interaction with partners is defined by partner link types which put port types into context by identifying usually two[26] of them and assigning them a role to distinguish if a partner is a provider or a user of a web service.

Port types are used to describe operations provided by a web service and which messages are to be used during these operations. Port types are part of WSDL, the language WSBPEL is mainly built upon.

Normally, the two port types are not defined in the same namespace, because each system is independent from each other and varies in defining the underlying operations and even more the messages to be used. But theoretically the two port types may still derive from the same namespace, e.g. one service provides some kind of simple *callback*[27] functionality.

---

[24] In fact, even if the process has it's boundaries within one organisation one could still see a 'partnership' between the participating systems.
[25] Compare Andrews et al., 2003, chapter 7.
[26] It is also possible to have only one role which means that the specific service does not have any requirements for the caller to meet.
[27] Compare Andrews et al., 2003, chapter 7.1.

```
<plnk:partnerLinkType name="ncname">
    <plnk:role name="ncname">
      <plnk:portType name="qname"/>
    </plnk:role>
    <plnk:role name="ncname">?
      <plnk:portType name="qname"/>
    </plnk:role>
  </plnk:partnerLinkType>
                              Source: Andrews et al. (2003), section 7.1.
```

Source 1: Partner link type definition.

3.2.2.2.2 Partner Links

Partner links add even more semantics to the exchanged messages by identifying the role a service has, i.e. if a partner is offering a service through one of his port types or if he is able to invoke a service offered by the business process itself.

```
  <partnerLinks>
    <partnerLink name="ncname" partnerLinkType="qname"
             myRole="ncname"? partnerRole="ncname"?>+
    </partnerLink>
  </partnerLinks>
                              Source: Andrews et al. (2003), section 7.2.
```

Source 2: Partner link definition.

The roles have to be named according to the roles defined in the corresponding partner link type. Those names are then used for correlating responses to partner links of the same type.

3.2.2.2.3 (Business) Partners

By grouping one or more partner links together, it should be possible to explicitly define which partner links a specific partner has to support.

> *"From the process perspective a partner definition introduces a constraint on the functionality that a business partner is required to provide. […] Partner definitions MUST NOT overlap, that is, a partner link MUST NOT appear in more than one partner definition."*

Because partner links may only occur in one partner definition, a partner who does not support all partner links associated with a specific partner cannot interact at all with the process at all using this partner link.

```
<partners>
  <partner name="ncname">+
    <partnerLink name="ncname"/>+
  </partner>
</partners>
                              Source: Andrews et al. (2003), section 7.3.
```

Source 3: Partner definition.

3.2.2.2.4 Endpoints References

> *"The fundamental use of endpoint references is to serve as the mechanism for dynamic communication of port-specific data for services. An endpoint reference makes it possible in BPEL4WS to dynamically select a provider for a particular type of service and to invoke their operations."* [28]

There are many business scenarios where the complete set of business partners needed to complete a specific process may not be known during the design time, e.g. during the order management step shown in Figure 13, where the supplier with the lowest fare for a certain product is unknown during design time and therefore has to be discovered and linked to dynamically.

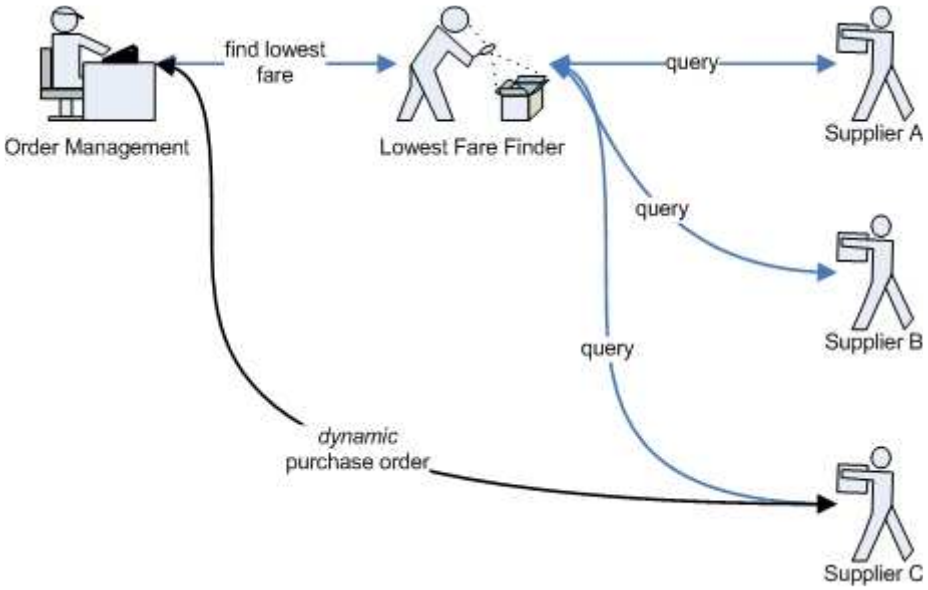


Figure 13: Dynamic discovery of a partner link during runtime.

A good illustration is given by ANDREWS ET AL. in one of their examples appended to the current WSBPEL specification.

```
<!-- Process seller response by
     setting the seller to the endpoint reference provided by
     the seller and invoking the response -->
[…]
<assign>
        <copy>
              <from variable="sellerData"
                    part="endpointReference"/>
              <to partnerLink="seller"/>
        </copy>
 </assign>
                              Source: Andrews et al. (2003), section 16.3.2.
```

Source 4: Dynamic use of a partner link through endpoint references.

3.2.2.3 Variables

Just like in ordinary programming languages, WSBPEL requires variables to be declared and assigned a *type* before they are used. They are mostly used as input or output container for web service requests.
The type of the variable is set using exactly on of the three possibilities *messageType* (WSDL message), *type* (XML Schema simple type) or *element* (XML Schema element).

[28] Compare Andrews et al., 2003, chapter 7.4.

```
<variables>
    <variable name="ncname" messageType="qname"?
                type="qname"? element="qname"?/>+
</variables>
                                Source: Andrews et al. (2003), section 9.2.
```

Source 5: Variable declaration.

Variables can be defined globally or locally, depending on the scope they are declared in and are only accessible in exactly that scope or other nested scopes.

There may only be one variable with the same name in the same scope although it is possible to have a local variable with the same name as an already existing variable in a higher scope, however only under the condition that the two variables still must have the same *messageType*, *type* or *element*.[29]

3.2.2.4 Correlations

The main issue a business process focuses on are interactions: interactions with internal systems, interactions with a single business partner or even interactions with multiple partners at the same time within the same process.

During these interactions messages are exchanged several times but the specific process itself is instantiated only once. Therefore, it is necessary to look for correlated messages and deliver them to the appropriate process instance.

As WSBPEL enables a loosely coupled system per se this necessity implicates some difficulties that other, more tightly coupled, systems do not have. As stated by ANDREWS ET AL., the burden to solve those problems cannot just be left to all the developers of process definitions but should be addressed by the infrastructure hosting the process definitions:

> *"In the object-oriented world, such stateful interactions are mediated by object references, which intrinsically provide the ability to reach a specific object (instance) with the right state and history for the interaction. This works reasonably well in tightly coupled implementations where a dependency on the structure of the implementation is normal. In the loosely coupled world of Web Services, the use of such references would create a fragile web of implementation dependencies that would not survive the independent evolution of business process implementation details at each business partner."* [30]

Correlations are used to associate messages belonging to the same process instance. A good example would be a buyer-seller relationship where a buyer sends a purchase order to a seller. Whenever the seller needs to send a message about this request back to the buyer (or the buyer needs to send a cancellation or additional information to the seller) there has to be a connection to the previously sent purchase order.

Of course, there may be more than just one field to associate messages to the right process instance. If companies do business with each other, they often need to match incoming communication to their internal processes. In our example the buyer needs to find out to which of his executed purchase orders an arriving order confirmation or rejection belongs to. On the other hand the seller may afterwards want to include his own references on the issued invoice to be able to trace its payment.

---

[29] Compare Andrews et al., 2003, section 9.2.
[30] Compare Andrews et al., 2003, chapter 10.

Another aspect as described by KHALAF & NAGY that is also covered by correlation is the decision to be made upon incoming messages, whether a new process has to be instantiated or it is intended for an already existing process waiting for a response.

> *"Message correlation is the BPELAWS mechanism which allows processes to participate in stateful conversations. […] When a message arrives for a Web service which has been implemented using BPEL, that message must be delivered somewhere -- either to a new or an existing instance of the process. The task of determining to which conversation a message belongs, in BPEL's case the task of locating/instantiating the instance, is what message correlation is all about."*[31]

WSBPEL copes with this requirement through the use of correlation sets, for further details please refer to section 3.2.4.2.

3.2.2.5 Standard elements and attributes

All WSBPEL activities are equipped with standard elements and attributes that may be used to give the activity a name or create *links* between activities to put them in a specific order. Please refer to section 3.2.3.12 for details on *links*.

```
name="ncname"?
joinCondition="bool-expr"?
suppressJoinFailure="yes|no"?>
                              Source: Andrews et al. (2003), section 11.1.
```

Source 6: standard attributes

```
<source linkName="ncname" transitionCondition="bool-expr"?/>*
<target linkName="ncname"/>*
                              Source: Andrews et al. (2003), section 11.2.
```

Source 7: standard elements

3.2.3 Basic WSBPEL Building Blocks

3.2.3.1 Receive

This activity provides, among other things, the main starting point for any business process which was previously defined and deployed within a WSBPEL engine. From an external point of view, its interface isn't different at all from any other web service[32].

To present the entry point of a business process, the value of the *createInstance* attribute of the receive activity has to be set to *yes*, only then a new instance of this process is going to be created. In this case, there may be no other activities inside the process definition besides similar receive activities prior to this initial activity. These other receive activities have to use the same correlation sets and on their part create a new instance of this process as well in case they are invoked first. Therefore, it's possible to create several different entry points for the same business process if there is the business need to do so.

The identification of the *partnerlink*, the *portType* and the *operation* tells the WSBPEL engine which web service is called. In a receive activity this will usually be one offered by the business process itself.

Through the use of a variable, the invoking application can hand over data to our business process which can then be used during its execution.

---

[31] Compare Khalaf and Nagy, 2003a.
[32] Compare Juric, 2005.

```
<receive partnerLink="ncname" portType="qname" operation="ncname"
          variable="ncname"? createInstance="yes|no"?
          standard-attributes>
    standard-elements
    <correlations>?
       <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
</receive>
                              Source: Andrews et al. (2003), section 11.4.
```

Source 8: Basic activities: receive

As mentioned in section 3.2.2.1, WSBPEL processes can communicate in an asynchronous or in a synchronous way with their clients, but this has no impact on the definition of the initial receive activity that creates a process instance described in this chapter.

3.2.3.2 Reply

Contrary to the receive activity, synchronous and asynchronous WSBPEL processes differentiate in the use of the reply activity, i.e. it may only be used in synchronous handled processes.

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
        variable="ncname"? faultName="qname"?
        standard-attributes>
    standard-elements
    <correlations>?
       <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
</reply>
                              Source: Andrews et al. (2003), section 11.4.
```

Source 9: Basic activities: reply

In an asynchronous process, the final reply to the client is done by a so called *callback* function where an *invoke* is used to call the client and submit the answer, as illustrated in Figure 12.

If an error occurs, this exception may be communicated through a *faultName*. Whenever a *faultName* is present, the *variable* attribute will carry information concerning this fault and be of the *messageType* of this fault.

3.2.3.3 Invoke

An *invoke* statement is used to call web services offered by other partners. Similar to a *receive* activity the *partnerlink*, *portType* and operation indicate the specific web service to be contacted.

Again, there is a difference in using invoke in a synchronous or an asynchronous way. When used to call a synchronously working web service, the *invoke* activity will block the flow of its *scope* of the process until the reply is received. In this case both, input and output, variables have to be present. If the called web service is working asynchronously the *invoke* activity only needs an input variable and a suitable *receive* activity should be defined later if an answer is expected.

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
          inputVariable="ncname"? outputVariable="ncname"?
          standard-attributes>
    standard-elements
    <correlations>?
       <correlation set="ncname" initiate="yes|no"?
                    pattern="in|out|out-in"/>+
    </correlations>
```

```
    <catch faultName="qname" faultVariable="ncname"?>*
      activity
    </catch>
    <catchAll>?
      activity
    </catchAll>
    <compensationHandler>?
      activity
    </compensationHandler>
</invoke>
                                Source: Andrews et al. (2003), section 11.3.
```

Source 10: Basic activities: invoke

The *catch* statements work similar to those in ordinary programming languages. If one is recognized, the appropriate branch is processed. If there is no *catch* defined within an *invoke*, the next higher scope is examined the fault is escalated to it.

See section 3.2.3.14 for details on compensation handlers.

3.2.3.4 Pick

When using a *pick* statement, the WSBPEL engine will wait for any incoming requests matching one of the defined *onMessage* patterns and execute the proper part. The *onMessage* part is very similar to a *receive* activity described in section 3.2.3.1, enhanced with a branch dependent on an alarm timer, which will be processed if no suitable message is received before the alarm goes off.

```
<pick createInstance="yes|no"? standard-attributes>
    standard-elements
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname" variable="ncname"?>+
      <correlations>?
         <correlation set="ncname" initiate="yes|no"?>+
      </correlations>
      activity
    </onMessage>
    <onAlarm (for="duration-expr" | until="deadline-expr")>*
      activity
    </onAlarm>
</pick>
                                Source: Andrews et al. (2003), section 12.4.
```

Source 11: Basic activities: pick

Another special case is the use of the *createInstance* attribute. If a new instance of this process should be created, each of the possible incoming messages has to fulfil the same requirements as a similar *receive* activity, already described in section 3.2.3.1.

3.2.3.5 Assign

The assign element can be used to copy a variable, parts of a variable, an expression, a message property or even a partner link to a type-compatible variable or partner link. It is often used to save received input elements or to calculate result values just like in any ordinary programming language.

```
<assign standard-attributes>
    standard-elements
    <copy>+
      from-spec
      to-spec
    </copy>
```

```
</assign>
                                    Source: Andrews et al. (2003), section 9.3.
```

Source 12: Basic activities: assign

Source 13 below shows a simple illustration of an assign statement. Please refer to the WSBPEL Specification for details.

```
<assign >
    <copy>
        <from variable=”customer” part=”id”/>
        <to variable=”customerid”/>
    </copy>
</assign>
                                    Source: Andrews et al. (2003), section 9.3.
```

Source 13: Basic activities: example of an assign statement

3.2.3.6 Throw

The *throw* statement is straightforward and does not require predefining fault names that may occur. The only requirement to use it is that the specified name has to be globally unique.

```
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
    standard-elements
</throw>
                                    Source: Andrews et al. (2003), section 11.6.
```

Source 14: Basic activities: throw

3.2.3.7 Wait

It is possible to suspend the execution of the current process for a specific period of time or until a certain point in time has arrived.

```
<wait (for="duration-expr" | until="deadline-expr") standard-
attributes>
    standard-elements
</wait>
                                    Source: Andrews et al. (2003), section 11.7.
```

Source 15: Basic activities: wait

An example for an *until* expression would be: *until=”’2006-02-16T09:00+01:00”’*.

3.2.3.8 Empty

An *empty* statement is similar to a *wait*, with the difference that it does not wait depend on a specific time to finish. According to ANDREWS ET AL. it may be useful to wait for an expected fault.

```
<empty standard-attributes>
    standard-elements
</empty>
                                    Source: Andrews et al. (2003), section 11.8.
```

Source 16: Basic activities: empty

3.2.3.9 Sequence

The *sequence* statement is used to group activities which should be executed in chronological order as they appear in this list. Together with the *flows*, *sequences* provide a way to have different ‘threads’ executed synchronously in a given order.

```
<sequence standard-attributes>
    standard-elements
    activity+
</sequence>
```

Source: Andrews et al. (2003), section 12.1.

Source 17: Basic activities: sequence

3.2.3.10 Switch

As in ordinary programming languages, a *switch* statement can be used to evaluate a boolean expression with an alternative branch in case no other condition was met.

```
<switch standard-attributes>
    standard-elements
    <case condition="bool-expr">+
      activity
    </case>
    <otherwise>?
      activity
    </otherwise>
</switch>
```

Source: Andrews et al. (2003), section 12.2.

Source 18: Basic activities: switch

3.2.3.11 While

Again, the *while* statement is analogue to while statements in ordinary programming languages, therefore a detailed explanation is omitted at this point.

```
<while condition="bool-expr" standard-attributes>
     standard-elements
     activity
</while>
```

Source: Andrews et al. (2003), section 12.3.

Source 19: Basic activities: while

3.2.3.12 Flow

The *flow* element allows WSBPEL to enable concurrent processing of several branches. It can also be combined more than once with the *sequence* activity, allowing process designers to create complex timelines for their activities.

> *"[…] a flow activity creates a set of concurrent activities directly nested within it. It further enables expression of synchronization dependencies between activities that are nested directly or indirectly within it."*[33]

Another very important feature that comes with the flow activity is the concept of *links*. Links are necessary to express dependencies between different activities of a *flow* for example between two sequences or even between activities nested in them.

```
<flow standard-attributes>
    standard-elements
    <links>?
      <link name="ncname">+
    </links>
    activity+
</flow>
```

[33] Compare Andrews et al., 2003, chapter 12.5.

Source: Andrews et al. (2003), section 12.5.

Source 20: Basic activities: flow

First of all, *links* have to be given a name. Later, these names can be used to define a *source* and a *target* of that link. The source of a link identifies the preceding part whereas the target of a link has to wait until all or at least one of the preceding elements is finished, depending on its attribute *joinCondition*.

```
<flow>
    <links>
      <link name="XtoY"/>
      <link name="CtoD"/>
    </links>
    <sequence name="X">
       <source linkName="XtoY"/>
       <invoke name="A" .../>
       <invoke name="B" .../>
    </sequence>
    <sequence name"Y">
       <target linkName="XtoY"/>
       <receive name="C" ...>
         <source linkName="CtoD"/>
       </receive>
       <invoke name="E" .../>
    </sequence>
    <invoke partnerLink="D" ...>
      <target linkName="CtoD"/>
    </invoke>
</flow>
```

Source: Andrews et al. (2003), section 12.5.

Source 21: Basic activities: example fort he use of links in flows

There are several aspects to pay attention to while working with links, among the most obvious ones is for instance to avoid cyclic directed graphs. To read more about the use of links please refer to ANDREWS ET AL. (2003), section 12.5.

3.2.3.13 Scopes

The concept of a scope represents the idea of a segregated unit within a business process model which can be used to give activities a semantic context to which they belong to.

It is possible to have shared variables, correlation sets as well as compensation, fault and event handlers for all nested activities within a single scope and therefore scopes play an important when it comes to exception handling during the execution of a WSPEL process. If that is the case, it is very likely that previously started activities may need to be undone.

```
<scope variableAccessSerializable="yes|no" standard-attributes>
    standard-elements
    <variables>?
      ... see above under <process> for syntax ...
    </variables>
    <correlationSets>?
      ... see above under <process> for syntax ...
    </correlationSets>
    <faultHandlers>?
      ... see above under <process> for syntax ...
    </faultHandlers>
    <compensationHandler>?
      ... see above under <process> for syntax ...
```

```
        </compensationHandler>
        <eventHandlers>?
          ...
        </eventHandlers>
        activity
    </scope>
```

Source: Andrews et al. (2003), section 13.

Source 22: Basic activities: scopes

A scope has exactly one primary activity which can be extended to be of arbitrary depth through nested activities. For information on correlation sets please refer to section 3.2.4.2.

There are three kinds of error handlers available which will be discussed in the following sections. Many business process scenarios can have a runtime of hours, days or even weeks. During their execution many complex interactions may occur, the possibility of exceptions in a highly networked business environment are by no means negligible.
Therefore, error handling and the recovery from exceptions have to be a substantial part of WSBPEL, which distinguishes three different layers to cover deviant behaviours during process enactment, each one to cope with more or less expectable and severe problems or allowing better dynamic interaction with the process itself.

The attribute *variableAccessSerializable* can be used to create scopes which provide concurrency control over the access to shared variables. Those scopes are then called *serializable scopes*.

Their behaviour is described best with an example given in the WSBPEL specification:

> *"Suppose two concurrent serializable scopes, S1 and S2, access a common set of variables (external to them) for read or write operations. The semantics of serializability ensure that the results of their behavior would be no different if all conflicting activities (read/write and write/write activities) on any shared variable were conceptually reordered in such a way that either all activities within S1 are completed before those in S2 or vice versa."* [34]

3.2.3.14 Compensation handlers

Compensation handlers offer an elegant way for undoing previously triggered activities if necessary. For example, if a purchase order is received which causes the two activities billing and shipping to be processed concurrently and either of them fails, e.g. because the ordered good is out of stock, the billing activity should also be revoked automatically and maybe the customer should be informed as well.

```
<compensationHandler>?
    activity
</compensationHandler>
```

Source: Andrews et al. (2003), section 13.3.

Source 23: Basic activities: compensation handlers

There are two ways how compensation handlers can be integrated in a process definition. The required activities to undo other already started activities can either be explicitly declared as shown in Source 23, or they can be defined inside an *invoke* element (see section 3.2.3.3 for details). If used within an *invoke* activity, the compensation handler will be called whenever the activity it is part of needs to be compensated.

[34] Compare Andrews et al., 2003, section 13.6.

ANDREWS ET AL. included a demonstrative comparison of those two possibilities which in their example are completely equivalent.

```
<invoke partnerLink="Seller" portType="SP:Purchasing"
          operation="SyncPurchase"
          inputVariable="sendPO"
          outputVariable="getResponse">
    <correlations>
       <correlation set="PurchaseOrder" initiate="yes"
                    pattern="out"/>
    </correlations>

    <compensationHandler>
        <invoke partnerLink="Seller" portType="SP:Purchasing"
                operation="CancelPurchase"
                inputVariable="getResponse"
                outputVariable="getConfirmation">
          <correlations>
              <correlation set="PurchaseOrder" pattern="out"/>
          </correlations>
        </invoke>
    </compensationHandler>
</invoke>
                          Source: Andrews et al. (2003), section 13.3.1.
```

Source 24: Basic activities: example of an compensation handler inside an *invoke* activity

```
<scope>
    <compensationHandler>
        <invoke partnerLink="Seller" portType="SP:Purchasing"
                operation="CancelPurchase"
                inputVariable="getResponse"
                outputVariable="getConfirmation">
           <correlations>
              <correlation set="PurchaseOrder" pattern="out"/>
           </correlations>
        </invoke>
    </compensationHandler>
    <invoke partnerLink="Seller" portType="SP:Purchasing"
            operation="SyncPurchase"
            inputVariable="sendPO"
            outputVariable="getResponse">
       <correlations>
          <correlation set="PurchaseOrder" initiate="yes"
                       pattern="out"/>
       </correlations>
    </invoke>
</scope>
                          Source: Andrews et al. (2003), section 13.3.1.
```

Source 25: Basic activities: example of an compensation handler inside an *scope* element

Invoking compensation for a scope or an activity is done straightforward with the *compensate* activity where the name of the concerned element has to be included in the *scope* attribute in both times[35]. Please note, that compensation may only be triggered in fault or compensation handlers. One should also be aware of the fact, that only those elements which already completed regularly will successfully be able to compensate. For those that did not finish in a standard way, *fault handlers* have to be used.

```
<compensate scope="ncname"? standard-attributes>
```

[35] Meaning the *invoke* activity can be seen as the 'scope' for this compensation handler.

```
    standard-elements
</compensate>
                                Source: Andrews et al. (2003), section 13.3.2.
```

Source 26: Basic activities: invoking compensation handlers.

One important characteristic about compensation is the fact that, whenever a compensation handler is invoked, it will operate in a snapshot view of the system at this moment. Nothing inside the process can be changed during compensation. Only external entities can be affected.

3.2.3.15 Fault handlers

As stated in the previous section, compensation handlers fail to undo the partial work done in incomplete or faulty scopes. Therefore it is sometimes necessary to catch and deal with occurring errors immediately.

It is possible to deal accordingly to the specific problem through the use of *catch* branches. Each node may define a *faultName* and a *faultVariable*, both are optional. It doesn't make sense to use neither of them because there is also a *catchAll* branch to cover all unhandled faults.

If the fault handler is called, the *catch* with the appropriate *faultName* and *faultVariable*, meaning the WSDL message types are equivalent, will be called. If there is no match, a matching *faultName* will overrule a matching *faultVariable*. If neither of those two options applies the default *catchAll* branch will be processed.

```
<faultHandlers>?
    <!-- there must be at least one fault handler or default -->
    <catch faultName="qname"? faultVariable="ncname"?>*
      activity
    </catch>
    <catchAll>?
      activity
    </catchAll>
</faultHandlers>
                                Source: Andrews et al. (2003), section 13.4.
```

Source 27: Basic activities: fault handlers

As a *catchAll* element is optional as well, it could be possible that a fault is not dealt with at all in the current scope. In this case, the fault will be escalated to the next higher level. If no scope in the hierarchy catches the fault, the process will terminate abnormally.

As with compensation handlers it is possible to include fault handlers straightforward in an *invoke* activity.

```
<invoke partnerLink="Seller"
          portType="SP:Purchasing"
          operation="SyncPurchase"
          inputVariable="sendPO"
          outputVariable="getResponse">
    <catch faultName="SP:POFault" faultVariable="POFault">
      <!-- handle the fault -->
    </catch>
</invoke>
                                Source: Andrews et al. (2003), section 13.4.
```

Source 28: Basic activities: example of an implicitly declared fault handler

3.2.3.16 Event Handlers

This third kind of 'error handler' is in fact not concerned with errors directly but allows a more dynamic behaviour during process enactment.

> *"It is important to emphasize that event handlers are considered a part of the normal behavior of the scope, unlike fault and compensation handlers."*[36]

Event handlers provide this agility by offering two ways, similar to a *pick* activity, to intervene concurrently in the normal processing of a related scope or even the whole process.
The handlers may be defined dynamically using variable data and they will be waiting for the specified event whenever the scope is active or the process is instantiated.

```
<eventHandlers>?
    <!-- there must be at least one onMessage or
         onAlarm handler -->
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname"
               variable="ncname"?>*

      <correlations>?
          <correlation set="ncname" initiate="yes|no">+
      </correlations>
      activity
    </onMessage>
    <onAlarm for="duration-expr"? until="deadline-expr"?>*
      activity
    </onAlarm>
</eventHandlers>
                                Source: Andrews et al. (2003), section 13.5.
```

Source 29: Basic activities: event handlers

After an event handler has been triggered the nested activities will be executed concurrently with the process instance and the handler will again wait for incoming messages. Thus, it's possible to have same event be triggered again, resulting in overlapping variable access times and it is therefore necessary to deal with concurrency and simultaneous access to shared variables. One possible solution would be to use so called *serializable scopes*, as described in section 3.2.3.13, and leave the synchronisation to the WSPEL engine.

3.2.4 Advanced WSBPEL Topics

3.2.4.1 Message Properties

The structure of a message is defined trough it's *messageType*. Furthermore it's possible to highlight a part of the message by naming a *property* for it and assigning a simple type to it.

> *"A property definition creates a globally unique name and associates it with an XML Schema simple type. The intent is not to create a new type. The intent is to create a name that has greater significance than the type itself."* [37]

Hence, properties are used to associate a name with a type, not to declare any variables though, but to emphasize the meaning of information available in such a type.

---

[36] Compare Andrews et al., 2003, chapter 13.5.
[37] Compare Andrews et al., 2003, chapter 8.2.

WEBBER[38] compares this concept to object references in traditional programming languages, where a friendly name with a given type information is bound to another object. In WSBPEL this binding has to be done through *property aliases* which bind the value of the property to the value of a message part.

> *"A property is a named, typed data element which is defined within a WSDL document, and whose value is extracted from an instance of a WSDL message by applying a message-specific XPath expression. In WSDL, a propertyAlias defines each such mapping. The mappings are message specific, hence a single property can have multiple propertyAliases associated with it."*[39]

In a nutshell, properties do not contain information themselves, but are assigned with a meaningful name, an appropriate simple type and through the use of property aliases are connected to a specific part of a message.

```
<wsdl:definitions name="ncname"
   xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
process/">
    <bpws:property name="ncname" type="qname"/>
    ...
    <bpws:propertyAlias propertyName="qname"
           messageType="qname" part="ncname" query="queryString"/>
    ...

</wsdl:definitions>
                                    Source: Andrews et al. (2003), section 8.2.
```

Source 30: Advanced topics: message properties

The concept of properties is mainly used by correlation sets described in the following section, where they are used to create groups of identifiers to map messages to the right process instances.

A simple example of how to define message properties will be given here, extended with the related correlation sets using it in the next section.

```
<?xml version="1.0"?>
<definitions name="correlatedMessages"
  [...] >
  <!--define schema types for PO and invoice information -->
  <types>
     <xsd:schema
targetNamespace="http://samples.otn.com/supplyMessages">
      <xsd:complexType name="PurchaseOrder">
       <xsd:sequence>
        <xsd:element name="CID" type="xsd:string"/>
        <xsd:element name="Order" type="xsd:int"/>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="PurchaseOrderResponse">
       <xsd:sequence>
        <xsd:element name="CID" type="xsd:string"/>
        <xsd:element name="Order" type="xsd:int"/>
        <xsd:element name="VID" type="xsd:string"/>
        <xsd:element name="InvNum" type="xsd:int"/>
              </xsd:sequence>
      </xsd:complexType>
```

[38] Compare Webber, 2003, chapter "Message Properties and Property Aliases".
[39] Compare Khalaf and Nagy, 2003a.

```
      </xsd:schema>
    </types>
    <message name="POMessage">
      <part name="PO" type="tns:PurchaseOrder"/>
    </message>
    <message name="POResponse">
      <part name="POR" type="tns:PurchaseOrderResponse"/>
    </message>
    <bpws:propertyAlias propertyName="cor:customerID"
            messageType="tns:POMessage" part="PO"
            query="/PO/CID"/>
    <bpws:propertyAlias propertyName="cor:orderNumber"
            messageType="tns:POMessage" part="PO"
            query="/PO/Order"/>
    <bpws:propertyAlias propertyName="cor:customerID"
            messageType="tns:POResponse" part="POR"
            query="/POR/CID"/>
    <bpws:propertyAlias propertyName="cor:orderNumber"
            messageType="tns:POResponse" part="POR"
            query="/POR/Order"/>
    <bpws:propertyAlias propertyName="cor:vendorID"
            messageType="tns:POResponse" part="POR"
            query="/POR/VID"/>
    <bpws:propertyAlias propertyName="cor:invoiceNumber"
            messageType="tns:POResponse" part="POR"
            query="/POR/InvNum"/>
    <!-- ... -->
  </definitions>
                    SOURCE: Oracle (2005), Tutorial 109. CorrelationSets.
```

Source 31: Message properties example.

The example shown above in Source 31 shows a WSDL file for a simple buyer-seller-relationship containing two messages and emphasizes their content using six message properties. These can now be used to define correlation sets as shown in the next section.

3.2.4.2 Correlation Sets

A correlation set consists of one or more message properties and is used as some kind of fingerprint to match incoming messages to the right process. It may be used together with the following activities: *receive*, *reply*, *invoke* and *pick*.

First of all it is necessary to group the message properties appropriately into the needed correlation sets and naming them. The names are then used during the process execution to refer to the required correlation sets.

```
<correlationSets>?
    <correlationSet name="ncname" properties="qname-list"/>+
</correlationSets>
                              Source: Andrews et al. (2003), section 10.2.
```

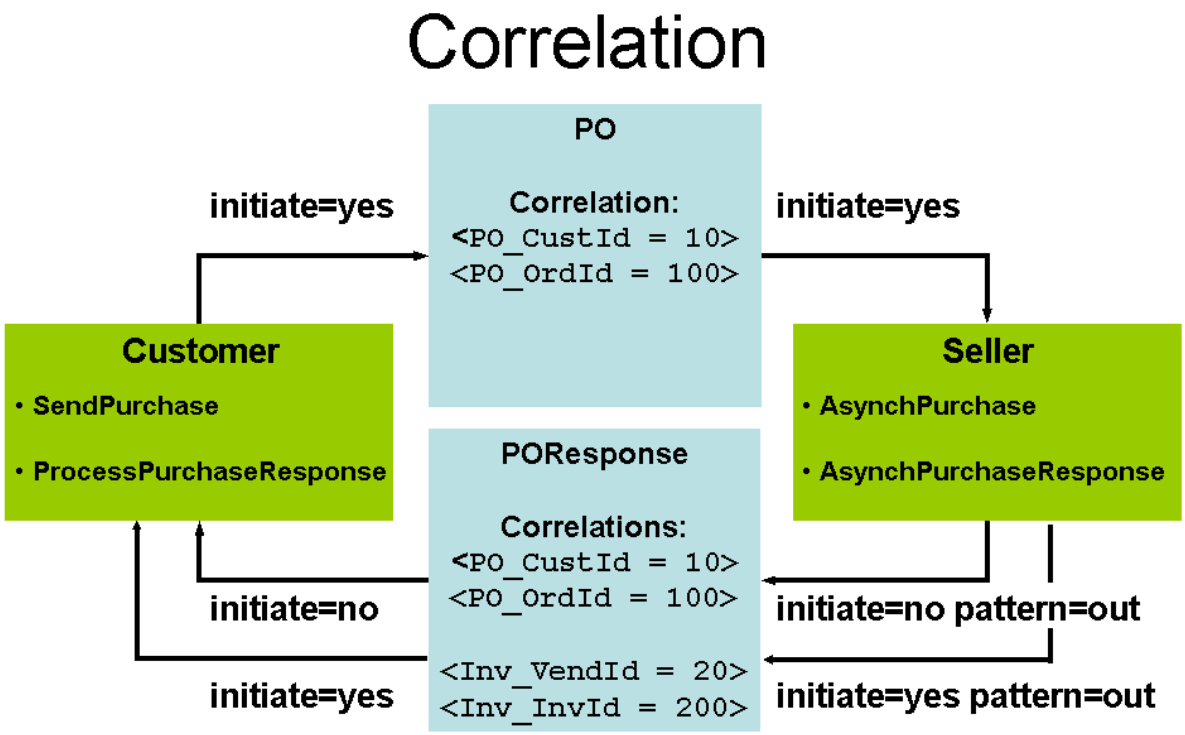
Source 32: Advanced topics: correlation sets

To be of any use, a correlation set has to be initiated once during its lifetime, which means, that during this initialization the values of the related incoming message are assigned to the property or the properties of the given correlation set. To cause the WSBPEL engine to do so, it is necessary to assigning the value *yes* to the attribute *initiate* of our correlation set in the appropriate activity as shown in the continued example from the last section in Source 33: with a *receive* and an *invoke* activity using correlation sets.

```
<process name="Seller"
  [...] >

  <partnerLinks>
    <partnerLink name="Buyer"
         partnerLinkType="seller:Seller"
         myRole="SellerProvider"
         partnerRole="BuyerRequester"/>
  </partnerLinks>

  <variables>
    <variable name="input" messageType="smsg:POMessage"/>
    <variable name="output" messageType="smsg:POResponse"/>
  </variables>

  <correlationSets>
    <correlationSet name="PurchaseOrder"
        properties="cor:customerID cor:orderNumber"/>
    <correlationSet name="Invoice"
        properties="cor:vendorID cor:invoiceNumber"/>
  </correlationSets>

  <sequence>
    <!-- receive PO from buyer-->
    <receive partnerLink="Buyer" portType="seller:Seller"
    operation="AsyncPurchase"
    variable="input"
    createInstance="yes">
      <correlations>
        <correlation set="PurchaseOrder" initiate="yes"/>
      </correlations>
    </receive>

    [...]
</process>
                    SOURCE: Oracle (2005), Tutorial 109. CorrelationSets.
```

Source 33: Correlation sets example.

When using the *invoke* activity another distinction, called *pattern*, is possible, to cause the correlation set to be initiated either before the partner web service is called, or after the reply from the called web service received. The assigned value *out* in our example therefore means that the correlation set with the name *Invoice* should be initialized before the partner web service is contacted. The other possibility would have been to assign the attribute *pattern* the value *in*, which would have caused the WSPEL engine to wait with the initialization until a reply to our *invoke* activity is received.



SOURCE: Kavantzas & Lehmann (2003), Correlation.

Figure 14: Example for correlations.

4 Process Controlling and Auditing

The continuous advancements in the field of application support applicable for many different business purposes implicate very easy access to a huge amount of business data and its metadata for an evaluation of the business.

Although this information looks like a goldmine for business analysts, it also bears its drawbacks if used inaccurately. Just because something can be measured easily does not mean that its analysis is of any relevance. Moreover, one could easily forget about far more significant tasks within the business just because they may be harder to measure.

> *"Controlling is - from a functional perspective - the sub-system of management, which coordinates planning, evaluation and information supply in a goal-oriented, system-building or system-connecting way. By doing so, controlling supports the adoption and coordination of the entire system."* [40]

This definition given by Horváth points out the important function of controlling for businesses in general and explains how controlling is embedded within the business structure. This chapter will try to present some of those points in more detail, while special attention is paid to process controlling in particular.

Since there are many answers to this question why an organisation should actually spend resources on controlling, among common ones like the constantly rising competition and sustained shorter product lifecycles through innovation, section 4.1 will present some of the more concrete examples.

Furthermore, possible data sources as the foundation of controlling are examined. While traditional examples would be financial performance indicators or customer surveys, section 4.2 tries to add the information gathered through business process auditing to the picture.

Controlling and auditing are used interchangeable in this chapter although it could be argued, that controlling accentuates more the top-down view of the subject, concerned with an overall strategy, whereas auditing could be seen as the bottom-up side of it, where information is actually collected.[41]

4.1 Reasons for Controlling

Although it seems to be self-evident that controlling is a necessity and should be integrated soundly within every organization structure, it is still hard for the persons in charge to name generic but meaningful reasons for doing so.

NEELY offers a simple framework which distinguishes between the 4 *CPs of measurement*[42] to describe the main reasons for controlling and argues that each given reason will fit into one of the following categories:

- Check position
- Communicate position

[40] Compare Horváth, 2002, p. 153.
[41] Zur Mühlen distinguishes between process monitoring and process controlling in terms of their chronological appearance: process monitoring happens during process runtime while process controlling deals with the ex-post analysis of a process.
[42] Compare Neely, 1998, p.71.

- Confirm priorities
- Compel progress

4.1.1 Check Position

If the business doesn't know where its place currently is, the use of any new strategy to improve the situation will be more like a lottery game. Furthermore, without controlling there is no way of telling which previously enforced strategies had an effect and how well they achieved their planned results.

> *"Without measures there is no way of checking whether the plans, either strategic or tactical, are appropriate or delivering the desired results."*[43]

Checking position consists of the following three steps which are needed for meaningful decision-making:

- establish position
- comparing position
- monitor progress

First of all, the business has to find out for itself which the key factors for their success are. For instance, if the company was in the logistic industry, customer satisfaction generated by on time delivery services would be such a factor.

Afterwards, one has to be very careful at choosing the appropriate information used to measure if and how well the defined objectives have been achieved. NEELY presents a few concrete examples which show that poorly chosen designs used for controlling cause employees to focus just on the measured figures, forgetting about the real value behind their work.[44]

Once, meaningful figures are available they can be used to track progress over time but also to compare the company to competitors in the same industry.

4.1.2 Communicate Position

Another very important aspect of controlling is the need to share the gained information among the stakeholders of the business.

The position can be communicated internally to individuals and teams to thank them or to stimulate competition among them and give them reason to outperform over their targets.

Depending on the nature of the measured business object, reports will be communicated to shareholders or the media for marketing reasons as well.

And finally it will be mandatory in some industries to provide detailed information to external parties like governmental regulators.

---

[43] Compare Neely, 1998, p.72.
[44] Compare Neely, 1998, p.33.

4.1.3 Confirm Priorities

Checking position will happen on a regular basis to identify gaps between the formulated strategy and the real world. Depending on this discrepancy, it may be necessary to adjust the previously defined priorities or even reconsider them completely.

An important aspect during this step is the *time lag* that occurs between executing the strategy, detecting abnormalities to the planned strategy and reacting to those unwanted situations. It is obvious that keeping these time lags as short as possible is desirable.

4.1.4 Compel Progress

By deciding on the activities and objectives to be measured, a clear signal is sent to the employees working on those items: these are of importance to the overall goals of the business!

Very often, rewards and benefits are used to make sure that they care as much about these goals as their management does, hence, encouraging them and their colleagues to reach their targets.

If incentives like this are being used, the schemes defining how these payments are connected to the measured figures again have to be very carefully designed. If poorly specified, they could easily result in counteracting the real business goals behind them, since individuals will care much more about reaching their targets to whatever cost, neglecting the company's interests.

4.2 Data Sources for Controlling

Selecting the appropriate data sources to measure business performance is a nontrivial task. First of all, the measured values represent achievements on different strategic and operational levels. Thus, not only the sources of the gathered data vary, but also the way they can be represented.

Figure 15 shows the continuous refinement from a company's mission to all the different actions that are necessary to serve this mission. Every single layer can and should be part of the controlling mechanism and different methods will be needed to be applied to each of them.

SOURCE: Broadbent, 1999, p.18.

Figure 15: Measurement of programmes.

While the overall mission, the core goals and the more specific, measurable, achievable, relevant and time-bound objectives[45] may be harder to capture in a numerical way and should be never

[45] Compare Broadbent, 1999, p.18.

left out of sight, the bottom three, i.e. strategies, plans and actions, deduced from the former are the ones that actually account for the success of the company.

According to BROADBENT, strategies describe how objectives are achieved through company assets, personnel and expertise. Plans refine this concept and consist of specific programmes to realize these strategies. Actions represent the actual executed work to realize the whole pyramid.

As shown in Figure 15, plans can be monitored by defining critical success factors whereas key performance indicators may be used to keep an eye on actions.[46]

It is obvious that, as the tasks to realize the company's vision are getting more concrete, so are the possibilities to monitor the progress achieved. Data collected at the lowest level is aggregated to serve as the basis for evaluating the next higher layer.

Other characteristics of the collected data on the different layers are the intervals at which it is gathered and analyzed and related to that the amount of decisions necessary to correct discrepancies to the specific targets.

While higher layer controlling is concentrating on monitoring managerial tasks and deals with important aspects like customer satisfaction, employee satisfaction, intellectual capital, supplier performance and, still the most important for some people, financial performance[47], the continuous advancements in the IT sector allows to tackle the problem more and more in-depth and intervene much earlier.

4.3 Application Support for Auditing

As mentioned in the introducing paragraphs of this chapter, the increase of application support for business process execution provides a new vast data source for auditing on the lowest level.

Thus, once the auditing framework is in its place, only little overhead is necessary to deduct priceless firsthand information for business decisions. Furthermore, the possibility of an automatic interventions during in real-time during the execution of a business process may open up new opportunities for process designers.

Since it does not solely depend on an ex-post analysis, this new potential goes beyond traditional process controlling. [48]

Thus, it is eligible to say, that the developments in the domain of application support for business process execution will have an impact on controlling as far as the action layer in Figure 15 is concerned.

According to BROADBENT, the criteria for performance measurement are changing as follows:[49]

- From accounting-dominant financial measures towards non-financial performance criteria
- From functional to task based

---

[46] Compare Broadbent, 1999 pp.21 for more details on critical success factors and key performance indicators.
[47] Compare Neely, 1998, chapter 1.
[48] If it is still seen as part of controlling. It could be argued that process interventions are part of the operational execution of a business process, not part of its controlling.
[49] Compare Broadbent, 1999, preface.

- From individual company to the whole value chain
- From the operational to the strategic
- From product to customer

This shift undermines the opportunities of business process monitoring of being a suitable tool to collect and analyze the required information.

The evaluation of the audited information is not part of this thesis, however, it should be briefly noted what kind of information can be collected during process execution.[50]

The data scope of audit information can roughly be classified in two groups. The first group includes information on frequencies, time-related ratios and other information at the process and activity levels.[51]

On the other hand, business processes have business objects associated to them which very often determine how the process is executed during runtime.

Depending on the structure type of the data representing these business objects, they can be very helpful to analyze processes ex-post but also during runtime.

SCHATTEN distinguishes three levels of structured information:[52]

- Highly structured information: relational model or XML format, high granularity
- Semi-structured information: often documents with metadata, high to medium granularity
- Unstructured information: no machine-readable structure or if so, low granularity

Thus, this second group of auditing information may be more or less useful for process analysis and real-time intervention respectively.

[50] Compare Zur Mühlen, 2004, section 4.3.1 for more detailed information.
[51] Compare Zur Mühlen, 2004, section 4.3.1, Information Content & Data Scope.
[52] Compare Schatten, 2003, section 4.3.

5 Empirical Part

So far, the ideas and elements of Business Process Management as well as the importance of auditing in a process oriented organization have been outlined. An introduction to WSBPEL as one representative of process automation technologies was given in chapter 3.

The empirical part of this thesis deals with the great potential automation brings to the world of BPM. Only minimal efforts need to be made to close the gap between executing a business process and monitoring it. SAYAL ET AL. also state that this combination allows business users to be notified or even react in real-time in case predefined conditions are met:

> *"BPMSs have the ability of logging information about the business processes they support, including for instance the start and completion time of each activity, its input and output data […] .This information is a gold mine for business and IT analysts: in fact, its analysis may reveal problems and inefficiencies in process executions and identify solutions in order to improve process execution quality […]. In addition, information on active processes can be used for notifying users and processes of quality degradations."* [53]

There exist plenty of WSBPEL engines, commercial and open source ones[54], which offer their own auditing features. Section 5.1 should give an overview of the capabilities of one commercial WSBPEL engine in this area.

Unfortunately most of these proprietary auditing features are not designed for interoperability with other WSBPEL engines, reducing their value as a monitoring instrument not only for interorganizational processes where heterogeneous systems are most likely to be found.

Therefore, a new approach to monitor WSBPEL processes is presented in section 5.2. The basic idea behind it is to extend a WSBPEL process definition in order to report interesting activities to an auditing web service. The resulting auditable process definition does not use proprietary elements but remains compliant with the WSBPEL standard. It is therefore possible to enhance process definitions running on different WSBPEL engines with the proposed auditing feature and centralise the collected information.

The presented prototype does not deal with the design nor with the analysis of the audited information, but aims at collecting as much information as possible about the process runtime, leaving the processing and the evaluation of the gathered information to another service.

Figure 16 illustrates the basic idea behind the proposed solution. The first part of the solution is the so called auditing web service, described in section 5.2.3, which provides the WSBPEL compliant way to capture the auditing information.
The second part of the prototype deals with the extension of the original WSBPEL process. As explained in chapter 3, the process definition is kept in an XML format and therefore, XSLT can be used to create an auditable version of the original process.

Furthermore, Figure 16 shows where these auditing activities are integrated, i.e. pre and post to each audited activity. This setup not only allows, besides other things, the calculation of cycle times, but can also help to detect exceptional situations.

---

[53] Compare Sayal et al., 2002, chapter 1.
[54] Compare Wikipedia contributors, 2005, chapter 3.

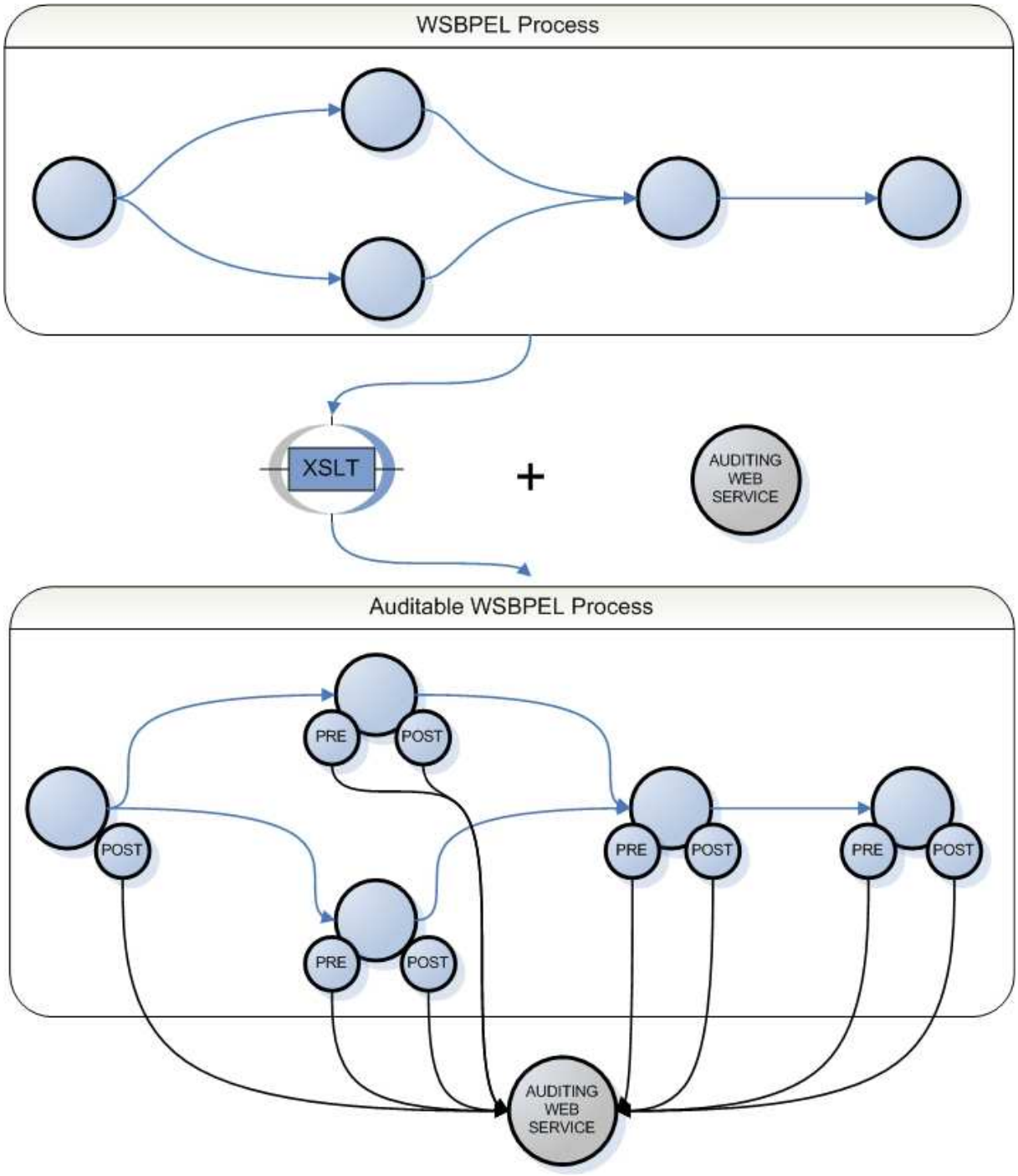


Figure 16: Creating an auditable version of a WSBPEL processes.

In the following sections of, an overview of the auditing features provided by Oracle BPEL Process Manager will be given, followed by a detailed presentation of the proposed prototype to overcome the shortcomings of proprietary solutions.

The empirical part has been developed and tested using Oracle's BPEL Process Manager. The auditing web service was implemented in C# and published on a Microsoft Internet Information Server.

5.1 Auditing in Oracle BPEL Process Manager

The Oracle BPEL Process Manager basically consists of three parts

- Oracle BPEL Process Manager Designer
- Oracle BPEL Process Manager Server
- Oracle BPEL Process Manager Console

The Designer was developed as a plug-in for the Eclipse Platform and represents a graphical user interface to create process definitions. It offers two screens shown in the following figures. The first one, called *Overview* and presented in Figure 17, is used to add partner links and variables. The example shows an auditable WSBPEL process which can be deduced from the already existing partner link to the auditing web service.

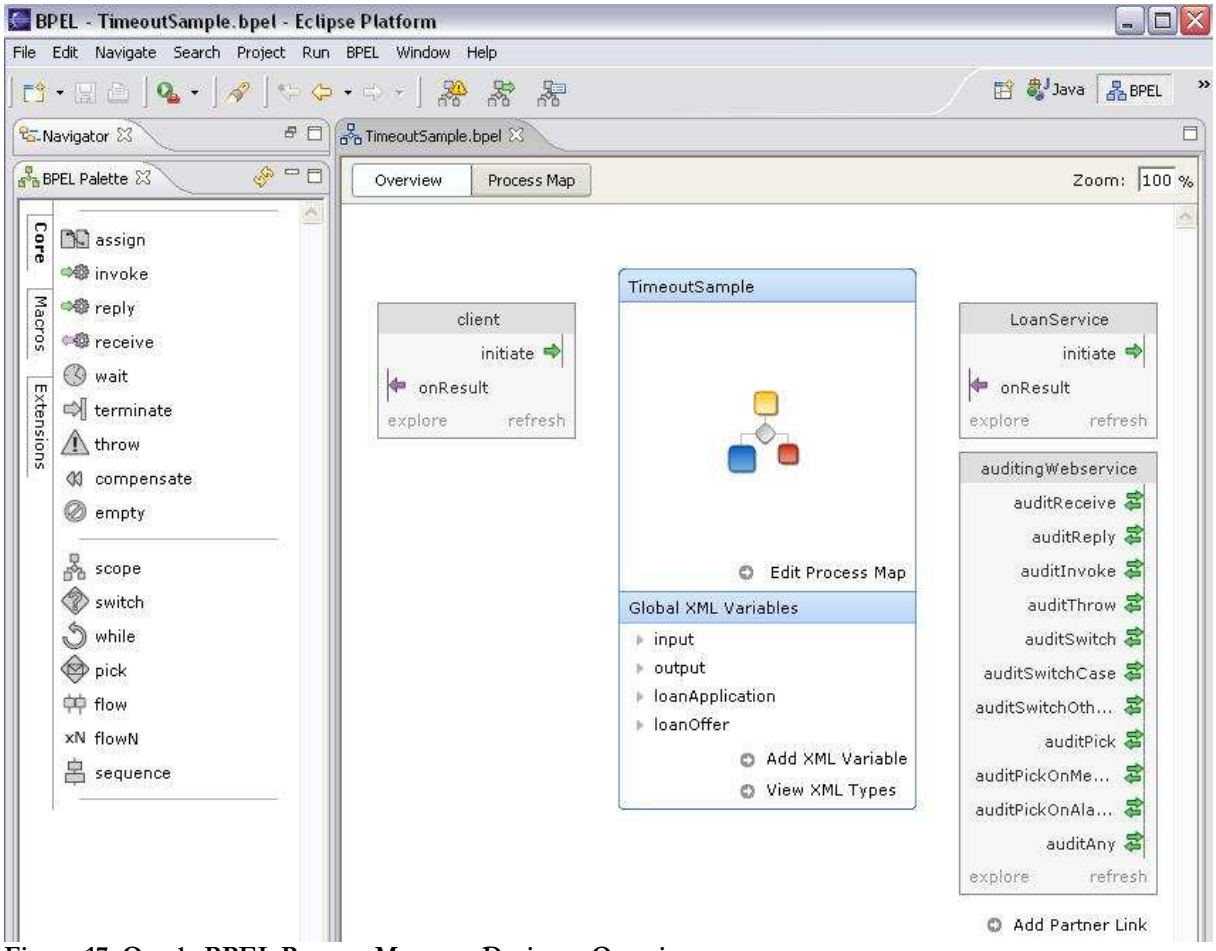


Figure 17: Oracle BPEL Process Manager Designer Overview.

The second screen named *Process Map* (see Figure 18) is used to design the actual process. On the left, the BPEL Palette offers the basic activities which can be dragged and dropped into the process map. The details on each selected activity can be seen on the right pane called *BPEL Inspector*, where standard attributes can be chosen.
The designer does not provide a friendly user interface for every possible WSBPEL element. For instance, to define message properties or correlation sets, there is still the possibility to switch down to the WSBPEL source and manually add those elements.

After a process has been fully designed, it can be validated and finally built by the BPEL Designer.

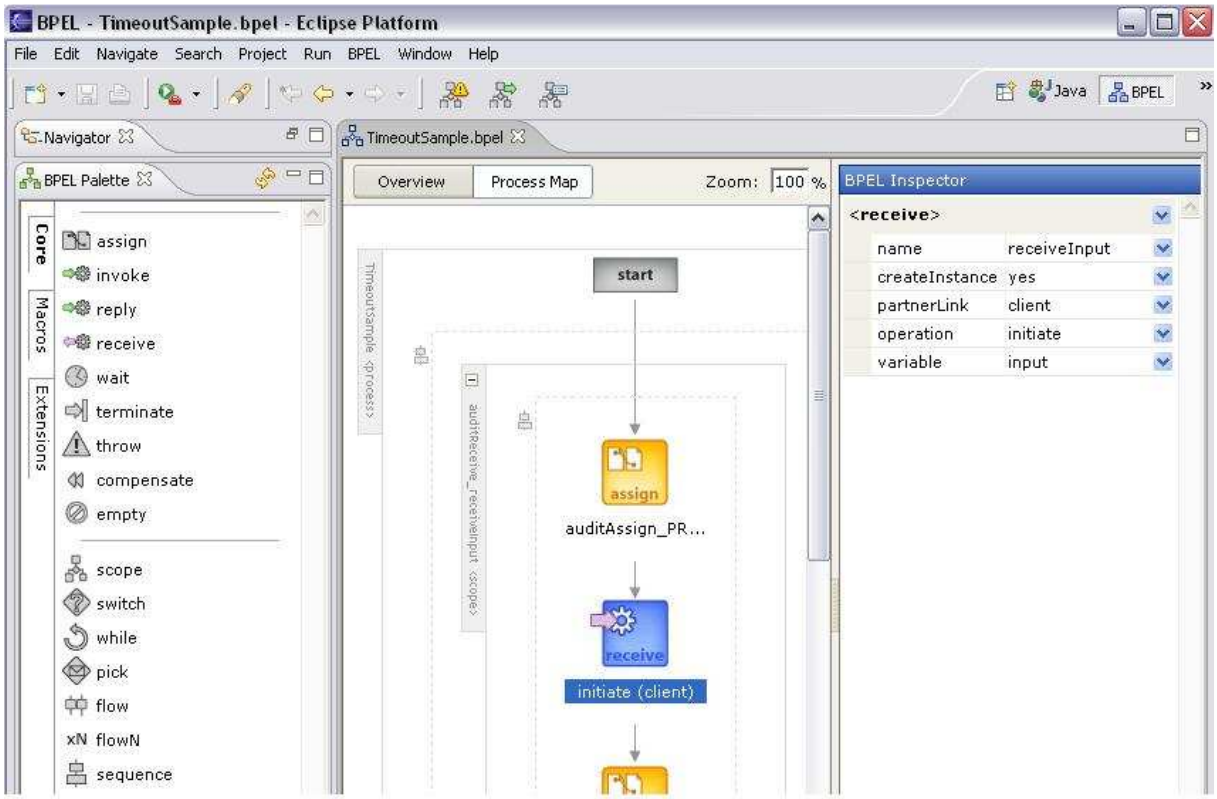


Figure 18: Oracle BPEL Process Manager Designer Process Map.

After a process has been designed and built with the Oracle BPEL Designer, it can be published to the Oracle BPEL Process Manager Server through the BPEL Console. The BPEL Console makes it easy to deploy the created processes so they can be invoked. It is also possible to instantiate a deployed process with arbitrary input messages for testing.

The Oracle BPEL PM Server logs information on each processed activity is available in the form of so called audit trails which contain timestamps and information on concerned variables by default. Those audit trails kept for instantiated processes can be viewed with the BPEL Console.

The BPEL Console only supports to examine the data of one process instance at a time which helps to debug process definitions or investigate specific process instances. However, it does not offer aggregated data of multiple instances which could be used as decision support by business users.

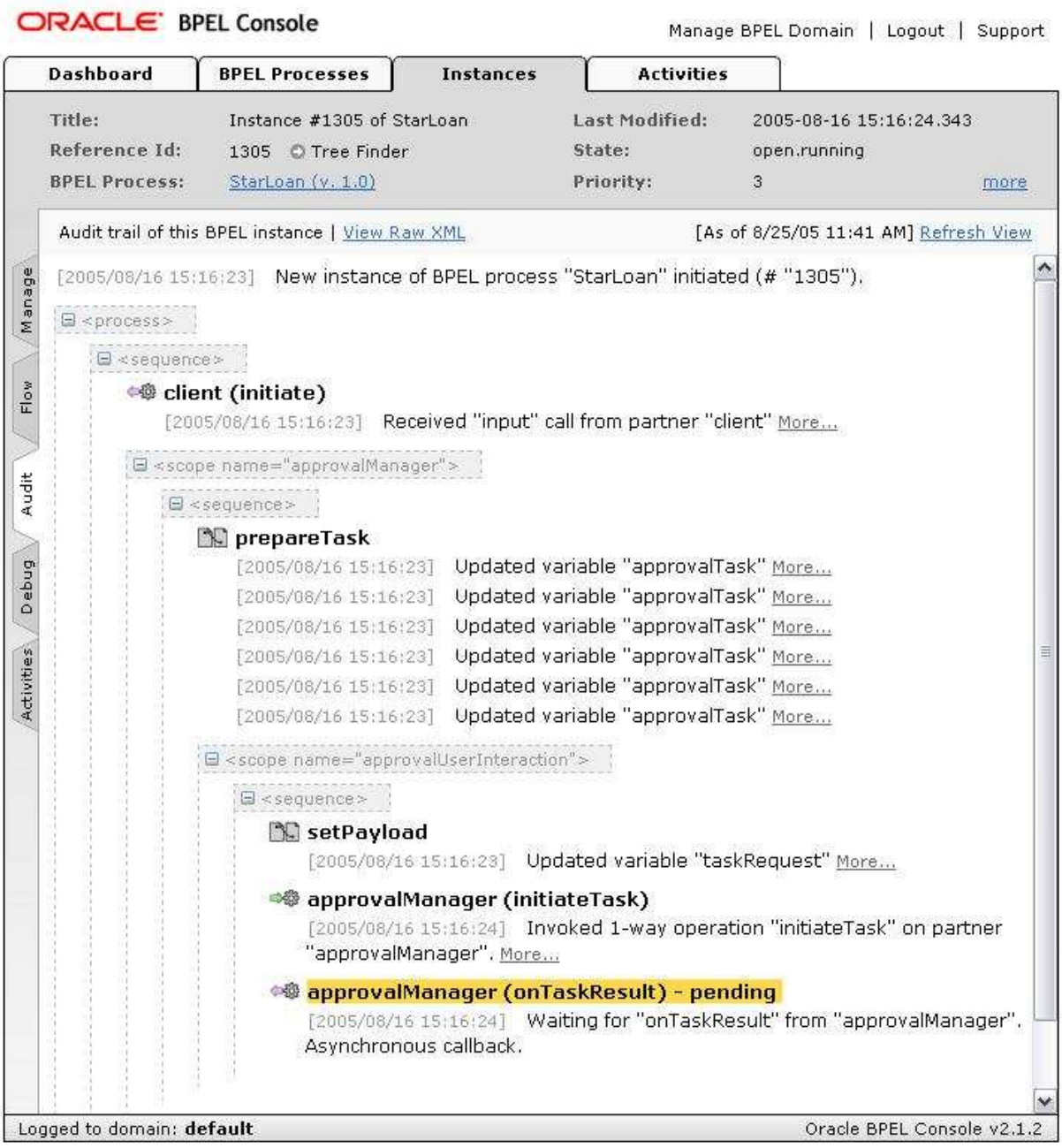


Figure 19: Audit trails provided by Oracle BPEL Process Manager.

The Oracle BPEL Process Manager offers two different ways to add more power to its auditing feature.

The first approach uses the possibility to combine WBSPEL with Java, also known as BPELJ. One of the built-in methods of the BPEL exec extension *bpelx:exec* makes it possible to attach more data to the audit trails kept by the Oracle BPEL PM Server.

```
void addAuditTrailEntry(String message, Object detail)
                Source: Bradshaw, Kennedy and West, 2005, section 10-4
```

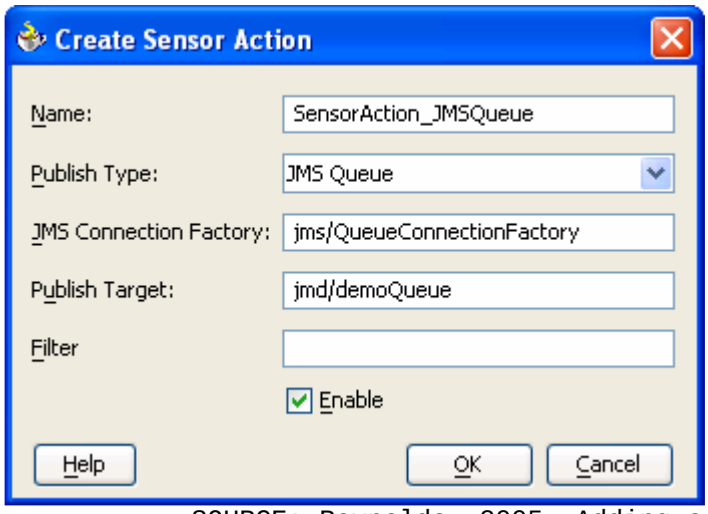
Source 34: Add an entry to the audit trail.

This approach shares the same shortcomings as ordinary audit trails, i.e. it's limited to one process instance only.

Another option which is still under development at the time of writing is to use so called *sensors*.[55] In a short article Anthony Reynolds describes the advantages of this concept compared to traditional auditing provided by Oracle's BPEL Process Manager as follows

> *"Here are some of the things that BAM can do, when given a suitable event feed from a BPEL process.*
>
> - *Calculate event durations and raise alerts if average event duration or individual event duration exceeds specific thresholds.*
> - *Provide statistics on the number of processes at a particular stage of the process.*
> - *Provide cumulative information about processes in flight - for example the amount of revenue expected to be received based on sales processes currently in flight.*
> - *Raise an alert if a certain percentage of events exceed a threshold, for example if more than 10% of loan requests are approved with bad credit.*
> - *…"* [56]

It will be possible to add sensors to any activity in Oracle's BPEL Process Manager. After a sensor has been associated with an activity, the publish type and target to be used can be chosen as shown in Figure 20.



SOURCE: Reynolds, 2005, Adding a Sensor to BPEL.

Figure 20: Sensors in Oracles BPEL Process Manager.

Both possibilities offer a way to receive greater details and with sensors it's even possible to provide a consolidated view over the results of many processes, although both approaches are not pure WSBPEL but a proprietary solution which will only work for Oracle's BPEL Process Manager.

5.2 Proposed Solution & Prototype

This section introduces the functionality and the ideas behind the prototype of the proposed solution.

First of all, the activities available for auditing in WSBPEL and the kind of information that can be audited of each of them are discussed.

---

[55] Compare Bradshaw, Kennedy & West, 2005, section 18.
[56] Compare Reynolds, 2005.

Afterwards, the necessary changes to create an auditable version of a WBSPEL process definition are studied in section 5.2.2.

Finally, the auditing web service that is used to receive the audit information as well as a possible XML Schema to capture it for further processing are briefly presented.

5.2.1 Audited WSBPEL Activities

The developed prototype is meant to report as much information as possible, but there exist several WSBPEL activities[57] which provide information of low interest and are therefore omitted. Furthermore, not all elements and attributes of the concerned activities are needed to be reported, e.g. the standard attributes *joinCondition* and *suppressJoinFailure*, as it is assumed that the original process definition is preserved for later analysis if required.

Only the following activities will be observed and monitored by the auditing web service presented in section 5.2.3 during process execution

- receive
- reply
- invoke
- throw
- pick
- switch

All of these activities are going to be reported to the auditing web service through *ivoke* activities which will be placed appropriately in advance and/or after the specific element as shown in Figure 21. To simplify matters they will be abbreviated with the terms *preaudit* and *postaudit*. The *postaudit* part not only provides values that changed during the enactment of the audited activity to the auditing web service. If missing it also enables the auditing web service to recognize that a certain activity caused the process to terminate abnormally as well as give information about the amount of time a process or an activity took to complete.

[57] See chapter 3 for details on available WSBPEL activities.

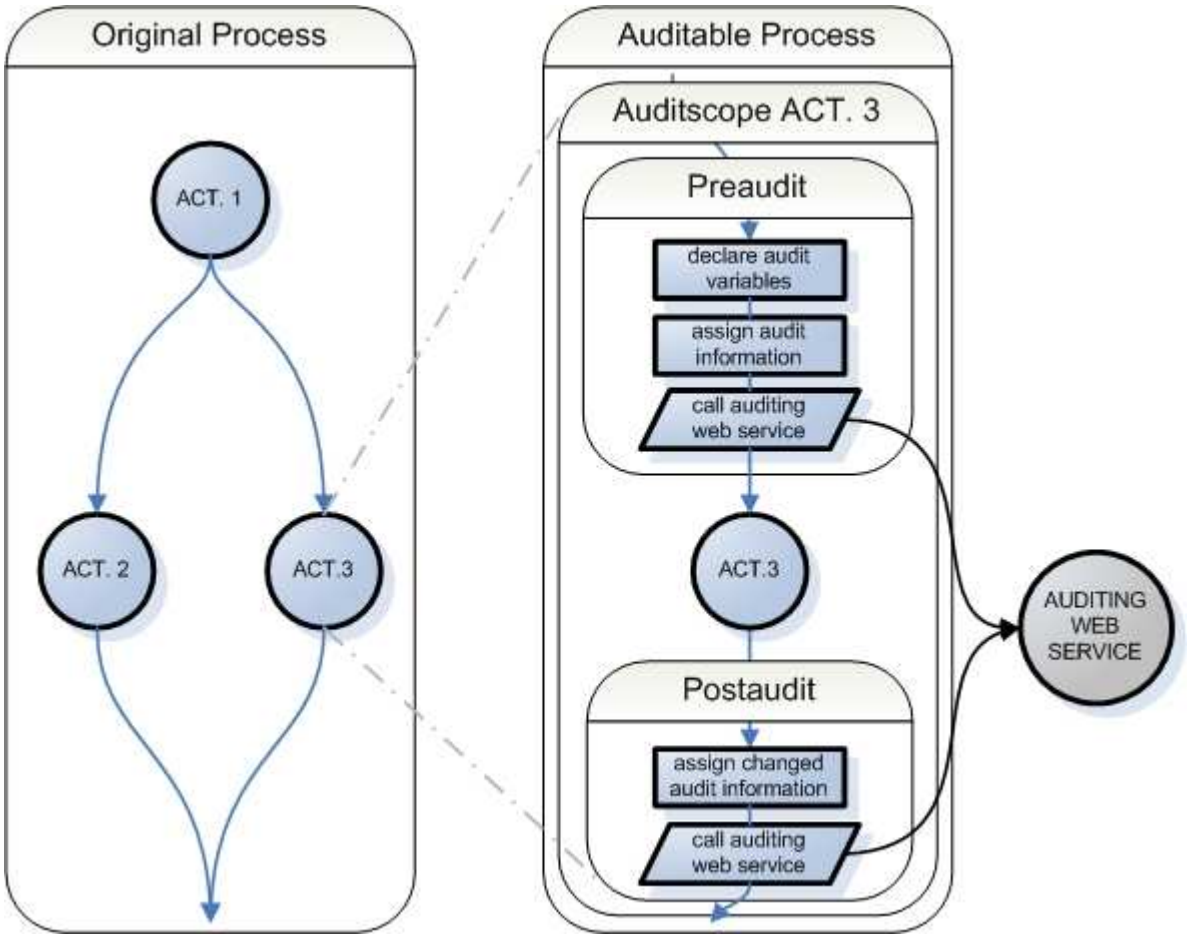


Figure 21: Auditing steps.

Prior to calling the auditing web service, the audit information needs to be assigned to the input variable which will be used to invoke the auditing web service. After *preaudit*, only the possibly changed contents will be updated for *postaudit*, e.g. the return variable of an *invoke* activity.

As described in chapter 3, there may exist more than one entry point for the creation of a new process instance, occurring either as receive or pick activities with the attribute *createInstance* set to *yes*. Since they represent the starting point for the execution of the process, no activity preceding them will be triggered before them. Thus, it doesn't make sense to extend the process definition with a *preaudit* activity for the elements as shown in Table 2. The question mark indicates, that it depends on the activity definition whether an audit activity should be included or not.
A *throw* activity immediately causes the WSBPEL engine to call associated fault handlers and leaves the current branch of the process, leaving the *postaudit* useless.

	receive	reply	invoke	throw	pick	switch
preaudit	?	✓	✓	✓	?	✓
postaudit	✓	✓	✓	×	✓	✓

Table 2: Pre- and postaudit can not be applied to all activities

For every audited activity, a new *scope* is created which will host all the necessary steps for *pre-* and *postaudit*, (also illustrated in Figure 21):

1. Variable creation

2. Preaudit variable assignment
3. Preaudit invoke statement
4. Unchanged audited activity
5. Postaudit variable assignment
6. Postaudit invoke statement

Together with the details on the examined activity, the correlation sets used during process enactment will be transmitted to the auditing web service no matter what type of activity is concerned. This information can be very important to recognize the process instance reported activities belonging to.[58]

Figure 22 shows the beginning of a simple WSBPEL process created with *Oracle BPEL PM Designer* with a starting receive activity, followed by an assignment and an the invocation of another web service. After the six previous steps have been performed to enhance auditing to this process definition, Figure 23 and Figure 24 show the changed process as it is presented in the *BPEL designer* view.

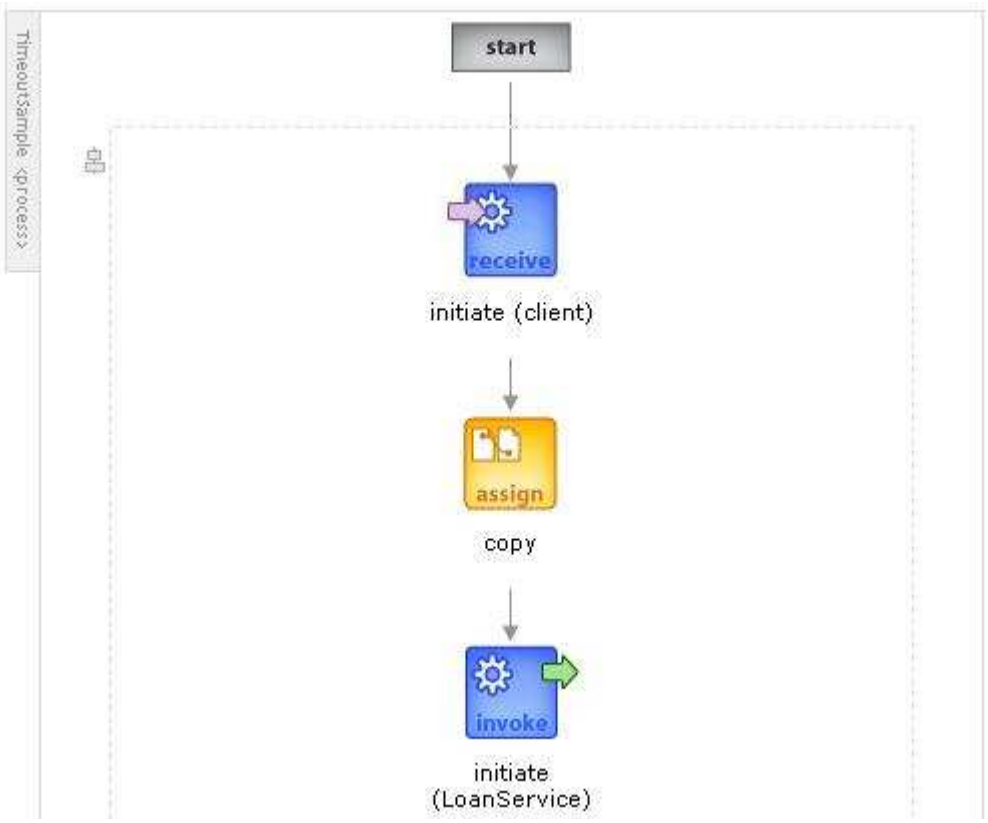


Figure 22: Unprocessed process definition sample.

One interesting detail about the changed *receive* activity with the name *initiate* in Figure 23 is the fact, that the third step, the *preaudit invoke* statement, is missing. The reason of course is easy determinable: it is the starting activity of this process with the *createInstance* attribute set to *yes*, therefore *preaudit* is omitted here. Nevertheless the *preaudit* variable assignment is kept for simplicity, because *postaudit* variable assignment only deals with possibly changed variable contents.

[58] See section 6 for further ideas to address this issue.

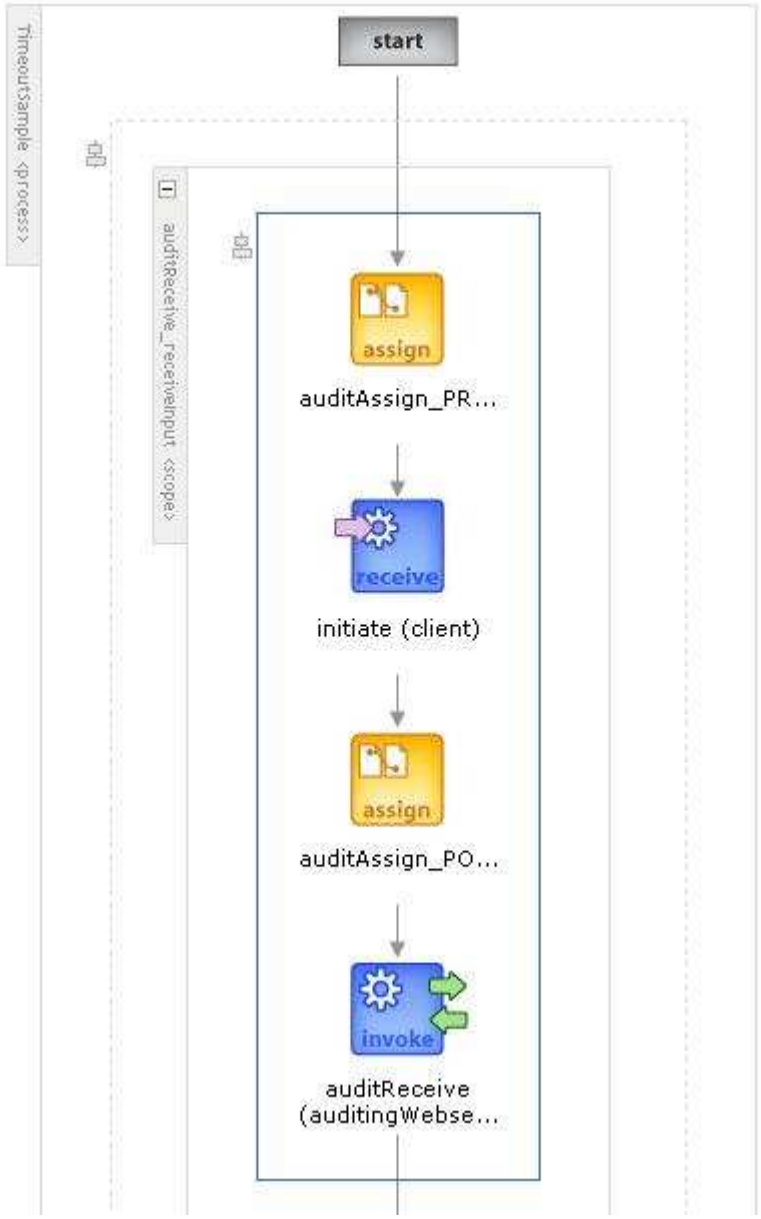


Figure 23: Processed receive activity

Figure 24 contains another interesting detail; the *postaudit* variable assignment for the invocation of a web service called *LoanService* is missing here. Source 35 shows the WSBPEL definition for the activity and the reason for the omitted assignment: no outputVariable has been defined, that means that the content of the audit variable can't changed at all and no new assignment is necessary.

```
<invoke name="invokeSL" partnerLink="LoanService"
portType="services:LoanService" operation="initiate"
inputVariable="loanApplication" />
```

Source 35: Invoke activity definition for initiate (LoanService)

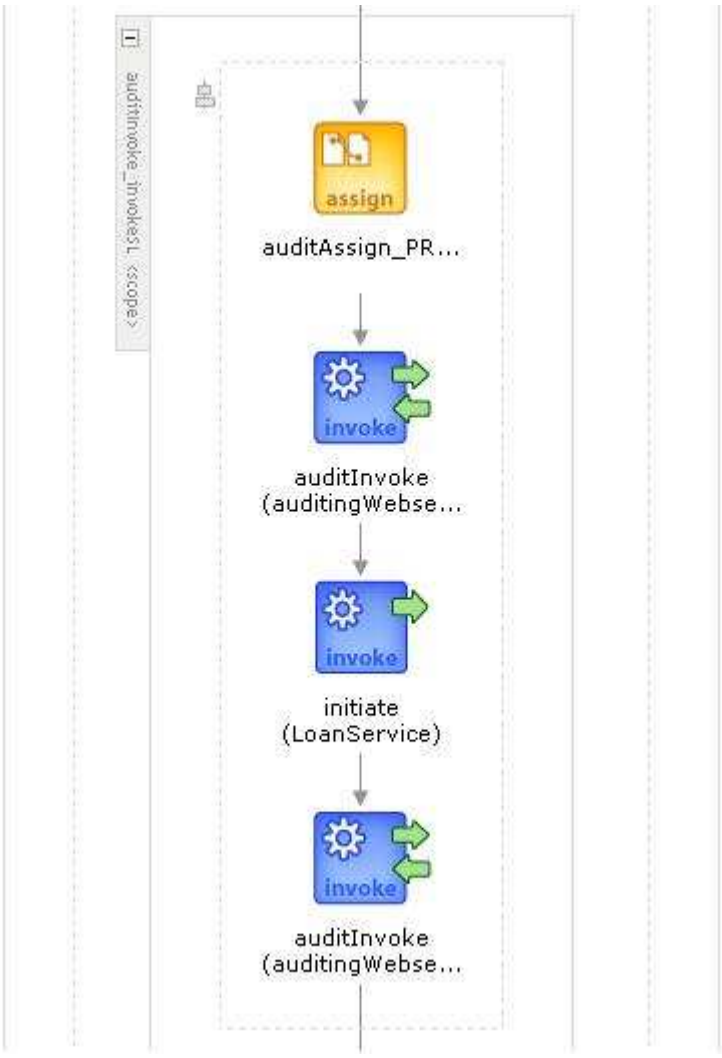


Figure 24: Processed invoke activity.

The listing in Table 3 shows which information about each one of the observed activities is available and will be sent to the auditing web service while executing the auditable new version of a WSBPEL process.

The following sections briefly study any special characteristics of a single activity and point out *when* information about a specific type of activity is sent to the auditing web service during the execution of a process.

Activity	Name	partner-Link	portType	operation	(input-) Variable-Name	(input-) Variable-Value	output-Variable-Name	output-Variable-Value	correlations	*Additionally reported items*
Receive	✓	✓	✓	✓	✓	✓	×	×	✓	
Reply	✓	✓	✓	✓	✓	✓	×	×	✓	faultName
Invoke	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Pick	✓	×	×	×	×	×	×	×	✓	
Pick – onMessage	✓	✓	✓	✓	✓	✓	×	×	✓	
Pick – onAlarm	✓	×	×	×	×	×	×	×	✓	durations
Throw	✓	×	×	×	faultVar	faultVal	×	×	✓	faultName
Switch	✓	×	×	×	×	×	×	×	✓	
Switch – Case	✓	×	×	×	×	×	×	×	✓	condition
Switch – Otherwise	✓	×	×	×	×	×	×	×	✓	

Table 3: List of audited attributes.

5.2.1.1 Receive

The *receive* activity is one of the two activities besides the *pick* activity that can be used as a starting point to instantiate a new process instance. In this case, *preaudit* is omitted.

Pre- and *postaudit* can otherwise be easily arranged before and after the audited activity.

```
** PRE_AUDIT ? **

<receive partnerLink="ncname" portType="qname" operation="ncname"
           variable="ncname"? createInstance="yes|no"?
           standard-attributes>
    standard-elements
    <correlations>?
       <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
</receive>

** POST_AUDIT **
```

Source 36: Arrangement of pre- and postaudit for a receive activity.

5.2.1.2 Reply

A *faultName* is reported additionally to the attributes of a *receive* activity, no further exceptions.

```
** PRE_AUDIT **

<reply partnerLink="ncname" portType="qname" operation="ncname"
         variable="ncname"? faultName="qname"?
         standard-attributes>
    standard-elements
    <correlations>?
       <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
</reply>

** POST_AUDIT **
```

Source 37: Arrangement of pre- and postaudit for a reply activity.

5.2.1.3 Invoke

The invoke activity is the only activity that makes use of two different variables, hence an input- as well as an output variable has to be taken care of.

```
** PRE_AUDIT **

<invoke partnerLink="ncname" portType="qname" operation="ncname"
          inputVariable="ncname"? outputVariable="ncname"?
          standard-attributes>
    standard-elements
    <correlations>?
       <correlation set="ncname" initiate="yes|no"?
                    pattern="in|out|out-in"/>+
    </correlations>
    <catch faultName="qname" faultVariable="ncname"?>*
      activity
    </catch>
    <catchAll>?
```

```
      activity
    </catchAll>
    <compensationHandler>?
      activity
    </compensationHandler>
</invoke>

** POST_AUDIT **
```

Source 38: Arrangement of pre- and postaudit for an invoke activity.

5.2.1.4 Pick

The pick activity can also be used instantiate a new process instance and therefore, *preaudit* can be omitted if the *createInstance* attribute is set to *yes*.

Another special property that it shares with the *switch* activity is, that *preaudit* can appear in two different places. Depending on which onMessage branch is triggered or if the onAlarm activity goes off first, the *postaudit* nested within that branch is called and the according auditing web service method is invoked.

```
** PRE_AUDIT ? **

<pick createInstance="yes|no"? standard-attributes>
    standard-elements
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname" variable="ncname"?>+
      <correlations>?
         <correlation set="ncname" initiate="yes|no"?>+
      </correlations>

      ** POST_AUDIT **

      activity
    </onMessage>
    <onAlarm (for="duration-expr" | until="deadline-expr")>*

      ** POST_AUDIT **

      activity
    </onAlarm>
</pick>
```

Source 39: Arrangement of pre- and postaudit for a pick activity.

Additionally to the usually reported attributes, either the *for* duration or the *until* duration will be reported to the auditing web service.

5.2.1.5 Throw

As stated earlier, a *throw* activity has no use for a *postaudit* as it will never be reached and is therefore omitted.

```
** PRE_AUDIT **

<throw faultName="qname" faultVariable="ncname"? standard-attributes>
    standard-elements
</throw>
```

Source 40: Arrangement of pre- and postaudit for a throw activity.

Furthermore, additionally to the name and the available correlations a throw statement has a given *faultName* and optionally a *faultVariable* which will be reported to the auditing web service.

5.2.1.6 Switch

Similar to the pick activity, *postaudit* can take place in several places and not just be placed after the whole activity because important runtime information would then be lost.

```
** PRE_AUDIT **

<switch standard-attributes>
    standard-elements
    <case condition="bool-expr">+

      ** POST_AUDIT **

      activity
    </case>
    <otherwise>?

      ** POST_AUDIT **

      activity
    </otherwise>
</switch>
```

Source 41: Arrangement of pre- and postaudit for a switch activity.

5.2.2 Enhancing auditing to WSBPEL using XSLT

While the last section is concerned with *what* is to be audited, this subchapter deals with *how* this transformation from the original process definition to an auditable version of it is done.

Although the proposed solution is applicable to any WSBPEL engine, small adoptions may need to be done since the contents of a WBSPEL project, especially the file containing the WSDL interface for this process, may differ in document structure and naming.

To help the reader to better understand the changes made, this section will start with a short introduction to Oracle BPEL Process Manager and the parts of a WSBPEL project built with it.

Afterwards, the parts which need to be changed and finally how they are changed will be described.

5.2.2.1 Oracle BPEL Process Manager Projects

The infrastructure needed to design, deploy and administrate WSBPEL processes with Oracle BPEL Process Manager was already described shortly in section 5.1. Since the proposed solution is to be applied before the process gets deployed, the Oracle BPEL PM Designer becomes the interesting part of this software.

After a new project has been created and the WSDL location of the first partner link was added, the file structure shown in Figure 25 contains all the information necessary to build and deploy the process.

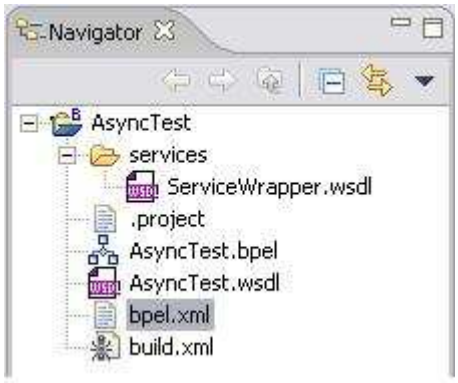


Figure 25: Oracle BPEL PM project contents.

The following table explains the content of each of these files, the directory *services* and its content is missing though, since it is the result of the already added partner WSDL location.

File	Description
.project	Eclipse BPEL Designer format project file.
bpel.xml	The deployment descriptor for the process. This file defines the locations of the WSDL files for services called by this process flow, along with other project-specific parameters.
build.xml	Apache Ant script for compiling and deploying this process.
AsyncTest.bpel	The BPEL source for the process. The New Project wizard creates an empty flow, with just the minimum activities and definitions for the selected flow type. For a synchronous BPEL process, the only activities are a receive activity to initiate the flow from a synchronous client request and a reply activity to return the output.
AsyncTest.wsdl	The WSDL (client) interface for this process. This file defines the input and output messages for this flow, the client interface and operations supported, and the BPEL partnerLinkTypes, so the flow can be incorporated into other processes. The New Project wizard generates a document-literal style WSDL that takes a string input message and returns a string response message.

Source: Bradshaw, Kennedy & West, 2005, Table 3-1.

Table 4: Oracle BPEL PM project files.

Using the Oracle BPEL PM Designer, the most important parts for a developer are the WSBPEL source file, the WSDL interface and the deployment descriptor. The client interface for this process does not need to be changed when creating an auditable version of this process. This leaves us with necessary adoptions at the WSBPEL source file and the deployment descriptor where location for the WSDL description of the auditing web service needs to be added.

The details on the structure of a WSBPEL source file were already described in chapter 3, therefore no further explanation about its content will be given in this section.

The deployment descriptor file in Figure 25 contains the WSDL locations listed in Source 42. The first WSDL file, AsyncTest.wsdl, describes the client interface of this process while the second entry defines the location of the WSDL description of a partner used during this process and points to another external file called *ServiceWrapper.wsdl*.

```
<?xml version="1.0"?>
<BPELSuitcase>
  <BPELProcess id="AsyncTest" src="AsyncTest.bpel">
    <partnerLinkBindings>
      <partnerLinkBinding name="client">
        <property name="wsdlLocation">
          AsyncTest.wsdl
        </property>
```

```
        </partnerLinkBinding>
        <partnerLinkBinding name="partnerLink">
          <property name="wsdlLocation">
            services/ServiceWrapper.wsdl
          </property>
        </partnerLinkBinding>
      </partnerLinkBindings>
    </BPELProcess>
  </BPELSuitcase>
```

Source 42: Oracle BPEL Process Manager, deployment descriptor example.

The actual binding information for the partner web service is found in this service wrapper file, an example is shown in Source 43:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions
  targetNamespace="http://hro.at/webservices/samples"
  xmlns:tns="http://hro.at/webservices/samples"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <import
    location="http://localhost/SomeWebService/Service.asmx?WSDL"/>
  <plnk:partnerLinkType name="ServiceSoapLink">
    <plnk:role name="ServiceSoapProvider">
      <plnk:portType name="tns:ServiceSoap" />
    </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```

Source 43: Oracle BPEL Process Manager, service wrapper example.

5.2.2.2 Creating an Auditable Version of a WSBPEL Process

After a business process has been successfully designed and captured with WSBPEL, the created source file containing the process definition as well as information on the WSDL locations of any invoked web service must be available in some way.

To create an auditable version of this WSBPEL process, the proposed solution basically needs to make two major changes to the existing definition

1. Add the WSDL location of the auditing web service to the deployment descriptor
2. Extend the WSBPEL source file

The first step adds the auditing web service to the list of known web services and enables the process to invoke the provided auditing operations.
During the second step, the existing WSBPEL process definition is extended to include the new partner link and add the *pre-* and *postaudit* activities as illustrated in Figure 16.

5.2.2.2.1 Adding the WSDL interface for the Auditing Web Service

In case of Oracle BPEL Process Manager, the information about the captured process will be included in the files described in section 5.2.2.1. Thus, the WSDL locations of invoked web services are listed in the deployment descriptor.

To add the WSDL location of the auditing web service to the example shown in Source 42, a service wrapper file has been created and is included as another partner link in the deployment descriptor.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions
  targetNamespace="http://hro.at/bpel/auditing/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:audit="http://hro.at/bpel/auditing/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <import
    location="http://localhost/auditingWebservice/Service.asmx?WSDL"/>
  <plnk:partnerLinkType name="ServiceSoapLink">
    <plnk:role name="ServiceSoapProvider">
      <plnk:portType name="audit:ServiceSoap" />
    </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```

Source 44: Oracle BPEL Process Manager, service wrapper for the auditing web service.

The new service wrapper file for the developed prototype is shown in Source 44 and was named *audit_ServiceWrapper.wsdl*. Like other partner links added to an Oracle BPEL PM Designer project, the file is placed in the *services* directory.

Finally, the new service wrapper is included in the deployment descriptor for this Oracle BPEL Process Manager project as shown in Source 45.

```
<?xml version="1.0"?>
<BPELSuitcase>
  <BPELProcess id="AsyncTest" src="AsyncTest.bpel">
    <partnerLinkBindings>
      <partnerLinkBinding name="client">
        <property name="wsdlLocation">
          AsyncTest.wsdl
        </property>
      </partnerLinkBinding>
      <partnerLinkBinding name="partnerLink">
        <property name="wsdlLocation">
          services/ServiceWrapper.wsdl
        </property>
      </partnerLinkBinding>
      <partnerLinkBinding name="auditingWebservice">
        <property name="wsdlLocation">
          services/audit_ServiceWrapper.wsdl
        </property>
      </partnerLinkBinding>
    </partnerLinkBindings>
  </BPELProcess>
</BPELSuitcase>
```

Source 45: Oracle BPEL Process Manager, deployment descriptor containing the auditing web service WSDL location.

Part of the proposed solution is a short XSLT file named *plink.xsl* which automatically processes a deployment descriptor and adds the new *partnerLinkBinding*. It can be found in the appendix.

5.2.2.2.2 Extending the WSBPEL Definition to Submit Audit Information

The second part of the proposed solution extends the WSBPEL source file itself, including the following steps:

- add the auditing namespace needed for variable declaration
- add the auditing web service partner link
- extend each audited activity
    - create local variables
    - assign values for *preaudit*
    - invoke the auditing web service for *preaudit*
    - keep the audited activity unchanged
    - assign values for *postaudit* if necessary
    - invoke the auditing web service for *postaudit*

Since the WSDL location of the auditing web service was added in the last step, a new partner link can be added to the WSBPEL source file and from that point on, the *pre-* and *postaudit invoke* statements to the auditing web service can be used.

```
<partnerLinks>
       <partnerLink name="client"
             partnerLinkType="tns:TimeoutSample"
             myRole="TimeoutSampleProvider"
             partnerRole="TimeoutSampleRequester" />
       <partnerLink name="LoanService"
             partnerLinkType="services:LoanService"
             myRole="LoanServiceRequester"
             partnerRole="LoanServiceProvider" />
       <partnerLink name="auditingWebservice"
             partnerLinkType="audit:ServiceSoapLink"
             partnerRole="ServiceSoapProvider" />
 </partnerLinks>
```

Source 46: Including the auditing web service partner link in the WSBPEL source file

Figure 26 shows the result as presented by Oracle's PM Designer in the *Overview* screen for a WSBPEL process, after the new partner link has been added.

Figure 26: Included auditing web service partner as shown in Oracle's BPEL PM Designer.

Finally, after the auditing web service has been added to the process and may be invoked just like any other web service, the actual auditing steps can be included.

Every activity extension is included within a single new scope for reasons of WSBPEL specification compliancy and simplicity.

It is necessary for compliancy, because the specific activity may be the only activity at a given location within the process definition and no surrounding *sequence* element would be needed for the original definition. If that was the case and the *pre-* and *postaudit* activities were placed before and after the audited entity, the WSBPEL specification would now require a *sequence* element.

Furthermore, it is important to create local variables for each audited activity, because WSBPEL processes can contain simultaneously processed activities and the use of several *pre-* and *postaudit* variables could overlap resulting in variables being assigned and assigned again before the auditing web service request is completed.

And last but not least simplicity is given in the resulting process definition which are better human readable if scopes are used to separate the extended activities from the rest of the original process definition.

The next two sections will give an insight on how exactly the audited activities will be extended. Two examples will be used to illustrate this task. First, the straightforward case of an *invoke* activity and afterwards the more complex extension regarding the *postaudit* part of a *pick* activity will be presented.

5.2.2.2.2.1 Creating and using local variables for auditing

The message types of the locally declared variables which are used to store the audit data originate from the auditing web service described in section 5.2.3. Basically, there are two different kinds of underlying data formats, a simple string type and one type that suits any XML input which is used to store variable content.

Unfortunately a small workaround is necessary to be able to assign a whole variable of the audited activity to the reserved part of the variable used for auditing.

> *"The from-spec is a variable of a WSDL message type and the to-spec is not, or vice versa. This is not legal because parts of variables, selections of variable parts, or endpoint references cannot be assigned to/from variables of WSDL message types directly."*[59]

To solve this problem, a dummy method is provided by the auditing web service to create a new message type which solely consists of one part that takes any XML input. The audited variable is first assigned to the helping variable of that message type after it which its content can be assigned again to the meant part for it of the variable used to invoke the auditing web service.

Source 47 shows the part of the XSLT template used for *invoke* activities[60] that creates the local variables within a new scope. The variables *auditInvokeIn* and *auditInvokeOut* are used during the invocation of the auditing web service.[61] The helping variable which will later be used to assign the variable values of the audited invoke activity to *auditInvokeIn* is called *auditAny*.

---

[59] Compare Andrews et al., 2003, section 9.3.1.

[60] The declaration and use of variables for *invoke* and *pick* activities works analogous, therefore, only one example will be given.

[61] Compare chapter for details on the use of the *invoke* activity in WSBPEL.

```
<xsl:template match="bpel:invoke">
  <xsl:element name="scope">
    <xsl:attribute name="name">
      auditInvoke_<xsl:value-of select="@name"/>
    </xsl:attribute>
    <!-- declare variables -->
    <xsl:element name="variables">
      <xsl:element name="variable">
        <xsl:attribute name="name">auditInvokeIn</xsl:attribute>
        <xsl:attribute
          name="messageType">audit:auditInvokeSoapIn</xsl:attribute>
      </xsl:element>
      <xsl:element name="variable">
        <xsl:attribute name="name">auditInvokeOut</xsl:attribute>
        <xsl:attribute name="messageType">
          audit:auditInvokeSoapOut
        </xsl:attribute>
      </xsl:element>
      <xsl:element name="variable">
        <xsl:attribute name="name">auditAny</xsl:attribute>
        <xsl:attribute name="element">audit:auditAny</xsl:attribute>
      </xsl:element>
    </xsl:element>
[…]
```

Source 47: Variable declaration for an invoke activity (audit_bpel.xsl).

Afterwards, depending on the activity which is being audited, the input variable is assigned with the values of that activity required to be reported during *pre-* and *postaudit* as listed in section 5.2.1. The assignment of the reported string and XML elements is shown in Source 48, where for example the activity name and the input variable name represent entities stored as strings whereas the XML content of the input variable is assigned as a whole using the workaround described above.
The *if* clause is necessary because the WSBPEL specification allows a missing input variable which would result in an empty, invalid *from* statement.

```
<xsl:element name="assign">
  <xsl:attribute name="name">auditAssign_PRE_<
    xsl:value-of select="@name"/>
  </xsl:attribute>
  <!-- inputvariablename -->
  <xsl:element name="copy">
    <xsl:element name="from">
      <xsl:attribute name="expression">
        '<xsl:value-of select="@inputVariable"/>'
      </xsl:attribute>
    </xsl:element>
    <xsl:element name="to">
      <xsl:attribute name="variable">auditInvokeIn</xsl:attribute>
      <xsl:attribute name="part">parameters</xsl:attribute>
      <xsl:attribute name="query">
        /audit:auditInvoke/audit:inputVariableName
       </xsl:attribute>
    </xsl:element>
  </xsl:element>
  <!-- inputvariablevalue - may be missing -->
  <xsl:if test="@inputVariable">
    <xsl:element name="copy">
      <xsl:element name="from">
        <xsl:attribute name="variable">
          <xsl:value-of select="@inputVariable"/>
```

```
            </xsl:attribute>
          </xsl:element>
          <xsl:element name="to">
            <xsl:attribute name="variable">auditAny</xsl:attribute>
          </xsl:element>
        </xsl:element>
        <xsl:element name="copy">
          <xsl:element name="from">
            <xsl:attribute name="variable">auditAny</xsl:attribute>
          </xsl:element>
          <xsl:element name="to">
            <xsl:attribute name="variable">auditInvokeIn</xsl:attribute>
            <xsl:attribute name="part">parameters</xsl:attribute>
            <xsl:attribute name="query">
              /audit:auditInvoke/audit:inputVariableValue
            </xsl:attribute>
          </xsl:element>
        </xsl:element>
      </xsl:if>

      […]
```

Source 48: Assigning values to the auditing variables (audit_bpel.xsl).

5.2.2.2.2.2 Invoking the auditing web service

After all assignments for *preaudit* have been made, the input variable is ready to be sent to the auditing web service. Source 49 shows how this is done for an *invoke* activity. The six steps described in 5.2.1 can be applied straightforward to audit this kind of activity, i.e. after the following regular activity the changed values are assigned to the audit variable and the auditing web service is invoked again to finish the *postaudit* part.

```
<xsl:element name="invoke">
  <xsl:attribute name="name">
    auditInvoke_PRE_<xsl:value-of select="@name"/>
  </xsl:attribute>
  <xsl:attribute name="partnerLink">auditingWebservice</xsl:attribute>
  <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
  <xsl:attribute name="operation">auditInvoke</xsl:attribute>
  <xsl:attribute name="inputVariable">auditInvokeIn</xsl:attribute>
  <xsl:attribute name="outputVariable">auditInvokeOut</xsl:attribute>
  <xsl:if test="bpel:correlations">
    <xsl:element name="correlations">
      <xsl:for-each select="bpel:correlations/bpel:correlation">
        <xsl:element name="correlation">
          <xsl:attribute name="set">
            <xsl:value-of select="@set"/>
          </xsl:attribute>
          <xsl:attribute name="initiate">no</xsl:attribute>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:if>
</xsl:element>
```

Source 49: Invoking the auditing web service to report an *invoke* activity.

In the comparison shown in Table 2, three activities are marked to deviate from this stepwise extension. As an example for those, the pick activity will be examined in more detail now because it's not only an exception regarding the *prepost* auditing. The possibilities it offers to either wait for a message or for an alarm to go off result in various potential branches for *postaudit*.

As stated earlier, the *pick* activity can be used to create a new process instance whenever a certain message arrives, making this very *pick* activity the first part of the process to be executed. Therefore, it is not allowed to have any other activities preceding it which requires the test on the *createInstance* attribute shown in Source 50.

```
<xsl:template match="bpel:pick">
  <xsl:choose>
    <xsl:when test="@createInstance and (@createInstance='yes')">
      <!-- keep process related invoke activity unchanged -->
      <xsl:copy>
        <xsl:apply-templates select="@* | node()"/>
      </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:element name="scope">
        <xsl:attribute name="name">
          auditPick_<xsl:value-of select="@name"/>
        </xsl:attribute>
     […]
```

Source 50: Preaudit xslt for a pick activity.

The second speciality needs to be covered during *postaudit*. A pick activity can include an alarm timer and multiple message ports and it cannot be known which one of them should be force the auditing web service to be invoked. Therefore, each one of the existing branches needs to be dealt with.
The developed prototype includes the *postaudit* parts as illustrated in Source 39, becoming the first activity within the specific branch. To do so, two new templates for *onAlarm* and *onMessage* as shown in Source 51 have been created to host the postaudit pieces of which one is usually going to be triggered.

```
<xsl:template match="bpel:onMessage">
  <xsl:element name="onMessage">
    <xsl:attribute name="partnerLink">
      <xsl:value-of select="@partnerLink"/>
    </xsl:attribute>
    <xsl:attribute name="portType">
      <xsl:value-of select="@portType"/>
    </xsl:attribute>
    <xsl:attribute name="operation">
      <xsl:value-of select="@operation"/>
    </xsl:attribute>
    <xsl:if test="@variable">
      <xsl:attribute name="variable">
        <xsl:value-of select="@variable"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:element name="scope">
      <xsl:attribute name="name">
        auditPickOnMessage_<xsl:value-of select="../@name"/>
      </xsl:attribute>
      <xsl:element name="variables">
        […]
      </xsl:element>
      <xsl:element name="sequence">
        <!-- assign values to invoke variable -->
        <xsl:element name="assign">
          <xsl:attribute name="name">
            auditAssign_POST_<xsl:value-of select="../@name"/>
```

```
            </xsl:attribute>
            <!-- activity name -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">
                  '<xsl:value-of select="../@name"/>'
              </xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">
                  auditPickOnMessageIn
                </xsl:attribute>
                <xsl:attribute name="part">
                  parameters
                </xsl:attribute>
                <xsl:attribute name="query">
                  /audit:auditPickOnMessage/audit:name
                </xsl:attribute>
              </xsl:element>
            </xsl:element>
[…]
```

Source 51: *OnMessage* part of a pick *postaudit.*

Switch is another activity whose *postaudit* requires the same setup as described in 5.2.1.6 and works similar with its underlying *case* and *otherwise* parts.

5.2.3 Auditing web service

The auditing web service is invoked whenever a *pre-* or *postaudit* part of an audited activity is reached. It provides an operation for each one of the audited activities described in section 5.2.1 to reduce the amount of work for parsing the incoming data figure out what kind of activity has been reported.

The web service is somehow stateless as far as the auditable version of the WSBPEL process is concerned in that it doesn't care about the content of the incoming requests containing the audit information.

All it does is accepting the information, reassembling the data in an XML document and distributing the results.

Because the auditing web service does not maintain a state about any of the former requests, it is not possible to make any assumptions about the next one to come other than from the content of this package. Therefore, the auditing web service should be seen as some kind of adapter for distributing the audited information to the next stage

The auditing web service prototype is programmed in C# and tested with a Microsoft Internet Information Server.

5.2.3.1 Receiving audit information

The auditing web service provides an operation for each one of the audited activities presented in section 5.2.1. The parameters defined here become the message types used to declare the variables for invoking the auditing web service. The parameters types are either strings or *XmlAnyElements* which can be used to store any type of XML content of a WSBPEL variable.

In the following, the different web service operations which are called by the auditable version of a WSBPEL process are listed for the sake of completeness.

Another method not covered here is the dummy method, which is used to create a special message type for the workaround described in section 5.2.2.2.2.1

5.2.3.1.1 Receive

```
[WebMethod]
public void auditReceive(
      string name,
      string partnerLink,
      string portType,
      string operation,
      string variableName,
      [XmlAnyElement("correlations")] XmlElement correlations,
      [XmlAnyElement("variableValue")] XmlElement variableValue)
{
}
```

5.2.3.1.2 Reply

```
[WebMethod]
public void auditReply(
      string name,
      string partnerLink,
      string portType,
      string operation,
      string variableName,
      string faultName,
      [XmlAnyElement("correlations")] XmlElement correlations,
      [XmlAnyElement("variableValue")] XmlElement variableValue)
{
}
```

5.2.3.1.3 Invoke

```
[WebMethod]
public void auditInvoke(
      string name,
      string partnerLink,
      string portType,
      string operation,
      string inputVariableName,
      string outputVariableName,
      [XmlAnyElement("correlations")] XmlElement correlations,
      [XmlAnyElement("inputVariableValue")] XmlElement
            inputVariableValue,
      [XmlAnyElement("outputVariableValue")] XmlElement
            outputVariableValue)
{
}
```

5.2.3.1.4 Pick

```
[WebMethod]
public void auditPick(
      string name,
      [XmlAnyElement("correlations")] XmlElement correlations)
{
}
```

```
[WebMethod]
public void auditPickOnMessage(
      string name,
      string partnerLink,
      string portType,
      string operation,
      string variableName,
      [XmlAnyElement("correlations")] XmlElement correlations,
      [XmlAnyElement("variableValue")] XmlElement variableValue)
{
}

[WebMethod]
public void auditPickOnAlarm(
      string name,
      string forDuration,
      string untilDuration,
      [XmlAnyElement("correlations")] XmlElement correlations)
{
}
```

5.2.3.1.5 Throw

```
[WebMethod]
public void auditThrow(
      string name,
      string faultName,
      string faultVariableName,
      [XmlAnyElement("correlations")] XmlElement correlations,
      [XmlAnyElement("faultVariableValue")] XmlElement
            faultVariableValue)
{
}
```

5.2.3.1.6 Switch

```
[WebMethod]
public void auditSwitch(
      string name,
      [XmlAnyElement("correlations")] XmlElement correlations)
{
}

[WebMethod]
public void auditSwitchCase(
      string name,
      string condition,

      [XmlAnyElement("correlations")] XmlElement correlations)
{
}

[WebMethod]
public void auditSwitchOtherwise(
      string name,
      [XmlAnyElement("correlations")] XmlElement correlations)
{
}
```

5.2.3.1.7 Dummy method: auditAny

As stated earlier, a workaround is necessary to assign variable values to the invoke variables used for auditing. This method is used to create the message type which can be used by the auditable version of the WSBPEL process.

```
[WebMethod]
public void auditAny(
      [XmlAnyElement("anyValue") ] XmlElement anyValue)
{
}
```

5.2.3.2 Reassembling Audit Information

The audited information can now easily be consolidated in a XML document to be communicated to interested parties.

A possible XML Schema for that purpose is shown in Figure 27, which accepts data on a single audited activity. As described in Table 3, the only two attributes shared by all activity types are *name* and *correlations*, which have been grouped together for each type. The remaining attributes are associated with each activity on it's on.

As an example, the schema information for the audit type of a *receive* activity is highlighted, for further details please refer to the appendix where the complete XML Schema can be found.

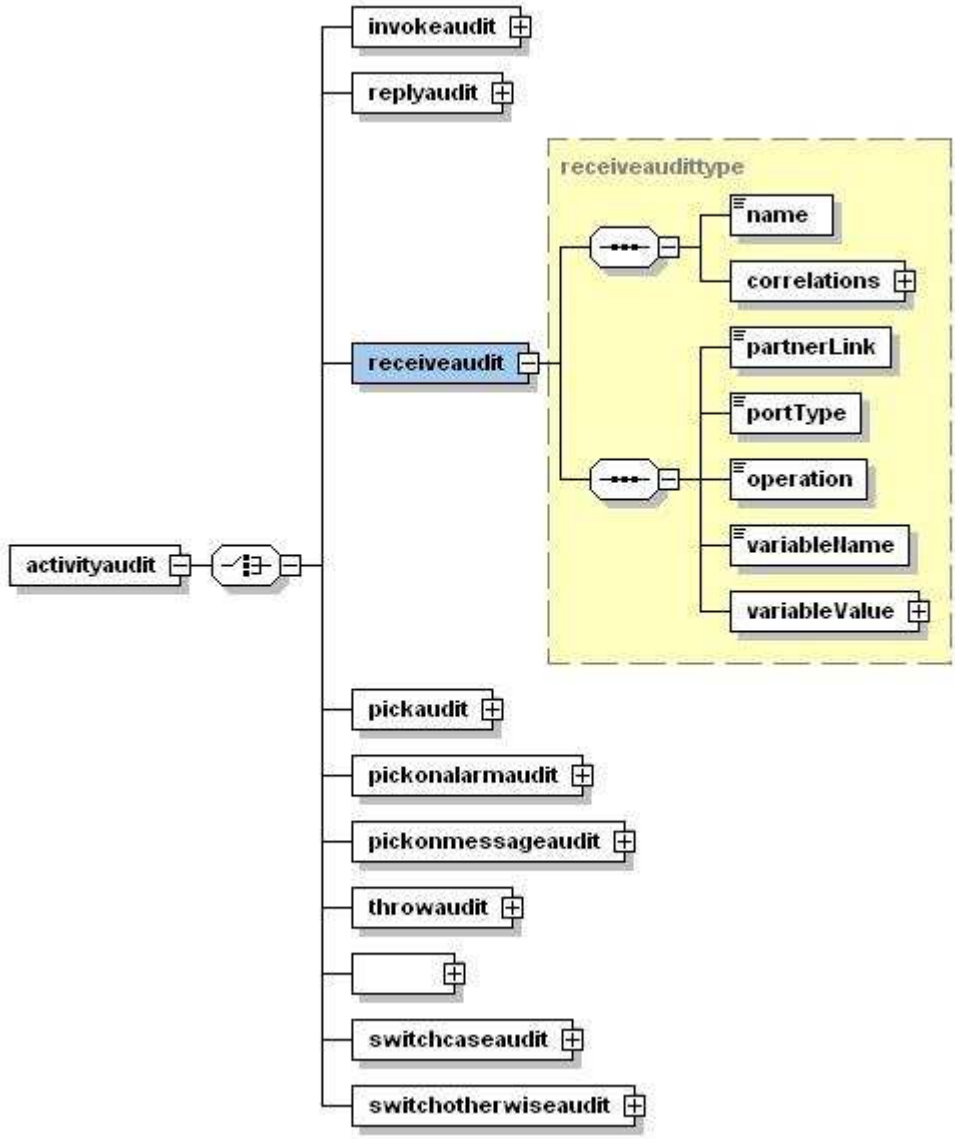


Figure 27: Possible XML Schema for audit information on WSBPEL activities.

6 Summary and Outlook

6.1 Summary

In the beginning of this thesis, the surrounding topics of WSBPEL as well as WSBPEL itself have been studied.

First, the main concept of BPM and the nature of business processes have been examined. The quest of an adequate BPM has become a main strategy for many companies during the last two decades. The optimisation of business processes throughout company borders has already leveraged a lot of potential in today's networked business environment. Nevertheless, it can be doubted that the full potential has already been tapped.

Furthermore, the need and the reasons of controlling in an organization have been discussed and it was argued that the importance of business process monitoring is growing. The use of information technology for process automation implicates a new information source for auditing which offers access to a vast amount of data and even continues to the point of real-time analyses of running process instances making it possible to intervene if necessary.

WSBPEL, as one of the state of the art technologies for facilitating the execution of business processes, has been studied in depth to provide a basis for the presentation of a prototype in the empirical part, together with the technologies on the lower layer of the web service stack on top of which WSBPEL is residing.

Finally, the empirical part introduces a solution to add WSBPEL compliant auditing support to an existing WSBPEL process. To do so, the WSBPEL process definition was extended to notify and provide auditing information to an external auditing web service. The collected information can then be further distributed in real-time by any available means of communication.

6.2 Future Work

The presented prototype is a solid solution to extend any WSBPEL conformant process definition automatically to an auditable version of the same process. The auditing web service receives valuable process audit trail information if a new process instances gets instantiated. The audited process data can be distributed to other systems that might be interested for evaluation.

Unfortunately, the loosely coupled infrastructure between the stateless auditing web service and its successors also is one of the biggest shortcomings of this design.

If the original process definition is well designed, e.g. consistent naming of activities or the use correlation sets, the reported *pre-* and *postaudit* activities can easily be grouped to the very activity and process instance they belong to, otherwise they are logged in some way anonymous. It is still possible to count and evaluate the content of each executed activity, but it's not always feasible to make assumptions about a whole process.

If the auditing web service had the possibility to track process instances through the use of its own correlation set, it would be simple to group each reported activity to its process instance.
The processing and analysis should logically remain outside the auditing web service to provide consistency and transparency.

A possible approach would be to add a new correlation set during the creation of an auditable process definition. Whenever a new process instance is first reported to the auditing web service,

a unique identifier could be returned by the analysis service and used to initiate the correlation set. The following activities belonging to that process instance can then be identified through this new correlation set.

7 References

Tony Andrews, Microsoft, Francisco Curbera, IBM, Hitesh Dholakia, Siebel Systems, Yaron Goland, BEA, Johannes Klein, Microsoft, Frank Leymann, IBM, Kevin Liu, SAP, Dieter Roller, IBM, Doug Smith, Siebel Systems, Satish Thatte, (Editor) Microsoft, Ivana Trickovic, SAP, Sanjiva Weerawarana, IBM: *Business Process Execution Language for Web Services Version 1.1*, Microsoft, 2003, available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/bpel1-1.asp [31.05.2005].

Assaf Arkin, Intalio,Sid Askary, Intalio, Scott Fordin, Sun Microsystems, Wolfgang Jekeli, SAP, Kohsuke Kawaguchi, Sun Microsystems, David Orchard, BEA Systems, Stefano Pogliani, Sun Microsystems, Karsten Riemer, Sun Microsystems, Susan Struble, Sun Microsystems, Pal Takacsi-Nagy, BEA Systems, Ivana Trickovic, SAP, Sinisa Zimek, SAP: *Web Service Choreography Interface (WSCI) 1.0*, W3C, 2002, available at: http://www.w3.org/TR/wsci/ [26.09.2005].

Adam Bosworth, BEA, Don Box, Microsoft (Editor), Erik Christensen, Microsoft, Francisco Curbera, IBM (Editor), Donald Ferguson, IBM, Jeffrey Frey, IBM, Chris Kaler, Microsoft, David Langworthy, Microsoft, Frank Leymann, IBM, Steve Lucco, Microsoft, Steve Millet, Microsoft, Nirmal Mukhi, IBM, Mark Nottingham, BEA, David Orchard, BEA, John Shewchuk, Microsoft, Tony Storey, IBM, Sanjiva Weerawarana, IBM: *Web Services Addressing (WS-Addressing)*, 2003, available at: http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-addressing0303.asp [11.07.2005].

Deanna Bradshaw, Mark Kennedy, Craig West: *Oracle® BPEL Process Manager: Developer's Guide*, 10g Release 2 (10.1.2), Oracle, 2005, available at: http://download-uk.oracle.com/otndocs/products/bpel/bpeldev.pdf [20.07.2005].

Mick Broadbent: Measuring business performance, CIMA, London, 1999.

James Bryce Clark, Luc Clement, Tony Rogers: *UDDI v3: The Registry Standard for SOA*, OASIS, 2005, available at: http://www.oasis-open.org/presentations/uddi_v3_webcast_20050222.pdf [16.10.2005].

Luis Felipe Cabrera, Microsoft, George Copeland, Microsoft, Max Feingold, Microsoft, (Editor) Robert Freund, Hitachi, Tom Freund, IBM, Jim Johnson, Microsoft, Sean Joyce, IONA, Chris Kaler, Microsoft, Johannes Klein, Microsoft, David Langworthy, Microsoft,, Mark Little, Arjuna Technologies, Anthony Nadalin, IBM, Eric Newcomer, IONA, David Orchard, BEA Systems, Ian Robinson, IBM, John Shewchuk, Microsoft, Tony Storey, IBM: Web *Services Coordination (WS-Coordination)*, Version 1.0, 2005, available at: ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf [05.10.2005].

Erik Christensen, Microsoft, Francisco Curbera, IBM Research, Greg Meredith, Microsoft, Sanjiva Weerawarana, IBM Research: *Web Services Description Language (WSDL) 1.1*, W3C, 2001, available at: http://www.w3.org/TR/wsdl [25.10.2005].

John Colgrave, IBM, Karsten Januszewski, Microsoft: *Technical Note: Using WSDL in a UDDI Registry*, Version 2.0.2, OASIS, 2004, available at: http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm [22.10.2005].

Thomas Davenport: *Process Innovation, Reengineering Work through Information Technology*, Harvard Business School Press, Boston (MA), 1993.

Tom DeMarco: Controlling *Software Projects, Management Measurement & Estimation*, Prentice Hall, 1982.

Asuman Dogac, Leonid Kalinichenko, M. Tamer Ozsu & Amit Sheth, editors: *Workflow Management Systems and Interoperability*, volume 164 of NATO Advanced Science Institutes (ASI), Series F:Computer and Systems Sciences, Springer-Verlag, 1998.

Rudolf Fiedler: Einführung *in das Controlling*, 2. Auflage, Oldenbourg Wissenschaftsverlag GmbH, München, 2001.

Peter Furniss: WSBPEL *Issues List, Issue 98: What version number are we working towards*, Choreology, 2004, available at: http://www.choreology.com/external/WS_BPEL_issues_list.html#Issue98 [10.08.2005].

Dimitrios Georgakopoulos & Aphrodite Tsalgatidou: *Technology and Tools for Comprehensive Business Process Lifecycle Management*, In Dogac et. al [DOGA98], 1998.

Michael Hammer & James Champy: *Reengineering the Corporation: A Manifesto for Business Revolution*, 1st ed., HarperBusiness, New York (NY), 1993.

Mark Hellinger, Scott Fingerhut: *Business Activity Monitoring: EAI Meets Data Warehousing*, eAI Journal, 2002.

P. Horváth : *Controlling*, 8th edition, München, 2002.

Kazunori Iwasa, Fujitsu Limited, Jacques Durand, Fujitsu Software Corporation, Tom Rutt, Fujitsu Software Corporation, Mark Peel, Novell, Inc., Sunil Kunisetty, Oracle, Doug Bunting, Sun Microsystems: *WS-Reliability 1.1*, OASIS, 2004, available at: http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf [26.09.2005].

Diane Jordan, John Evdemon: *OASIS Web Services Business Process Execution Language (WSBPEL) TC*, OASIS, 2005, available at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel [10.08.2005].

Matjaz B. Juric: *BPEL and Java*, TheServerSide.COM, 2005, available at: http://www.theserverside.com/articles/article.tss?l=BPELJava [14.06.2005].

Nickolas Kavantzas, Mike Lehmann: *BPEL: Building Standards-Based Business Processes with Web Services*, Oracle, 2003, available at: http://download.oracle.com/owsf_2003/40024.ppt [09.11.2005].

Martin Keen, Jonathan Cavell, Sarah Hill, Chee Keong Kee, Wendy Neave, Bradley Rumph, Hoang Tran: *BPEL4WS Business Processes with WebSphere Business Integration: Understanding, Modeling, Migrating*, 1st. ed., IBM, 2004, available at: http://www.redbooks.ibm.com/redbooks.nsf/0/9c602efe338cda6f85256ebb00471cf4?OpenDocument [01.08.2005].

Rania Khalaf, William Nagy: Business Process with BPEL4WS: *Learning BPEL4WS, Part 6, Correlation, fault handling, and compensation*, IBM, 2003, available at: http://www-128.ibm.com/developerworks/library/ws-bpelcol6/ [20.07.2005].

Rania Khalaf, William Nagy: Business Process with BPEL4WS: *Learning BPEL4WS, Part 7, Adding correlation and fault handling to a process*, IBM, 2003, available at: http://www-128.ibm.com/developerworks/webservices/library/ws-bpelcol7/ [20.07.2005].

Harpal Kochar: Oracle *Business Activity Monitoring, An Oracle White Paper*, Oracle, 2005, available at: http://www.oracle.com/technology/products/integration/bam/pdf/bam_whitepaper.pdf [28.07.2005].

G. Kramler, W. Retschitzegger: *Specification of Interorganizational Workflows - A Comparison of Approaches*, Institute of Software Technology and Interactive Systems, Business Informatics Group, Vienna University of Technology, 2002, available at: ftp://ftp.ifs.uni-linz.ac.at/pub/publications/2002/0702.pdf [10.08.2005].

Frank Leymann: *Web Services Flow Language, (WSFL 1.0)*, IBM, 2001, available at: http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf [10.08.2005].

Mark Little, Jim Webber: *Introducing WS-Coordination*, SYS-CON Media, 2003, available at: http://webservices.sys-con.com/read/39751.htm [10.10.2005].

Michael zur Mühlen: *Workflow-based Process Controlling: Foundation, Design, and Application of Workflow-driven Process Information Systems*, Logos Verlag, Berlin, 2004, available at: http://www.workflow-research.de/Publications/Book/index.html [24.05.2005].

Anthony Nadalin, IBM, Chris Kaler, Microsoft, Phillip Hallam-Baker, VeriSign, Ronald Monzillo, Sun: *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*, OASIS, 2004, available at: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf [05.10.2005].

Andy Neely: *Measuring business performance*, Economist Books, 1998.

OASIS: *Introduction to UDDI: Important Features and Functional Concepts*, OASIS, 2004, available at: http://uddi.org/pubs/uddi-tech-wp.pdf [22.10.2005].

Oracle: *Oracle BPEL Process Manager 10.1.2 Information, BPEL Samples*, Oracle, 2005, available at: http://www.oracle.com/technology/products/ias/bpel/htdocs/1012_support.html#samples [08.11.2005].

Chris Peltz: Web *services orchestration and choreography*, IEEE Computer Society, 2003, available at: http://ieeexplore.ieee.org/iel5/2/27718/01236471.pdf [10.08.2005].

Will Provost: *WSDL First*, 2003, available at: http://webservices.xml.com/pub/a/ws/2003/07/22/wsdlfirst.html [28.10.2005].

Anthony Reynolds*: Sensors in BPEL*, 2005, available at: http://www.orablogs.com/reynolds/archives/001286.html [28.07.2005].

Claus von Riegen, SAP, Ivana Trickovic, SAP: *Using BPEL4WS in a UDDI registry*, OASIS, 2004, available at: http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.htm [22.10.2005].

Mehmet Sayal, Fabio Casati, Umeshwar Dayal, Ming-Chien Shan: *Business Process Cockpit (Extended Abstract),* Proceedings of the 28th VLDB Conference, HP, 2002.

Alexander Schatten: *Sustainable Web-Based Organisation of Project-Related Information and Knowledge*, Institute for Software Technology and Interactive Systems, Vienna University of Technology, 2003, available at: http://www.schatten.info/pub/thesis/phd-thesis.pdf [10.11.2005].

Aaron Skonnard: *Understanding WSDL*, Northface University, 2003, available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/understandWSDL.asp [22.10.2005].

Brent Sleeper: *The Evolution of UDDI*, UDDI.org White Paper, The Stencil Group, Inc., 2002, available at: http://www.uddi.org/pubs/the_evolution_of_uddi_20020719.pdf [16.10.2005].

Adam Smith: *The Wealth of Nations*, 2nd volume, W. Strahan & T. Cadell, London, 1776.

Satish Thatte: XLANG*, Web Services for Business Process Design*, Microsoft, 2001, available at: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm#_Toc515125569 [10.08.2005].

Sam Thompson: *Implementing WS-Security, A case study*, IBM, 2003, available at: http://www-128.ibm.com/developerworks/webservices/library/ws-security.html [03.10.2005].

Jim Webber*: Introducing BPEL4WS 1.0*, 2003, available at: http://webservices.sys-con.com/read/39830.htm [20.07.2005].

Wikipedia contributors: *BPEL*, Wikipedia: The Free Encyclopedia, 2005, available at: http://en.wikipedia.org/wiki/BPEL [27.07.2005].

Scott Woodgate, Stephen Mohr, Brian Loesgen, Susie Adams, Alex Cobb, Benjamin Goeltz, Brandon Gross, Chris Whytock, Erik Leaseburg, Gavin Islip, Imran Aziz, Kevin Smith, Michael Roze, Naveen Goli, Puru Amradkar, and Stephen Roger: *BizTalk Server 2004 Unleashed*, 1st ed., Sams Publishing, Indianapolis, 2004.

8 Appendix

8.1 audit_ServiceWrapper.wsdl

```
<?xml version="1.0" encoding="utf-8"?>
<definitions
      targetNamespace="http://hro.at/bpel/auditing/"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:audit="http://hro.at/bpel/auditing/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
<import location="http://localhost/auditingWebservice/Service.asmx?WSDL"/>
      <plnk:partnerLinkType name="ServiceSoapLink">
            <plnk:role name="ServiceSoapProvider">
                  <plnk:portType name="audit:ServiceSoap" />
            </plnk:role>
      </plnk:partnerLinkType>
</definitions>
```

8.2 audit_plink.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:template match="@* | node()">
            <xsl:copy>
                  <xsl:apply-templates select="@* | node()"/>
            </xsl:copy>
      </xsl:template>
      <xsl:template match="//partnerLinkBindings">
            <xsl:element name="partnerLinkBindings">
                  <xsl:copy-of select="partnerLinkBinding"/>
                  <xsl:element name="partnerLinkBinding">
                        <xsl:attribute name="name">
                              auditingWebservice
                        </xsl:attribute>
                        <xsl:element name="property">
                              <xsl:attribute name="name">
                                    wsdlLocation
                              </xsl:attribute>
                              services/audit_ServiceWrapper.wsdl
                        </xsl:element>
                  </xsl:element>
            </xsl:element>
      </xsl:template>
</xsl:stylesheet>
```

8.3 audit_bpel.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:audit="http://hro.at/bpel/auditing/">
  <xsl:template match="@* | node()">
    <xsl:copy>
```

```
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
  <!-- ****** ADD auditing namespace ****** -->
  <xsl:template match="/bpel:process">
    <xsl:copy>
      <xsl:attribute
name="audit:dummyattribute">xslt1.0needsadummyattribute</xsl:attribute>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
  <!-- ****** ADD auditing partner link ****** -->
  <xsl:template match="/bpel:process/bpel:partnerLinks">
    <xsl:element name="partnerLinks">
      <xsl:copy-of select="bpel:partnerLink"/>
      <xsl:element name="partnerLink">
        <xsl:attribute name="name">auditingWebservice</xsl:attribute>
        <xsl:attribute
name="partnerLinkType">audit:ServiceSoapLink</xsl:attribute>
        <xsl:attribute
name="partnerRole">ServiceSoapProvider</xsl:attribute>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <!-- ****** AUDIT INVOKE ****** -->
  <xsl:template match="bpel:invoke">
    <xsl:element name="scope">
      <xsl:attribute name="name">auditInvoke_<xsl:value-of
select="@name"/></xsl:attribute>
      <!-- declare variables -->
      <xsl:element name="variables">
        <xsl:element name="variable">
          <xsl:attribute name="name">auditInvokeIn</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditInvokeSoapIn</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditInvokeOut</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditInvokeSoapOut</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditAny</xsl:attribute>
          <xsl:attribute name="element">audit:auditAny</xsl:attribute>
        </xsl:element>
      </xsl:element>
      <xsl:element name="sequence">
        <!-- PREactivity auditing -->
        <!-- assign values to invoke variable -->
        <xsl:element name="assign">
          <xsl:attribute name="name">auditAssign_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <!-- activity name -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@name"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditInvokeIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditInvoke/audit:name</xsl:attribute>
```

```
            </xsl:element>
          </xsl:element>
          <!-- partnerlink -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@partnerLink"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditInvokeIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditInvoke/audit:partnerLink</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- porttype -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@portType"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditInvokeIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditInvoke/audit:portType</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- operation -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@operation"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditInvokeIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditInvoke/audit:operation</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- inputvariablename -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@inputVariable"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditInvokeIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditInvoke/audit:inputVariableName</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- outputvariablename -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@outputVariable"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditInvokeIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
```

```
              <xsl:attribute
name="query">/audit:auditInvoke/audit:outputVariableName</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- inputvariablevalue - may be missing -->
          <xsl:if test="@inputVariable">
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable"><xsl:value-of
select="@inputVariable"/></xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditInvokeIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditInvoke/audit:inputVariableValue</xsl:attribute>
              </xsl:element>
            </xsl:element>
          </xsl:if>
          <!-- outputvariablevalue - may be missing -->
          <xsl:if test="@outputVariable">
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable"><xsl:value-of
select="@outputVariable"/></xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditInvokeIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditInvoke/audit:outputVariableValue</xsl:attribute>
              </xsl:element>
            </xsl:element>
          </xsl:if>
        </xsl:element>
        <!-- call auditingservice -->
        <xsl:element name="invoke">
          <xsl:attribute name="name">auditInvoke_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
          <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
          <xsl:attribute name="operation">auditInvoke</xsl:attribute>
          <xsl:attribute name="inputVariable">auditInvokeIn</xsl:attribute>
```

```
          <xsl:attribute
name="outputVariable">auditInvokeOut</xsl:attribute>
          <xsl:if test="bpel:correlations">
            <xsl:element name="correlations">
              <xsl:for-each select="bpel:correlations/bpel:correlation">
                <xsl:element name="correlation">
                  <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                  <xsl:attribute name="initiate">no</xsl:attribute>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:element>
        <!-- keep process related invoke activity unchanged -->
        <xsl:copy>
          <xsl:apply-templates select="@* | node()"/>
        </xsl:copy>
        <!-- POSTactivity auditing -->
        <!-- assign possibly changed values to invoke variable -->
        <!-- outputvariablevalue - may be missing -->
        <xsl:if test="@outputVariable">
          <xsl:element name="assign">
            <xsl:attribute name="name">auditAssign_POST_<xsl:value-of
select="@name"/></xsl:attribute>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable"><xsl:value-of
select="@outputVariable"/></xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditInvokeIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditInvoke/audit:outputVariableValue</xsl:attribute>
              </xsl:element>
            </xsl:element>
          </xsl:element>
        </xsl:if>
        <!-- call auditingservice -->
        <xsl:element name="invoke">
          <xsl:attribute name="name">auditInvoke_POST_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
          <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
          <xsl:attribute name="operation">auditInvoke</xsl:attribute>
          <xsl:attribute name="inputVariable">auditInvokeIn</xsl:attribute>
          <xsl:attribute
name="outputVariable">auditInvokeOut</xsl:attribute>
          <xsl:if test="bpel:correlations">
            <xsl:element name="correlations">
              <xsl:for-each select="bpel:correlations/bpel:correlation">
                <xsl:element name="correlation">
```

```
                  <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                  <xsl:attribute name="initiate">no</xsl:attribute>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <!-- ****** END AUDIT INVOKE ****** -->
  <!-- ****** AUDIT RECEIVE ****** -->
  <xsl:template match="bpel:receive">
    <xsl:element name="scope">
      <xsl:attribute name="name">auditReceive_<xsl:value-of
select="@name"/></xsl:attribute>
      <!-- declare variables -->
      <xsl:element name="variables">
        <xsl:element name="variable">
          <xsl:attribute name="name">auditReceiveIn</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditReceiveSoapIn</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditReceiveOut</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditReceiveSoapOut</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditAny</xsl:attribute>
          <xsl:attribute name="element">audit:auditAny</xsl:attribute>
        </xsl:element>
      </xsl:element>
      <xsl:element name="sequence">
        <!-- assign values to invoke variable -->
        <xsl:element name="assign">
          <xsl:attribute name="name">auditAssign_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <!-- activity name -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@name"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReceiveIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReceive/audit:name</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- partnerlink -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@partnerLink"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReceiveIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReceive/audit:partnerLink</xsl:attribute>
```

```
            </xsl:element>
          </xsl:element>
          <!-- porttype -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@portType"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReceiveIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReceive/audit:portType</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- operation -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@operation"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReceiveIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReceive/audit:operation</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- variablename -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@variable"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReceiveIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReceive/audit:variableName</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- variablevalue - may be missing -->
          <xsl:if test="@variable">
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable"><xsl:value-of
select="@variable"/></xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditReceiveIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditReceive/audit:variableValue</xsl:attribute>
              </xsl:element>
```

```
            </xsl:element>
          </xsl:if>
        </xsl:element>
        <xsl:choose>
          <xsl:when test="@createInstance or (@createInstance!='yes')">
            <!-- keep process related invoke activity unchanged -->
            <xsl:copy>
              <xsl:apply-templates select="@* | node()"/>
            </xsl:copy>
          </xsl:when>
          <xsl:otherwise>
            <!-- PREactivity auditing -->
            <!-- call auditingservice -->
            <xsl:element name="invoke">
              <xsl:attribute name="name">auditReceive_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
              <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
              <xsl:attribute
name="portType">audit:ServiceSoap</xsl:attribute>
              <xsl:attribute name="operation">auditReceive</xsl:attribute>
              <xsl:attribute
name="inputVariable">auditReceiveIn</xsl:attribute>
              <xsl:attribute
name="outputVariable">auditReceiveOut</xsl:attribute>
              <xsl:if test="bpel:correlations">
                <xsl:element name="correlations">
                  <xsl:for-each
select="bpel:correlations/bpel:correlation">
                    <xsl:element name="correlation">
                      <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                      <xsl:attribute name="initiate">no</xsl:attribute>
                    </xsl:element>
                  </xsl:for-each>
                </xsl:element>
              </xsl:if>
            </xsl:element>
            <!-- keep process related invoke activity unchanged -->
            <xsl:copy>
              <xsl:apply-templates select="@* | node()"/>
            </xsl:copy>
          </xsl:otherwise>
        </xsl:choose>
        <!-- POSTactivity auditing -->
        <!-- assign possibly changed values to invoke variable -->
        <xsl:if test="@variable">
          <xsl:element name="assign">
            <xsl:attribute name="name">auditAssign_POST_<xsl:value-of
select="@name"/></xsl:attribute>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable"><xsl:value-of
select="@variable"/></xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
```

```
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditReceiveIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditReceive/audit:variableValue</xsl:attribute>
              </xsl:element>
            </xsl:element>
          </xsl:element>
        </xsl:if>
        <!-- call auditingservice -->
        <xsl:element name="invoke">
          <xsl:attribute name="name">auditReceive_POST_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
          <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
          <xsl:attribute name="operation">auditReceive</xsl:attribute>
          <xsl:attribute
name="inputVariable">auditReceiveIn</xsl:attribute>
          <xsl:attribute
name="outputVariable">auditReceiveOut</xsl:attribute>
          <xsl:if test="bpel:correlations">
            <xsl:element name="correlations">
              <xsl:for-each select="bpel:correlations/bpel:correlation">
                <xsl:element name="correlation">
                  <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                  <xsl:attribute name="initiate">no</xsl:attribute>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <!-- ****** END AUDIT RECEIVE ****** -->
  <!-- ****** AUDIT REPLY ****** -->
  <xsl:template match="bpel:reply">
    <xsl:element name="scope">
      <xsl:attribute name="name">auditReply_<xsl:value-of
select="@name"/></xsl:attribute>
      <xsl:element name="variables">
        <xsl:element name="variable">
          <xsl:attribute name="name">auditReplyIn</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditReplySoapIn</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditReplyOut</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditReplySoapOut</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditAny</xsl:attribute>
          <xsl:attribute name="element">audit:auditAny</xsl:attribute>
        </xsl:element>
      </xsl:element>
      <xsl:element name="sequence">
        <!-- PREactivity auditing -->
        <!-- assign values to invoke variable -->
        <xsl:element name="assign">
```

```
          <xsl:attribute name="name">auditAssign_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <!-- activity name -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@name"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReplyIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReply/audit:name</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- partnerlink -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@partnerLink"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReplyIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReply/audit:partnerLink</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- porttype -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@portType"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReplyIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReply/audit:portType</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- operation -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@operation"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReplyIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReply/audit:operation</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- faultName -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@faultName"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReplyIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
```

```
              <xsl:attribute
name="query">/audit:auditReply/audit:faultName</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- inputvariablename -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@variable"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditReplyIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditReply/audit:variableName</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- variablevalue - may be missing -->
          <xsl:if test="@variable">
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable"><xsl:value-of
select="@variable"/></xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditReplyIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditReply/audit:variableValue</xsl:attribute>
              </xsl:element>
            </xsl:element>
          </xsl:if>
        </xsl:element>
        <!-- call auditingservice -->
        <xsl:element name="invoke">
          <xsl:attribute name="name">auditReply_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
          <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
          <xsl:attribute name="operation">auditReply</xsl:attribute>
          <xsl:attribute name="inputVariable">auditReplyIn</xsl:attribute>
          <xsl:attribute
name="outputVariable">auditReplyOut</xsl:attribute>
          <xsl:if test="bpel:correlations">
            <xsl:element name="correlations">
              <xsl:for-each select="bpel:correlations/bpel:correlation">
                <xsl:element name="correlation">
                  <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                  <xsl:attribute name="initiate">no</xsl:attribute>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
```

```
        </xsl:element>
        <!-- keep process related invoke activity unchanged -->
        <xsl:copy>
          <xsl:apply-templates select="@* | node()"/>
        </xsl:copy>
        <!-- POSTactivity auditing -->
        <!-- assign possibly changed values to invoke variable -->
        <!-- call auditingservice -->
        <xsl:element name="invoke">
          <xsl:attribute name="name">auditReply_POST_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
          <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
          <xsl:attribute name="operation">auditReply</xsl:attribute>
          <xsl:attribute name="inputVariable">auditReplyIn</xsl:attribute>
          <xsl:attribute
name="outputVariable">auditReplyOut</xsl:attribute>
          <xsl:if test="bpel:correlations">
            <xsl:element name="correlations">
              <xsl:for-each select="bpel:correlations/bpel:correlation">
                <xsl:element name="correlation">
                  <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                  <xsl:attribute name="initiate">no</xsl:attribute>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <!-- ****** END AUDIT REPLY ****** -->
  <!-- ****** AUDIT THROW ****** -->
  <xsl:template match="bpel:throw">
    <xsl:element name="scope">
      <xsl:attribute name="name">auditThrow_<xsl:value-of
select="@name"/></xsl:attribute>
      <xsl:element name="variables">
        <xsl:element name="variable">
          <xsl:attribute name="name">auditThrowIn</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditThrowSoapIn</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditThrowOut</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditThrowSoapOut</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditAny</xsl:attribute>
          <xsl:attribute name="element">audit:auditAny</xsl:attribute>
        </xsl:element>
      </xsl:element>
      <xsl:element name="sequence">
        <!-- PREactivity auditing -->
        <!-- assign values to invoke variable -->
        <xsl:element name="assign">
          <xsl:attribute name="name">auditAssign_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <!-- activity name -->
          <xsl:element name="copy">
```

```
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@name"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditThrowIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditThrow/audit:name</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- faultName -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@faultName"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditThrowIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditThrow/audit:faultName</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!-- faultvariablename -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@faultVariable"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditThrowIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditThrow/audit:faultVariableName</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <!--faultvariablevalue - may be missing -->
          <xsl:if test="@faultVariable">
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="faultVariable"><xsl:value-of
select="@faultVariable"/></xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="variable">auditAny</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute name="variable">auditThrowIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditThrow/audit:faultVariableValue</xsl:attribute>
              </xsl:element>
            </xsl:element>
          </xsl:if>
        </xsl:element>
        <!-- call auditingservice -->
        <xsl:element name="invoke">
```

```
          <xsl:attribute name="name">auditThrow_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
          <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
          <xsl:attribute name="operation">auditThrow</xsl:attribute>
          <xsl:attribute name="inputVariable">auditThrowIn</xsl:attribute>
          <xsl:attribute
name="outputVariable">auditThrowOut</xsl:attribute>
          <xsl:if test="bpel:correlations">
            <xsl:element name="correlations">
              <xsl:for-each select="bpel:correlations/bpel:correlation">
                <xsl:element name="correlation">
                  <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                  <xsl:attribute name="initiate">no</xsl:attribute>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:element>
        <!-- keep process related invoke activity unchanged -->
        <xsl:copy>
          <xsl:apply-templates select="@* | node()"/>
        </xsl:copy>
        <!-- POSTactivity auditing -->
        <!-- assign possibly changed values to invoke variable -->
        <!-- call auditingservice -->
        <xsl:element name="invoke">
          <xsl:attribute name="name">auditThrow_POST_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
          <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
          <xsl:attribute name="operation">auditThrow</xsl:attribute>
          <xsl:attribute name="inputVariable">auditThrowIn</xsl:attribute>
          <xsl:attribute
name="outputVariable">auditThrowOut</xsl:attribute>
          <xsl:if test="bpel:correlations">
            <xsl:element name="correlations">
              <xsl:for-each select="bpel:correlations/bpel:correlation">
                <xsl:element name="correlation">
                  <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                  <xsl:attribute name="initiate">no</xsl:attribute>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <!-- ****** END AUDIT THROW ****** -->
  <!-- ****** AUDIT SWITCH ****** -->
  <xsl:template match="bpel:switch">
    <xsl:element name="scope">
      <xsl:attribute name="name">auditSwitch_<xsl:value-of
select="@name"/></xsl:attribute>
      <xsl:element name="variables">
        <xsl:element name="variable">
          <xsl:attribute name="name">auditSwitchIn</xsl:attribute>
```

```
          <xsl:attribute
name="messageType">audit:auditSwitchSoapIn</xsl:attribute>
        </xsl:element>
        <xsl:element name="variable">
          <xsl:attribute name="name">auditSwitchOut</xsl:attribute>
          <xsl:attribute
name="messageType">audit:auditSwitchSoapOut</xsl:attribute>
        </xsl:element>
      </xsl:element>
      <xsl:element name="sequence">
        <!-- PREactivity auditing -->
        <!-- assign values to invoke variable -->
        <xsl:element name="assign">
          <xsl:attribute name="name">auditAssign_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <!-- activity name -->
          <xsl:element name="copy">
            <xsl:element name="from">
              <xsl:attribute name="expression">'<xsl:value-of
select="@name"/>'</xsl:attribute>
            </xsl:element>
            <xsl:element name="to">
              <xsl:attribute name="variable">auditSwitchIn</xsl:attribute>
              <xsl:attribute name="part">parameters</xsl:attribute>
              <xsl:attribute
name="query">/audit:auditSwitch/audit:name</xsl:attribute>
            </xsl:element>
          </xsl:element>
        </xsl:element>
        <!-- call auditingservice -->
        <xsl:element name="invoke">
          <xsl:attribute name="name">auditSwitch_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
          <xsl:attribute name="portType">audit:ServiceSoap</xsl:attribute>
          <xsl:attribute name="operation">auditSwitch</xsl:attribute>
          <xsl:attribute name="inputVariable">auditSwitchIn</xsl:attribute>
          <xsl:attribute
name="outputVariable">auditSwitchOut</xsl:attribute>
          <xsl:if test="bpel:correlations">
            <xsl:element name="correlations">
              <xsl:for-each select="bpel:correlations/bpel:correlation">
                <xsl:element name="correlation">
                  <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                  <xsl:attribute name="initiate">no</xsl:attribute>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:element>
        <!-- keep process related invoke activity and extend CASE and
OTHERWISE branches-->
        <xsl:copy>
          <xsl:apply-templates select="@* | node()"/>
        </xsl:copy>
        <!-- POSTactivity auditing - see CASE and OTHERWISE templates-->
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <xsl:template match="bpel:case">
    <!-- POSTactivity auditing -->
```

```
    <!-- assign values to invoke variable -->
    <xsl:element name="case">
      <xsl:attribute name="condition"><xsl:value-of
select="@condition"/></xsl:attribute>
      <xsl:element name="scope">
        <xsl:attribute name="name">auditSwitchCase_<xsl:value-of
select="@name"/></xsl:attribute>
        <xsl:element name="variables">
          <xsl:element name="variable">
            <xsl:attribute name="name">auditSwitchCaseIn</xsl:attribute>
            <xsl:attribute
name="messageType">audit:auditSwitchCaseSoapIn</xsl:attribute>
          </xsl:element>
          <xsl:element name="variable">
            <xsl:attribute name="name">auditSwitchCaseOut</xsl:attribute>
            <xsl:attribute
name="messageType">audit:auditSwitchCaseSoapOut</xsl:attribute>
          </xsl:element>
        </xsl:element>
        <xsl:element name="sequence">
          <xsl:element name="assign">
            <xsl:attribute name="name">auditAssign_POST_<xsl:value-of
select="../@name"/></xsl:attribute>
            <!-- activity name -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="../@name"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditSwitchCaseIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditSwitchCase/audit:name</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <!-- condition -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="@condition"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditSwitchCaseIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditSwitchCase/audit:condition</xsl:attribute>
              </xsl:element>
            </xsl:element>
          </xsl:element>
          <!-- call auditingservice -->
          <xsl:element name="invoke">
            <xsl:attribute name="name">auditSwitchCase_POST_<xsl:value-of
select="@name"/></xsl:attribute>
            <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
            <xsl:attribute
name="portType">audit:ServiceSoap</xsl:attribute>
            <xsl:attribute name="operation">auditSwitchCase</xsl:attribute>
            <xsl:attribute
name="inputVariable">auditSwitchCaseIn</xsl:attribute>
```

```
            <xsl:attribute
name="outputVariable">auditSwitchCaseOut</xsl:attribute>
            <xsl:if test="bpel:correlations">
              <xsl:element name="correlations">
                <xsl:for-each select="bpel:correlations/bpel:correlation">
                  <xsl:element name="correlation">
                    <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                    <xsl:attribute name="initiate">no</xsl:attribute>
                  </xsl:element>
                </xsl:for-each>
              </xsl:element>
            </xsl:if>
          </xsl:element>
          <!-- keep process related invoke activity and extend CASE and
OTHERWISE branches-->
          <xsl:apply-templates select="@* | node()"/>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <xsl:template match="bpel:otherwise">
    <!-- POSTactivity auditing -->
    <!-- assign values to invoke variable -->
    <xsl:element name="otherwise">
      <xsl:element name="scope">
        <xsl:attribute name="name">auditSwitchOtherwise_<xsl:value-of
select="@name"/></xsl:attribute>
        <xsl:element name="variables">
          <xsl:element name="variable">
            <xsl:attribute
name="name">auditSwitchOtherwiseIn</xsl:attribute>
            <xsl:attribute
name="messageType">audit:auditSwitchOtherwiseSoapIn</xsl:attribute>
          </xsl:element>
          <xsl:element name="variable">
            <xsl:attribute
name="name">auditSwitchOtherwiseOut</xsl:attribute>
            <xsl:attribute
name="messageType">audit:auditSwitchOtherwiseSoapOut</xsl:attribute>
          </xsl:element>
        </xsl:element>
        <xsl:element name="sequence">
          <xsl:element name="assign">
            <xsl:attribute name="name">auditAssign_POST_<xsl:value-of
select="../@name"/></xsl:attribute>
            <!-- activity name -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="../@name"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditSwitchOtherwiseIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditSwitchOtherwise/audit:name</xsl:attribute>
              </xsl:element>
            </xsl:element>
          </xsl:element>
          <!-- call auditingservice -->
          <xsl:element name="invoke">
```

```
            <xsl:attribute
name="name">auditSwitchOtherwise_POST_<xsl:value-of
select="@name"/></xsl:attribute>
            <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
            <xsl:attribute
name="portType">audit:ServiceSoap</xsl:attribute>
            <xsl:attribute
name="operation">auditSwitchOtherwise</xsl:attribute>
            <xsl:attribute
name="inputVariable">auditSwitchOtherwiseIn</xsl:attribute>
            <xsl:attribute
name="outputVariable">auditSwitchOtherwiseOut</xsl:attribute>
            <xsl:if test="bpel:correlations">
              <xsl:element name="correlations">
                <xsl:for-each select="bpel:correlations/bpel:correlation">
                  <xsl:element name="correlation">
                    <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                    <xsl:attribute name="initiate">no</xsl:attribute>
                  </xsl:element>
                </xsl:for-each>
              </xsl:element>
            </xsl:if>
          </xsl:element>
          <!-- keep process related activities -->
          <xsl:apply-templates select="@* | node()"/>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <!-- ****** END AUDIT SWITCH ****** -->
  <!-- ****** AUDIT PICK ****** -->
  <xsl:template match="bpel:pick">
    <xsl:choose>
      <xsl:when test="@createInstance and (@createInstance='yes')">
        <!-- keep process related invoke activity unchanged -->
        <xsl:copy>
          <xsl:apply-templates select="@* | node()"/>
        </xsl:copy>
      </xsl:when>
      <xsl:otherwise>
        <xsl:element name="scope">
          <xsl:attribute name="name">auditPick_<xsl:value-of
select="@name"/></xsl:attribute>
          <xsl:element name="variables">
            <xsl:element name="variable">
              <xsl:attribute name="name">auditPickIn</xsl:attribute>
              <xsl:attribute
name="messageType">audit:auditPickSoapIn</xsl:attribute>
            </xsl:element>
            <xsl:element name="variable">
              <xsl:attribute name="name">auditPickOut</xsl:attribute>
              <xsl:attribute
name="messageType">audit:auditPickSoapOut</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <xsl:element name="sequence">
            <!-- PREactivity auditing -->
            <!-- assign values to invoke variable -->
            <xsl:element name="assign">
              <xsl:attribute name="name">auditAssign_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
```

```
              <!-- activity name -->
              <xsl:element name="copy">
                <xsl:element name="from">
                  <xsl:attribute name="expression">'<xsl:value-of
select="@name"/>'</xsl:attribute>
                </xsl:element>
                <xsl:element name="to">
                  <xsl:attribute
name="variable">auditPickIn</xsl:attribute>
                  <xsl:attribute name="part">parameters</xsl:attribute>
                  <xsl:attribute
name="query">/audit:auditPick/audit:name</xsl:attribute>
                </xsl:element>
              </xsl:element>
            </xsl:element>
            <!-- call auditingservice -->
            <xsl:element name="invoke">
              <xsl:attribute name="name">auditPick_PRE_<xsl:value-of
select="@name"/></xsl:attribute>
              <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
              <xsl:attribute
name="portType">audit:ServiceSoap</xsl:attribute>
              <xsl:attribute name="operation">auditPick</xsl:attribute>
              <xsl:attribute
name="inputVariable">auditPickIn</xsl:attribute>
              <xsl:attribute
name="outputVariable">auditPickOut</xsl:attribute>
              <xsl:if test="bpel:correlations">
                <xsl:element name="correlations">
                  <xsl:for-each
select="bpel:correlations/bpel:correlation">
                    <xsl:element name="correlation">
                      <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                      <xsl:attribute name="initiate">no</xsl:attribute>
                    </xsl:element>
                  </xsl:for-each>
                </xsl:element>
              </xsl:if>
            </xsl:element>
            <!-- keep process related invoke activity unchanged -->
            <xsl:copy>
              <xsl:apply-templates select="@* | node()"/>
            </xsl:copy>
            <!-- POSTactivity auditing - see ONMESSAGE and ONALARM
templates-->
          </xsl:element>
        </xsl:element>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- POSTactivity auditing for ONMESSAGE-->
  <xsl:template match="bpel:onMessage">
    <xsl:element name="onMessage">
      <xsl:attribute name="partnerLink"><xsl:value-of
select="@partnerLink"/></xsl:attribute>
      <xsl:attribute name="portType"><xsl:value-of
select="@portType"/></xsl:attribute>
      <xsl:attribute name="operation"><xsl:value-of
select="@operation"/></xsl:attribute>
      <xsl:if test="@variable">
```

```
        <xsl:attribute name="variable"><xsl:value-of
select="@variable"/></xsl:attribute>
      </xsl:if>
      <xsl:element name="scope">
        <xsl:attribute name="name">auditPickOnMessage_<xsl:value-of
select="../@name"/></xsl:attribute>
        <xsl:element name="variables">
          <xsl:element name="variable">
            <xsl:attribute name="name">auditPickOnMessageIn</xsl:attribute>
            <xsl:attribute
name="messageType">audit:auditPickOnMessageSoapIn</xsl:attribute>
          </xsl:element>
          <xsl:element name="variable">
            <xsl:attribute
name="name">auditPickOnMessageOut</xsl:attribute>
            <xsl:attribute
name="messageType">audit:auditPickOnMessageSoapOut</xsl:attribute>
          </xsl:element>
          <xsl:element name="variable">
            <xsl:attribute name="name">auditAny</xsl:attribute>
            <xsl:attribute name="element">audit:auditAny</xsl:attribute>
          </xsl:element>
        </xsl:element>
        <xsl:element name="sequence">
          <!-- assign values to invoke variable -->
          <xsl:element name="assign">
            <xsl:attribute name="name">auditAssign_POST_<xsl:value-of
select="../@name"/></xsl:attribute>
            <!-- activity name -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="../@name"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditPickOnMessageIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditPickOnMessage/audit:name</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <!-- partnerlink -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="@partnerLink"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditPickOnMessageIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditPickOnMessage/audit:partnerLink</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <!-- porttype -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="@portType"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
```

```
                <xsl:attribute
name="variable">auditPickOnMessageIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditPickOnMessage/audit:portType</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <!-- operation -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="@operation"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditPickOnMessageIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditPickOnMessage/audit:operation</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <!-- variablename -->
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="@variable"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditPickOnMessageIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditPickOnMessage/audit:variableName</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <!-- variablevalue - may be missing -->
            <xsl:if test="@variable">
              <xsl:element name="copy">
                <xsl:element name="from">
                  <xsl:attribute name="variable"><xsl:value-of
select="@variable"/></xsl:attribute>
                </xsl:element>
                <xsl:element name="to">
                  <xsl:attribute name="variable">auditAny</xsl:attribute>
                </xsl:element>
              </xsl:element>
              <xsl:element name="copy">
                <xsl:element name="from">
                  <xsl:attribute name="variable">auditAny</xsl:attribute>
                </xsl:element>
                <xsl:element name="to">
                  <xsl:attribute
name="variable">auditPickOnMessageIn</xsl:attribute>
                  <xsl:attribute name="part">parameters</xsl:attribute>
                  <xsl:attribute
name="query">/audit:auditPickOnMessage/audit:variableValue</xsl:attribute>
                </xsl:element>
              </xsl:element>
            </xsl:if>
          </xsl:element>
          <!-- call auditingservice -->
          <xsl:element name="invoke">
```

```
            <xsl:attribute name="name">auditPickOnMessage_POST_<xsl:value-
of select="../@name"/></xsl:attribute>
            <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
            <xsl:attribute
name="portType">audit:ServiceSoap</xsl:attribute>
            <xsl:attribute
name="operation">auditPickOnMessage</xsl:attribute>
            <xsl:attribute
name="inputVariable">auditPickOnMessageIn</xsl:attribute>
            <xsl:attribute
name="outputVariable">auditPickOnMessageOut</xsl:attribute>
            <xsl:if test="bpel:correlations">
              <xsl:element name="correlations">
                <xsl:for-each select="bpel:correlations/bpel:correlation">
                  <xsl:element name="correlation">
                    <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                    <xsl:attribute name="initiate">no</xsl:attribute>
                  </xsl:element>
                </xsl:for-each>
              </xsl:element>
            </xsl:if>
          </xsl:element>
          <!-- keep process related invoke activity and extend CASE and
OTHERWISE branches-->
          <xsl:apply-templates select="@* | node()"/>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <!-- POSTactivity auditing for ONALARM-->
  <xsl:template match="bpel:onAlarm">
    <xsl:element name="onAlarm">
      <xsl:if test="@for">
        <xsl:attribute name="for"><xsl:value-of
select="@for"/></xsl:attribute>
      </xsl:if>
      <xsl:if test="@until">
        <xsl:attribute name="until"><xsl:value-of
select="@until"/></xsl:attribute>
      </xsl:if>
      <xsl:element name="scope">
        <xsl:attribute name="name">auditPickOnAlarm_<xsl:value-of
select="../@name"/></xsl:attribute>
        <xsl:element name="variables">
          <xsl:element name="variable">
            <xsl:attribute name="name">auditPickOnAlarmIn</xsl:attribute>
            <xsl:attribute
name="messageType">audit:auditPickOnAlarmSoapIn</xsl:attribute>
          </xsl:element>
          <xsl:element name="variable">
            <xsl:attribute name="name">auditPickOnAlarmOut</xsl:attribute>
            <xsl:attribute
name="messageType">audit:auditPickOnAlarmSoapOut</xsl:attribute>
          </xsl:element>
        </xsl:element>
        <xsl:element name="sequence">
          <!-- assign values to invoke variable -->
          <xsl:element name="assign">
            <xsl:attribute name="name">auditAssign_POST_<xsl:value-of
select="../@name"/></xsl:attribute>
            <!-- activity name -->
```

```
            <xsl:element name="copy">
              <xsl:element name="from">
                <xsl:attribute name="expression">'<xsl:value-of
select="../@name"/>'</xsl:attribute>
              </xsl:element>
              <xsl:element name="to">
                <xsl:attribute
name="variable">auditPickOnAlarmIn</xsl:attribute>
                <xsl:attribute name="part">parameters</xsl:attribute>
                <xsl:attribute
name="query">/audit:auditPickOnAlarm/audit:name</xsl:attribute>
              </xsl:element>
            </xsl:element>
            <!-- for -->
            <xsl:if test="@for">
              <xsl:element name="copy">
                <xsl:element name="from">
                  <xsl:attribute name="expression">'<xsl:value-of
select="@for"/>'</xsl:attribute>
                </xsl:element>
                <xsl:element name="to">
                  <xsl:attribute
name="variable">auditPickOnAlarmIn</xsl:attribute>
                  <xsl:attribute name="part">parameters</xsl:attribute>
                  <xsl:attribute
name="query">/audit:auditPickOnAlarm/audit:forDuration</xsl:attribute>
                </xsl:element>
              </xsl:element>
            </xsl:if>
            <!-- until -->
            <xsl:if test="@until">
              <xsl:element name="copy">
                <xsl:element name="from">
                  <xsl:attribute name="expression">'<xsl:value-of
select="@until"/>'</xsl:attribute>
                </xsl:element>
                <xsl:element name="to">
                  <xsl:attribute
name="variable">auditPickOnAlarmIn</xsl:attribute>
                  <xsl:attribute name="part">parameters</xsl:attribute>
                  <xsl:attribute
name="query">/audit:auditPickOnAlarm/audit:untilDuration</xsl:attribute>
                </xsl:element>
              </xsl:element>
            </xsl:if>
          </xsl:element>
          <!-- call auditingservice -->
          <xsl:element name="invoke">
            <xsl:attribute name="name">auditPickOnAlarm_POST_<xsl:value-of
select="../@name"/></xsl:attribute>
            <xsl:attribute
name="partnerLink">auditingWebservice</xsl:attribute>
            <xsl:attribute
name="portType">audit:ServiceSoap</xsl:attribute>
            <xsl:attribute
name="operation">auditPickOnAlarm</xsl:attribute>
            <xsl:attribute
name="inputVariable">auditPickOnAlarmIn</xsl:attribute>
            <xsl:attribute
name="outputVariable">auditPickOnAlarmOut</xsl:attribute>
            <xsl:if test="bpel:correlations">
              <xsl:element name="correlations">
                <xsl:for-each select="bpel:correlations/bpel:correlation">
```

```
                  <xsl:element name="correlation">
                    <xsl:attribute name="set"><xsl:value-of
select="@set"/></xsl:attribute>
                    <xsl:attribute name="initiate">no</xsl:attribute>
                  </xsl:element>
                </xsl:for-each>
              </xsl:element>
            </xsl:if>
          </xsl:element>
          <!-- keep process related invoke activity and extend CASE and
OTHERWISE branches-->
          <xsl:apply-templates select="@* | node()"/>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  <!-- ****** END AUDIT PICK ****** -->
</xsl:stylesheet>
```

8.4 auditing web service: WSDL Service Description

```
<wsdl:definitions
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://hro.at/bpel/auditing/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://hro.at/bpel/auditing/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://hro.at/bpel/auditing/">
      <s:element name="auditReceive">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="partnerLink"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="portType"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="operation"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="variableName"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="correlations">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
            <s:element minOccurs="0" maxOccurs="1" name="variableValue">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
```

```
              </s:sequence>
              <s:anyAttribute/>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="auditReceiveResponse">
      <s:complexType/>
    </s:element>
    <s:element name="auditReply">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="partnerLink"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="portType"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="operation"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="variableName"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="faultName"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="correlations">
            <s:complexType mixed="true">
              <s:sequence>
                <s:any maxOccurs="unbounded"/>
              </s:sequence>
              <s:anyAttribute/>
            </s:complexType>
          </s:element>
          <s:element minOccurs="0" maxOccurs="1" name="variableValue">
            <s:complexType mixed="true">
              <s:sequence>
                <s:any maxOccurs="unbounded"/>
              </s:sequence>
              <s:anyAttribute/>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="auditReplyResponse">
      <s:complexType/>
    </s:element>
    <s:element name="auditInvoke">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="partnerLink"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="portType"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="operation"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="inputVariableName"
type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1"
name="outputVariableName" type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1" name="correlations">
```

```
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
            <s:element minOccurs="0" maxOccurs="1"
name="inputVariableValue">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
            <s:element minOccurs="0" maxOccurs="1"
name="outputVariableValue">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="auditInvokeResponse">
        <s:complexType/>
      </s:element>
      <s:element name="auditThrow">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="faultName"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="faultVariableName"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="correlations">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
            <s:element minOccurs="0" maxOccurs="1"
name="faultVariableValue">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="auditThrowResponse">
        <s:complexType/>
      </s:element>
```

```
      <s:element name="auditSwitch">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="correlations">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="auditSwitchResponse">
        <s:complexType/>
      </s:element>
      <s:element name="auditSwitchCase">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="condition"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="correlations">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="auditSwitchCaseResponse">
        <s:complexType/>
      </s:element>
      <s:element name="auditSwitchOtherwise">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="correlations">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="auditSwitchOtherwiseResponse">
        <s:complexType/>
      </s:element>
      <s:element name="auditPick">
        <s:complexType>
          <s:sequence>
```

```
            <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="correlations">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="auditPickResponse">
        <s:complexType/>
      </s:element>
      <s:element name="auditPickOnMessage">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="partnerLink"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="portType"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="operation"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="variableName"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="correlations">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
            <s:element minOccurs="0" maxOccurs="1" name="variableValue">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any maxOccurs="unbounded"/>
                </s:sequence>
                <s:anyAttribute/>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="auditPickOnMessageResponse">
        <s:complexType/>
      </s:element>
      <s:element name="auditPickOnAlarm">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="name"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="forDuration"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="untilDuration"
type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="correlations">
              <s:complexType mixed="true">
                <s:sequence>
```

```
                <s:any maxOccurs="unbounded"/>
              </s:sequence>
              <s:anyAttribute/>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="auditPickOnAlarmResponse">
      <s:complexType/>
    </s:element>
    <s:element name="auditAny">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="anyValue">
            <s:complexType mixed="true">
              <s:sequence>
                <s:any maxOccurs="unbounded"/>
              </s:sequence>
              <s:anyAttribute/>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="auditAnyResponse">
      <s:complexType/>
    </s:element>
  </s:schema>
</wsdl:types>
<wsdl:message name="auditReceiveSoapIn">
  <wsdl:part name="parameters" element="tns:auditReceive"/>
</wsdl:message>
<wsdl:message name="auditReceiveSoapOut">
  <wsdl:part name="parameters" element="tns:auditReceiveResponse"/>
</wsdl:message>
<wsdl:message name="auditReplySoapIn">
  <wsdl:part name="parameters" element="tns:auditReply"/>
</wsdl:message>
<wsdl:message name="auditReplySoapOut">
  <wsdl:part name="parameters" element="tns:auditReplyResponse"/>
</wsdl:message>
<wsdl:message name="auditInvokeSoapIn">
  <wsdl:part name="parameters" element="tns:auditInvoke"/>
</wsdl:message>
<wsdl:message name="auditInvokeSoapOut">
  <wsdl:part name="parameters" element="tns:auditInvokeResponse"/>
</wsdl:message>
<wsdl:message name="auditThrowSoapIn">
  <wsdl:part name="parameters" element="tns:auditThrow"/>
</wsdl:message>
<wsdl:message name="auditThrowSoapOut">
  <wsdl:part name="parameters" element="tns:auditThrowResponse"/>
</wsdl:message>
<wsdl:message name="auditSwitchSoapIn">
  <wsdl:part name="parameters" element="tns:auditSwitch"/>
</wsdl:message>
<wsdl:message name="auditSwitchSoapOut">
  <wsdl:part name="parameters" element="tns:auditSwitchResponse"/>
</wsdl:message>
<wsdl:message name="auditSwitchCaseSoapIn">
  <wsdl:part name="parameters" element="tns:auditSwitchCase"/>
</wsdl:message>
```

```
  <wsdl:message name="auditSwitchCaseSoapOut">
    <wsdl:part name="parameters" element="tns:auditSwitchCaseResponse"/>
  </wsdl:message>
  <wsdl:message name="auditSwitchOtherwiseSoapIn">
    <wsdl:part name="parameters" element="tns:auditSwitchOtherwise"/>
  </wsdl:message>
  <wsdl:message name="auditSwitchOtherwiseSoapOut">
    <wsdl:part name="parameters"
element="tns:auditSwitchOtherwiseResponse"/>
  </wsdl:message>
  <wsdl:message name="auditPickSoapIn">
    <wsdl:part name="parameters" element="tns:auditPick"/>
  </wsdl:message>
  <wsdl:message name="auditPickSoapOut">
    <wsdl:part name="parameters" element="tns:auditPickResponse"/>
  </wsdl:message>
  <wsdl:message name="auditPickOnMessageSoapIn">
    <wsdl:part name="parameters" element="tns:auditPickOnMessage"/>
  </wsdl:message>
  <wsdl:message name="auditPickOnMessageSoapOut">
    <wsdl:part name="parameters" element="tns:auditPickOnMessageResponse"/>
  </wsdl:message>
  <wsdl:message name="auditPickOnAlarmSoapIn">
    <wsdl:part name="parameters" element="tns:auditPickOnAlarm"/>
  </wsdl:message>
  <wsdl:message name="auditPickOnAlarmSoapOut">
    <wsdl:part name="parameters" element="tns:auditPickOnAlarmResponse"/>
  </wsdl:message>
  <wsdl:message name="auditAnySoapIn">
    <wsdl:part name="parameters" element="tns:auditAny"/>
  </wsdl:message>
  <wsdl:message name="auditAnySoapOut">
    <wsdl:part name="parameters" element="tns:auditAnyResponse"/>
  </wsdl:message>
  <wsdl:portType name="ServiceSoap">
    <wsdl:operation name="auditReceive">
      <wsdl:input message="tns:auditReceiveSoapIn"/>
      <wsdl:output message="tns:auditReceiveSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditReply">
      <wsdl:input message="tns:auditReplySoapIn"/>
      <wsdl:output message="tns:auditReplySoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditInvoke">
      <wsdl:input message="tns:auditInvokeSoapIn"/>
      <wsdl:output message="tns:auditInvokeSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditThrow">
      <wsdl:input message="tns:auditThrowSoapIn"/>
      <wsdl:output message="tns:auditThrowSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditSwitch">
      <wsdl:input message="tns:auditSwitchSoapIn"/>
      <wsdl:output message="tns:auditSwitchSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditSwitchCase">
      <wsdl:input message="tns:auditSwitchCaseSoapIn"/>
      <wsdl:output message="tns:auditSwitchCaseSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditSwitchOtherwise">
      <wsdl:input message="tns:auditSwitchOtherwiseSoapIn"/>
      <wsdl:output message="tns:auditSwitchOtherwiseSoapOut"/>
    </wsdl:operation>
```

```
    <wsdl:operation name="auditPick">
      <wsdl:input message="tns:auditPickSoapIn"/>
      <wsdl:output message="tns:auditPickSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditPickOnMessage">
      <wsdl:input message="tns:auditPickOnMessageSoapIn"/>
      <wsdl:output message="tns:auditPickOnMessageSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditPickOnAlarm">
      <wsdl:input message="tns:auditPickOnAlarmSoapIn"/>
      <wsdl:output message="tns:auditPickOnAlarmSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="auditAny">
      <wsdl:input message="tns:auditAnySoapIn"/>
      <wsdl:output message="tns:auditAnySoapOut"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="auditReceive">
      <soap:operation soapAction="http://hro.at/bpel/auditing/auditReceive"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditReply">
      <soap:operation soapAction="http://hro.at/bpel/auditing/auditReply"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditInvoke">
      <soap:operation soapAction="http://hro.at/bpel/auditing/auditInvoke"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditThrow">
      <soap:operation soapAction="http://hro.at/bpel/auditing/auditThrow"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditSwitch">
      <soap:operation soapAction="http://hro.at/bpel/auditing/auditSwitch"
style="document"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditSwitchCase">
      <soap:operation
soapAction="http://hro.at/bpel/auditing/auditSwitchCase" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditSwitchOtherwise">
      <soap:operation
soapAction="http://hro.at/bpel/auditing/auditSwitchOtherwise"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditPick">
      <soap:operation soapAction="http://hro.at/bpel/auditing/auditPick"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditPickOnMessage">
      <soap:operation
soapAction="http://hro.at/bpel/auditing/auditPickOnMessage"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditPickOnAlarm">
      <soap:operation
soapAction="http://hro.at/bpel/auditing/auditPickOnAlarm"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditAny">
      <soap:operation soapAction="http://hro.at/bpel/auditing/auditAny"
style="document"/>
      <wsdl:input>
```

```
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="auditReceive">
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="auditReply">
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="auditInvoke">
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="auditThrow">
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="auditSwitch">
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="auditSwitchCase">
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="auditSwitchOtherwise">
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
```

```
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditPick">
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditPickOnMessage">
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditPickOnAlarm">
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="auditAny">
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Service">
    <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
      <soap:address
location="http://localhost/auditingWebservice/Service.asmx"/>
    </wsdl:port>
    <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
      <soap12::address
location="http://localhost/auditingWebservice/Service.asmx"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

8.5 XML Schema for Activity Audit Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
      <xs:element name="activityaudit">
            <xs:complexType>
                  <xs:choice>
                        <xs:element name="invokeaudit"
type="invokeaudittype"/>
                        <xs:element name="replyaudit"
type="replyaudittype"/>
                        <xs:element name="receiveaudit"
type="receiveaudittype"/>
                        <xs:element name="pickaudit" type="pickaudittype"/>
```

```
                        <xs:element name="pickonalarmaudit"
type="pickonalarmaudittype"/>
                        <xs:element name="pickonmessageaudit"
type="pickonmessageaudittype"/>
                        <xs:element name="throwaudit"
type="throwaudittype"/>
                        <xs:element name="switchaudit"
type="switchaudittype"/>
                        <xs:element name="switchcaseaudit"
type="switchcaseaudittype"/>
                        <xs:element name="switchotherwiseaudit"
type="switchotherwiseaudittype"/>
                  </xs:choice>
            </xs:complexType>
      </xs:element>
      <xs:complexType name="activitytype">
            <xs:sequence>
                  <xs:element name="name" type="xs:string"/>
                  <xs:element name="correlations" type="whatever"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="receiveaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype">
                        <xs:sequence>
                              <xs:element name="partnerLink"
type="xs:string"/>
                              <xs:element name="portType" type="xs:string"/>
                              <xs:element name="operation"
type="xs:string"/>
                              <xs:element name="variableName"
type="xs:string"/>
                              <xs:element name="variableValue"
type="whatever"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="replyaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype">
                        <xs:sequence>
                              <xs:element name="partnerLink"
type="xs:string"/>
                              <xs:element name="portType" type="xs:string"/>
                              <xs:element name="operation"
type="xs:string"/>
                              <xs:element name="variableName"
type="xs:string"/>
                              <xs:element name="variableValue"
type="whatever"/>
                              <xs:element name="faultName"
type="xs:string"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="invokeaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype">
                        <xs:sequence>
                              <xs:element name="partnerLink"
type="xs:string"/>
```

```
                              <xs:element name="portType" type="xs:string"/>
                              <xs:element name="operation"
type="xs:string"/>
                              <xs:element name="inputVariableName"
type="xs:string"/>
                              <xs:element name="inputVariableValue"
type="whatever"/>
                              <xs:element name="outputVariableName"
type="xs:string"/>
                              <xs:element name="outputVariableValue"
type="whatever"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="pickaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype">
                        <xs:sequence>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="pickonmessageaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype">
                        <xs:sequence>
                              <xs:element name="partnerLink"
type="xs:string"/>
                              <xs:element name="portType" type="xs:string"/>
                              <xs:element name="operation"
type="xs:string"/>
                              <xs:element name="variableName"
type="xs:string"/>
                              <xs:element name="variableValue"
type="whatever"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="pickonalarmaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype">
                        <xs:sequence>
                              <xs:element name="duration" type="xs:string"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="throwaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype">
                        <xs:sequence>
                              <xs:element name="faultName"
type="xs:string"/>
                              <xs:element name="faultVariableName"
type="xs:string"/>
                              <xs:element name="faultVariableValue"
type="whatever"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
```

```
      <xs:complexType name="switchaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype"/>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="switchcaseaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype">
                        <xs:sequence>
                              <xs:element name="condition"
type="xs:string"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="switchotherwiseaudittype">
            <xs:complexContent>
                  <xs:extension base="activitytype"/>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="whatever">
            <xs:sequence>
                  <xs:any processContents="skip" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:schema>
```