Auditable WSBPEL: probing and monitoring of business processes with web services

Josef Schiefer*

Institute for Software Technology and Interactive Systems,
Vienna University of Technology,
Favoritenstrasse 9-11/188,
Vienna 1040, Austria
E-mail: js@ifs.tuwien.ac.at
*Corresponding author

Heinz Roth

Secure Business Austria,
Favoritenstrasse 16, Vienna 1040, Austria
E-mail: roth@securityresearch.at

Alexander Schatten

Institute for Software Technology and Interactive Systems,
Vienna University of Technology,
Favoritenstrasse 9-11/188, Vienna 1040, Austria
E-mail: aschatt@ifs.tuwien.ac.at

Abstract: Today's business climate requires organisations to constantly evolve IT strategies to respond to new opportunities or threats. Tracking the achievement of business goals, objectives and strategies is increasingly used to measure and adjust the outcome of business processes. In this paper, we introduce a web service-based approach for probing WSBPEL processes. With our approach, organisations are able to automatically extend existing WSBPEL processes with auditing extensions which capture audit information during process execution time. We show how to transform a WSBPEL model into an auditable model in order to seamlessly integrate audit trails on different process engines for monitoring purposes. Based on our experience building an auditable process model, we discuss various ways for extending WSBPEL for auditing purposes.

Keywords: business process management; monitoring; controlling; web services; workflow; business process execution language for web services; BPEL4WS.

Reference to this paper should be made as follows: Schiefer, J., Roth, H. and Schatten, A. (XXXX) 'Auditable WSBPEL: probing and monitoring of business processes with web services', *Int. J. Business Process Integration and Management*, Vol. X, No. Y, pp.XXX–XXX.

Biographical notes: Josef Schiefer received a PhD in Information Systems from the University of Vienna and was a Research Assistant in the Department of Software Technology at the Vienna University of Technology. He joined IBM's Thomas J. Watson Research Center where he continued his research in the areas of business process management, business process intelligence, workflow management and data warehousing. Today, he is Technical Director at Senactive Inc., a company offering software for real-time sense and respond solutions.

Heinz Roth received a Master in Computer Science and is currently pursuing a PhD in Information Systems from the Vienna University of Technology. Currently, he is working on Research Projects in the areas of event processing systems and data mining applications.

Alexander Schatten received a Master in Analytical Chemistry and his PhD in Information Systems from the Vienna University of Technology. He works in the open source community and is head of the open source workgroup of the Austrian Computer Society. His research areas are software engineering, event-driven architectures and groupware. Currently, he is a Researcher at the Institute for Software Technology and Interactive Systems at the Vienna University of Technology and is working as a consultant and freelance journalist for various publications.

1 Introduction

Business Process Management Systems (BPMSs) are software solutions that support the management of business process lifecycles. This includes the definition, execution and monitoring of business processes. For the execution of business processes, many organisations are increasingly using process engines supporting standard-based process models (such as WSBPEL) to improve the efficiency of their processes.

A major challenge of current BPMS solutions is to close the gap between executing a business process and monitoring it. As stated by Sayal et al. (2002), this combination, among other things, allows business users to be notified or even react in real-time in case predefined conditions are met. Furthermore, many types of events may be recorded during the execution of the business process, including the start and completion time of each activity, its input and output data, the assigned resources and the outcome of the execution. This information can be used for analysis purposes in order to 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 to notify users and processes of quality degradations.

Plenty of WSBPEL engines currently exist – commercial and open source ones – which offer their own auditing features. Unfortunately, most of these proprietary auditing features are not designed for interoperability with other WSBPEL engines, reducing their value as a monitoring instrument for interorganisational processes where heterogeneous systems are most likely to be found.

One reason for the limited capabilities is the lack of a broadly supported (industry) standard for audit trail information, which is implemented differently by major BPMSs. Although there is a standard specification for workflow audit trails in the reference model of the Workflow Management Coalition (1998), it is not supported by most workflow management products. Therefore, up until recently, it has been very difficult for process analysts to capture and use process audit information to get a clear picture regarding the status and performance of business processes.

In this paper, we propose a new approach to monitor WSBPEL processes which overcomes the aforementioned problems. The basic idea behind our approach is to extend a WSBPEL process definition with auditing activities in order to report relevant state changes to an auditing web service. The resulting auditable process definition does not use proprietary elements but remains compliant with the WSBPEL standard. Therefore, it is possible to enhance process definitions running on different WSBPEL engines with the proposed auditing feature and centralise the collected information.

The remainder of this paper is organised as follows. In Section 2, we reflect on related work on auditing and monitoring business processes. In Sections 3–6, we propose a new approach for transforming an existing WSBPEL process description into an auditable version. In Section 7, we demonstrate our approach step-by-step with a tool that we developed based on a concrete implemented example. In Section 8, we discuss an approach for extending WSBPEL for auditing purposes and finally, in Section 9, we conclude our paper and provide an outlook for future work.

2 Related work

For the execution and monitoring of business processes, many organisations are increasingly using BPMSs to improve the efficiency of their processes and reduce costs. During the execution of the business process, workflow management systems record many types of events, such as the start and completion time of each activity, the assigned resources and the outcome of the execution. Major BPMSs lack capabilities for transforming, accumulating and condensing audit trails of distributed business processes and using this information for monitoring and analysis purposes to provide feedback about the performance of business processes (in particular interorganisational business processes).

Leymann and Roller (2000) point out that audit trail data from automated business processes can quickly increase to a sizable amount. If several organisations are involved, for example, if processes across an integrated supply chain are to be supervised, the volume of information can quickly reach levels that negatively impact the operational performance of enterprise applications. One way to handle data intensive operations is the outsourcing of data processing to external parties. The outsourcing of process management services allows for the sharing of process information across multiple organisations. The positive economic effect of shared information in an integrated supply chain has been demonstrated in a number of studies (Gavirneni et al., 1999; Holton et al., 2002).

Jeng et al. (2003) have developed an agent-based architecture with the aim of providing continuous, real-time analytics for business processes. For the analytical processing they introduced an agent framework that is able to detect situations and exceptions in a business environment, perform complex analytical tasks and reflect on the gap between current situations and desired management goals. McGregor and Kumaran (2002) have presented a solution management framework that analyses workflow audit logs, utilising decision support system principles and agent technologies to feedback performance measures. This framework forms part of the Intelligent Workflow Monitoring System (IW-MONS) meta-methodology. An extension for this framework using web services was proposed by McGregor and Schiefer (2003), who state that current web service frameworks do not include the functionality required for web service execution performance measurement from an organisational perspective. Part of this work is the extension of the Business Process Execution Language for Web Services (BPEL4WS) (2003) with mechanisms to obtain performance information. Hacigümüs (2006) introduce efficient and flexible methods for auditing IT systems by leveraging the middleware platform capabilities. Hacigümüs shows how the service process

flows that have been executed in the system can be audited, both based on their structure and the information that is disclosed by them. Zur Muehlen (2001) has developed a generic architecture for workflow-based Process Information Systems. Based on the flow of management information in an organisation, he developed a cybernetic model of three distinct feedback levels for automated process regulation, operational process management and strategic process management.

Baresi and Guinea (2005) as well as Lazovik et al. (2004) state that correctness and quality of applications in the service-oriented world, which are captured in an increasingly abstract manner, cannot be assessed with the same techniques as other software pieces. Underlying services and their providers, as well as the process structure itself, can be adjusted very easily, which is one of the strengths of the SOA paradigm, but it has the disadvantage of a never-ending testing deficit. Therefore, particularly Baresi and Guinea propose that the validation of these systems has to be shifted to the runtime. Similar to our approach, they extend existing WSBPEL processes to contact a special web service called monitoring. But unlike our proposal, their goal is not to monitor process instances to gain real-time information, which can be used to assist with managerial tasks, but to detect participating services that deliver unexpected results.

3 Auditing WSBPEL processes

In this section, we introduce various strategies for auditing WSBPEL processes. There are five principal options:

1 instrumentation of web service requests of the WSBPEL process on protocol level

2 on the application server level

3 utilising the auditing service of a process engine used for enacting the WSBPEL process

4 using probes in the operational systems that track state changes of the business process

5 including the auditing mechanism as a partner within the WSBPEL process.

All five options have advantages and disadvantages which are discussed in the next sections.

3.1 Interception of web service requests on the protocol level

Using this option, a web service gateway is used to intercept any web service request from the BPMS and to extract the data needed for the auditing. The major advantage of this option is that the web service auditing is fairly straightforward, and many tools are available that allow an interception of web service requests (e.g. IBM Web Service Gateway). However, the web service requests often contain information only on the choreography level for WSBPEL processes and provide very limited visibility of internal process information. For instance, the web service requests do not include details about the process instances or variables used by the process engine to control the process execution. Therefore, with this option only an incomplete audit trail can be extracted which often results in a more complex audit trail processing in order to regain further process information.

3.2 Interception of web service requests on the application server level

Web services are usually executed on application servers, which offer facilities for listening and intercepting client requests. For instance, J2EE and .NET offer event and listener models for capturing audit data from web requests. Developers can take advantage of such facilities and implement their own listener components, which extract relevant information from web service requests. Although this approach is often easier to implement and deploy than using a tool for the interception of web service requests on the protocol level, it has the same deficiencies when it comes to the visibility of internal process information.

3.3 Auditing service of BPMSs

Most BPMSs supporting WSBPEL provide auditing services that record the activities of the process engine. However, the auditing services can vary significantly between BPMSs. Although there already exist defined standards for workflow logs (Workflow Management Coalition, 1998), the major vendors of BPMSs mostly still use proprietary audit trails. However, using the auditing services of BPMSs also has some advantages over the previously discussed options. If WSBPEL processes can be directly instrumented within a process engine, any process runtime data is available for capturing contextual information about a running business process. This may include state information about the process instance or information about the allocated resources for a particular activity. Although this option is the most flexible one for extracting and propagating audit trail data with minimal latency, only few process engines support flexible auditing. Another trade-off for this approach is that it is proprietary to BPMSs, meaning that the interfaces of auditing services can vary significantly.

3.4 Probes in operational systems

In order to track state changes of business processes, probes in the operational systems can also be used. Probes can sense the operations of a source system and also extract and generate audit data from the operational runtime data. They can be implemented at various application layers (e.g. database layer, business-logic layer, presentation layer), but must be exposed to sufficient runtime information about the business process in order to perform the logging activities.

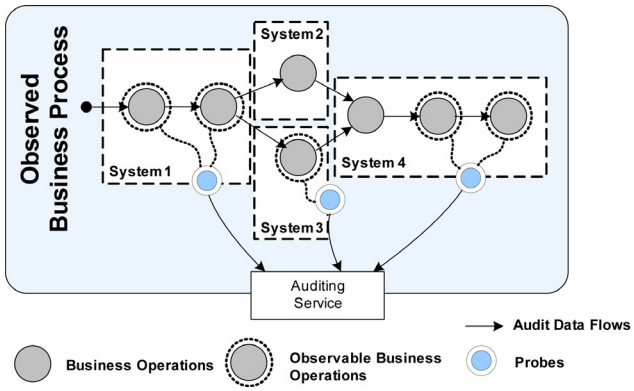
For instance, triggers on database tables can automatically generate audit data each time a table record is inserted. Although the delays for extracting and propagating the audit data are minimal, database triggers have a performance impact on the operational source system and sometimes do not have easy access to all

runtime data of a business process (e.g. data of business objects has to be gathered from multiple tables).

Often the operational source systems offer auditing mechanisms that can be utilised. By using logging daemons that extract information from log files or database tables on a scheduled basis, the required audit data about the business process can be extracted. Due to the scheduled data extraction, this option can cause some latency in the audit trail processing.

Figure 1 shows a business process supported by four operational systems with three of them providing probes. An auditing service collects the logging data from various probes and consolidates it for further use. The biggest challenge with such an approach is the last step. The consolidation and integration of data from probes of various source systems is a major undertaking, since the formats, semantics and quality of the audit data can vary significantly for the source systems.

Figure 1 Operational systems with probes



3.5 *Auditing web service as part of a WSBPEL process*

An *auditing web service* provides the mechanism to gather audit trail information about business processes during execution. The captured audit-trail is a time-sequenced record of all status changes of a business process. WSBPEL processes essentially implement a layer on top of WSDL, with WSDL defining the allowed operations and WSBPEL defining how these operations should be orchestrated. A WSBPEL document leverages WSDL in three ways:

1 Every WSBPEL process is exposed as a web service using WSDL. The WSDL describes the entry points for external services to interact with the process.

2 WSDL data types are used within a WSBPEL process to describe the information that passes between requests.

3 WSDL might be used to reference external services required by the process.

WSBPEL processes specify stateful interactions involving the exchange of messages between partners. The state of a business process includes the messages that are exchanged as well as intermediate data used in business logic and in composing messages sent to partners. Variables provide the means for holding messages that constitute the state of a business process. The messages held are often those that have been received from partners or are to be sent to partners. Variables can also hold data that are needed for keeping state information related to the process that is never exchanged with partners. Variables identify the specific data exchanged in a message flow, which typically maps to a WSDL message type. When a WSBPEL process receives a message, the appropriate variables are populated so that subsequent requests can access the data.

The major advantage of this approach is the seamless standard-based integration of the auditing web service with the business process. The auditing web service can also be shared among business partners and thereby enables audit trail information about WSBPEL processes to be captured centrally, beyond organisational boundaries.

Although this approach enables seamless integration of the auditing services with the business process engine, there are also some drawbacks. Since the BPMSs have to invoke the auditing web service for every state change of a business process, the overhead for these web service requests can become a potential bottleneck. Furthermore, this option requires a high availability of the auditing web service to avoid missing auditing requests when process state changes occur. Nevertheless, these issues can be solved in two ways. Firstly, there is a lot of experience in building scalable and robust web service-based applications as described by Oppenheimer and Patterson (2002) in order to overcome the above-mentioned issues and secondly, at the end of this paper, we propose some extensions of the WSBPEL specification, which allow a process engine to significantly optimise the auditing process.

McGregor and Schiefer (2003) developed a method to enable business processes defined using WSBPEL to log audit information to a centralised auditing service which is part of a Solution Management Service, by establishing the Solution Management Service as a partner within the WSBPEL process definition. Nevertheless, they do not provide any details on how to automatically extend a WSBPEL process for auditing it with web services. In this paper, we want to fill this gap and show an automated approach for making WSBPEL processes auditable.

4 Enabling auditing for existing WSBPEL processes

In this section, we propose a new approach to automatically enable auditing in WSBPEL processes. The basic idea is to extend a WSBPEL process definition with probe points in order to report relevant 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.

Figure 2 illustrates the proposed solution. Starting with an existing WSBPEL process, we developed an XSLT stylesheet which extends the WSBPEL process with probe points. From the WSBPEL process perspective, the probe

points are normal activities that invoke an auditing web service. In other words, the XSLT transformation, which creates an auditable version of the original process, does not affect its portability.

Figure 2 Creating an auditable version of a WSBPEL process

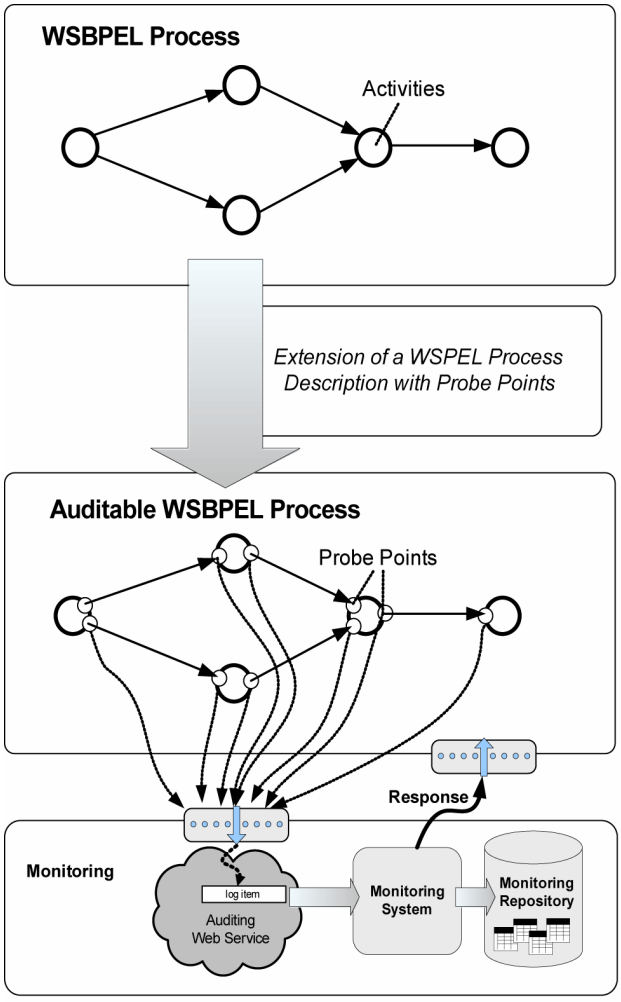


Figure 2 shows in the middle of the figure how activities are extended, that is, pre and post to each audited activity. Furthermore, Figure 2 shows a monitoring system which consumes and processes the audit trail data from the auditing web service. If the monitoring service discovers exceptional situations within the business process, it can respond to these situations by changing the state of the business process. When using WSBPEL, this response can also be done through a web service interface. In other words, the illustrated approach fully enables web service-based monitoring of business process. Please note that this paper does not discuss details on the processing of the audited information for monitoring purposes. For further details, please refer to Baresi and Guinea (2005).

We developed our auditing tool with the goal of reporting as much information as possible about the process state. In our approach, we create a shell around the process activities with pre and post-auditing steps that capture the audit trail information as shown in Figure 3.

For a WSBPEL process, more than one entry point may exist for creating a new process instance after invoking a *receive* or *pick* activity with their *createInstance* attribute set to *yes*. Since these two activities represent the starting points for the instantiation of the process, no activity can be executed before them. Thus, it does not make sense to extend the process definition with a *preaudit* activity for these activities as shown in Table 1.

The question mark indicates that it depends on the activity definition whether an audit activity can be included or not. For instance, if an audited activity has the attribute *createInstance* set to *no*, the preaudit step will be added otherwise it will be omitted. A *throw* activity immediately causes the WSBPEL engine to call associated fault handlers and leaves the current branch of the process, leaving the *post-audit* useless.

Figure 3 Auditing steps

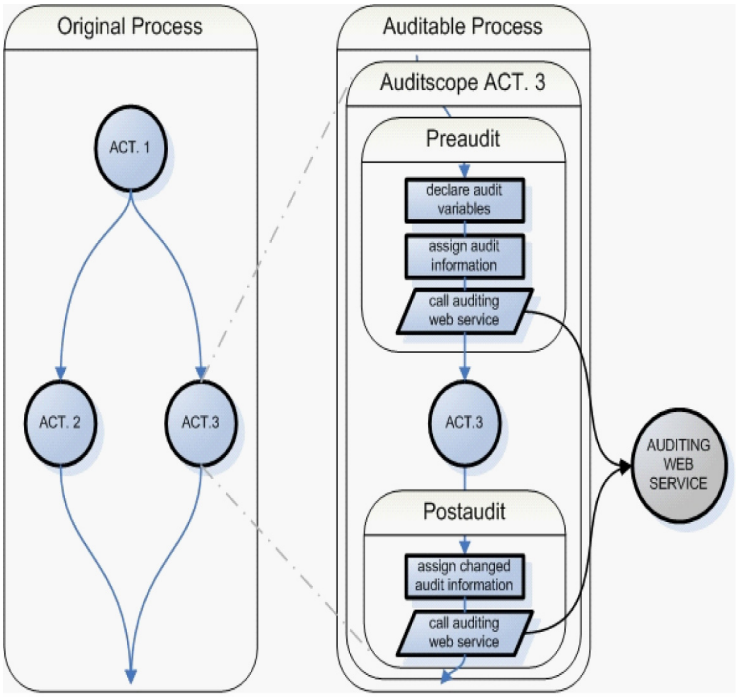


Table 1 Pre- and post-audits cannot be applied to all activities

	Preaudit	*Post-audit*
Receive	?	✓
Reply	✓	✓
Invoke	✓	✓
Pick	?	✓
Switch	✓	✓
Throw	✓	×
Catch	×	✓
Compensate	✓	✓
Empty	✓	✓
Wait	✓	✓
Terminate	✓	×
Assign	✓	✓
Scope	✓	✓
Sequence	✓	✓
Flow	✓	✓
While	✓	✓

For every audited activity, a new *scope* is created which hosts and executes all the necessary steps for *pre* and *post-audit* (see also Figure 3). The auditing steps within this scope are as follows.

Variable creation: an auditing variable is created which is used during runtime to capture runtime information from the audited activities and for forwarding it to an auditing web service.

1 *Preaudit variable assignment*: depending on the type of audited activity, input data is assigned to the auditing variable (e.g. a *reply* activity has only input data, while a *receive* activity has only output data).

2 *Preaudit invoke statement*: this step forwards the input data for the activity (held by the auditing variable) to the auditing web service.

3 *Invocation of audited activity*: execution of the audited process activity.

4 *Post-audit variable assignment*: similar to the preaudit, the auditing variable is used for capturing the output data for the process activity.

5 *Post-audit invoke statement*: the output data (held by the auditing variable) is forwarded to the auditing web service.

Figure A1 included in the Appendix provides an overview concerning which information from each of the observed activities is captured and will be sent to the auditing web service during process execution.

5 Auditing web service

The auditing web service is invoked whenever a *pre* or *post-audit* part of an audited activity is reached. It provides an operation for each of the audited activities. The web service is stateless and therefore very scalable. It accepts requests from the WSBPEL process engine containing the process state information in the form of an SOAP XML document and can be used to forward the auditing information to subscribing target systems (e.g. a monitoring system).

The auditing web service provides an operation for each of the audited activities. The parameters for these operations represent the message types used to declare the variables for invoking the auditing web service. The parameter types are either *strings* or *XmlAnyElements* which can be used to store any type of XML content of a process variable.

In the following, we show two examples for different types of web service operations which are called by the WSBPEL process engine.

Figure 4 shows the information passed from the WSBPEL engine when auditing a *receive* activity. All the data passed to this auditing web service operation represents the data that is also available to the audited *receive* activity. In the case of the *receive* activity, we capture the name of the activity, the partner link, the port type, the operation, the variable name as well as the variable value, and all the correlation sets used within the whole process. Furthermore, identifiers are captured for later analysis purposes, namely *processid*, *processinstanceid*, *activityid*, *activityinstanceid* and *probepoint*, which are explained in more detail in the next section. With this information, the auditing web service can create a complete snapshot of the data used to process the *receive* activity.

Figure 4 Auditing method for receive activities

```
[WebMethod]
[SoapDocumentMethod(OneWay = true)]
public void auditReceive(
        string name,
        string processid,
        string processinstanceid,
        string activityid,
        string activityinstanceid,
        string probepoint,
        string partnerLink,
        string portType,
        string operation,
        string variableName,
        [XmlAnyElement("correlations")]
          XmlElement correlations,
        [XmlAnyElement("variableValue")]
          XmlElement variableValue)
        {
           // Processes auditing request
        }
```

For the second example, we want to use a more complex WSBPEL activity, namely a *switch* activity. Figure 5 shows how we added the two probe points to the activity. As can be seen in Figure 5, we used three web service operations (Figure 6) for capturing the audit trail for

various situations when processing a *switch* activity. The method *auditSwitch()* is called before the *switch* activity is executed. It captures the name of the switch activity as well as existing correlation information. Before calling the activities in the *case* or *otherwise* branches of the *switch* activity, we insert an additional probe point for capturing the decision made by the *switch* statement. If a *case* branch of the switch activity is called, the web service operation *auditSwitchCase()* is invoked and *auditSwitchOtherwise()* for the *otherwise* branch.

Figure 5 Probe points for switch activities

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

Figure A2 included in the Appendix shows a meta-model for all WSDL message types accepted by the auditing web service. For each audited process activity, a web service operation corresponding to one of the WSDL message types is invoked. For example, if a *receive* activity is audited, the *auditReceiveType* with its attributes (mostly are inherited) results in the web service method shown in Figure 4. The WSDL message types are also used to declare auditing variables (see previous section) for capturing the audit trail information of process activities.

For our prototype, we implemented the auditing web service in C#, running on a Microsoft Internet Information Server. We used our auditing web service for capturing audit trail data from a Java-based WSBPEL process engine (Oracle BPEL Process Manager) showing that our auditing approach works platform independent.

6 Correlation of audit data

The auditing web service captures information about correlations used within a WSBPEL process. Correlations within WSBPEL processes are defined by correlation sets which are used within process activities. Target systems which receive and process the captured audit trail information often require linking the data from multiple requests to the auditing web service in order to calculate performance data or perform various types of analysis. The correlations needed by these target systems for audit trails very often cannot be foreseen when designing the business process. Very often typical target systems require a lot more correlation information than a WSBPEL process requires for managing a process instance during runtime.

Figure 6 Auditing method for switch activities

```
[WebMethod]
[SoapDocumentMethod(OneWay = true)]
public void auditSwitch(
        string name,
        string processid,
        string processinstanceid,
        string activityid,
        string activityinstanceid,
        string probepoint,
        [XmlAnyElement("correlations")]
        XmlElement correlations)
        {
           // Processes auditing request
        }

[WebMethod]
[SoapDocumentMethod(OneWay = true)]
public void auditSwitchCase(...)
        string name,
        string processid,
        string processinstanceid,
        string activityid,
        string activityinstanceid,
        string probepoint,
        string condition,
        [XmlAnyElement("correlations")]
        XmlElement correlations)
        {
           // Processes auditing request
        }

[WebMethod]
[SoapDocumentMethod(OneWay = true)]
public void auditSwitchOtherwise(...)
        {
           // Processes auditing request
        }
```

A simple example would be the calculation of the cycle time for a process instance. For such a metric calculation, a target system requires sufficient information to identify the event when a process was started and when it was completed. Nevertheless, a WSBPEL process model might not have a correlation set for these events or in the worst case, it might not have any correlation set at all. Target systems heavily depend upon the possibility to correlate audit trail information that has been reported to the auditing web service. Even simple questions like processing times of an activity or process depend on the correlation of audited activities which is essential to reveal meaningful performance data.

To uniquely distinguish and correlate a reported activity, five identifiers are added in our approach:

- *Process ID*: is used to determine the overall process definition within which the audited activity is defined. Per default, the name of the process definition is used as process ID. If no name has been defined, an artificial ID is generated.
- *Process instance ID*: is used to group all activities belonging to the same process instance.

- *Activity ID*: determines which activity of a process definition is being reported. Since multiple activities can have the same name within a process description, we decided to generate artificial activity IDs which represent the sequential position of the activity within the process description.
- *Activity instance ID*: is used to group the auditing requests for a single activity. With activity instance IDs it is possible to identify the corresponding preaudit and post-audit for a particular process activity.
- *Probe point*: specifies whether the *pre* or the *post-audit* part is being reported.

There are different ways to generate these identifiers. In the following, we introduce two possible solutions. For our prototype, we used the Oracle BPEL Process Manager as process engine and therefore, we were able to take advantage of Oracle XPath extensions for assigning and generating instance IDs. Figure 7 shows how we assigned the process instance ID to an auditing variable using the Oracle XPath extension *ora:getInstanceId()*.

Figure 7 XSLT instructions for assigning the process instance ID

```
[...]
<xsl:element name="copy">
  <xsl:element name="from">
    <xsl:attribute name="expression">
          ora:getInstanceId()</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:processinstanceid
    </xsl:attribute>
  </xsl:element>
</xsl:element>
[...]
```

However, there is also a vendor independent solution for generating instance IDs. Since the XPath expressions defined in the WSBPEL specification do not offer a way to auto-generate these unique identifiers, the auditing web service can be used as a generator for these identifiers.

Whenever the very first activity of a new process instance is audited, the process instance ID as well as the activity instance ID are not set. In such a case, the auditing web service returns a process instance ID which is used for every subsequent audited activity within the same process instance. Furthermore, an activity instance ID is returned for every preaudit of an activity.

Although this approach is fully vendor and platform independent, it implicates more complex stylesheets for transforming a WSBPEL process description as well as additional pre and post-processing of activities during process execution.

7 Tool support for WSBPEL auditing

In the following, we introduce a proof-of-concept prototype which we developed for the previously discussed auditing approach for WSBPEL processes. Our prototype is able to generate a fully auditable version of a WSBPEL process by automatically extending activities with pre and post-audits.

Users can load a WSBPEL process description and specify for each activity if and how it should be audited (pre and/or post-audits). After defining the auditing parameters, the tool automatically generates an auditable version of the process.

For better managing the XSLT transformations, our tool is using so-called *audit type configurations*. As shown earlier, the extension of activities with pre and post-audits can vary significantly. An audit type configuration defines all transformation steps required for a particular activity type. Each audit type configuration contains XSLT stylesheet instructions for an activity type which describe how to:

- find all occurrences of an activity type within a WSBPEL process definition
- extend the found activities with preaudits
- extend the found activities with post-audits.

After applying the audit type configurations to all activities of a process, the tool merges all transformed activities to a final auditable WSBPEL process description.

Our tool allows a user to select activities to be audited and to choose between preaudits, post-audits or both. Figure A3 shows how this selection is made using our prototype. Afterwards, the tool uses these settings to automatically extend the selected activities with probe points.

The example shown in Figure A3 uses a simple loan application process to illustrate this step. The loan application process basically receives a loan request, requests a credit rating index used to query two loan providers, selects the better offer and sends a reply back to the client. In the illustrated example, we extend the activity for querying the credit rating service with pre and post-audits.

When loading this process into our tool, a user can select the activities which should be audited. After providing this information, it is able to automatically generate the auditable version of the WSBPEL process. All required steps to audit the activity are encapsulated within a new scope. First, all information available about the audited activity is assigned together with the additional identifiers (discussed in Section 6) to a new local auditing variable. This variable is used to invoke the auditing web service. Afterwards, the original credit rating service query activity is executed. The results of this web service call are assigned again to the local auditing variable before finally invoking the auditing web service for the post-audit.

The captured data, such as variable values, timestamps, as well as correlation values provide valuable information

about a process instance which can be utilised by target systems to calculate performance data (e.g. cycle times) or track the execution path of a WSBPEL process.

8 Introducing orthogonal concerns into WSBPEL processes

In this paper, we deal with the issue of auditing business processes. In the current prototype, the necessary meta-information about the monitoring details is kept in a configuration file (eventually generated and managed by the GUI tool presented here). However, auditing can also be seen as an *orthogonal* or *crosscutting concern* also called aspect in this context, as is shown in Kiczales et al. (1997).

Since the main intention of WSBPEL is to provide an executable process description, it is not the main concern of a business execution language to provide monitoring. WSBPEL lacks in providing metadata for monitoring purposes, such as information about which activities should be audited, which auditing information should be captured from activities, and details about services which should receive the audit trail data. In general, there are several conceivable options to provide the necessary metadata:

- The first option was explained in further detail here: the meta-information can be stored separately in external configuration (files). This approach has the advantage of being non-invasive; however, we face similar problems here as in for example, deployment descriptors in J2EE or mapping information in O/R tools: the actual process and the meta-information have to be kept in sync, which can be cumbersome in many practical cases.
- The second option is making extensions of the (WSBPEL) standard, but this is difficult from the practical point of view. Moreover, many different crosscutting concerns can be assumed, and a standard probably should not support all of them in the core specification. Additionally, standard extensions are usually a tricky issue – particularly considering global standards driven by some big-players who might not share our vision of Business Process Management.
- A third option is using extension points of WSBPEL. Using this option, additional auditing meta-information is added to the process definition by introducing a new namespace for the extension elements. WSBPEL provides sufficient flexibility for adding meta-information for crosscutting concerns, such as auditing. In the following, we will discuss this option in further detail.

WSBPEL includes an <extensions> Element which allows defining new namespaces that can be used for elements required for the extension. As our proposed extensions are not critical to the process execution, they can be declared as ‘mustUnderstand='no'’, hence the WSBPEL processor is allowed to ignore them. Then, attributes or elements in this namespace can be used to annotate respective WSBPEL elements to add the meta-information required for the auditing. This type of annotation is actually similar to annotations in modern programming languages like Java 5, where such annotations are used to define aspects or O/R mapping declarations.

However, from the practical point of view, these annotations should be created by the user interface for designing the process model and interpreted by the monitoring extension to create the monitoring code.

Our proposed approach for auditing WSBPEL processes is a first step for enabling business processes with standard-based auditing mechanisms. The ultimate goal of our work is to use the experiences we gained from building auditing tools designed for WSBPEL processes to propose effective auditing extensions for the WSBPEL process standard.

However, it is clear that an extension of the WSBPEL process standard would have advantages beyond our definition of the concern:

- By extending the schema for the WSBPEL process definition, organisations could easily specify which information of a business process should be audited in a standard-based way.
- The web service interface for the auditing web service could be defined as part of the WSBPEL standard. This would allow tooling vendors and companies to adapt their own implementations for handling audit trail information.
- By using a web service for the auditing, new service providers could possibly arise, who could offer services for measuring and analysing business process performance. These service providers could offer sophisticated ways for processing process audit trails (e.g. process tracking, process mining, etc.) and also the possibility of linking audit trail data of processes from multiple organisations.
- A standard extension would bring auditing into common WSBPEL tools (e.g. process design and management tools, integrated development environments).

The WSBPEL extension for the auditing concern shown in the next sections in this paper has the advantage that it can be used within the current WSBPEL standard, and also that it could be used as a suggestion for future standard extensions.

8.1 Standard for auditing web service

In Section 5, we showed the operations of an auditing web service-based on both simple and complex activities. For standardisation, the WSDL has to be defined, which can handle all possible requests from a WSBPEL process engine to the auditing web service. As shown in the examples in previous sections, we propose using operations for logging *pre* and *post-auditing* of WSBPEL activities. As we have shown for the *switch* activity, there might be the need for multiple operations for a *pre* or *post-auditing* step.

8.2 Linking the WSBPEL process with the auditing web service

If a WSBPEL process engine should be able to invoke the auditing web service, we have to define the location of the auditing service somewhere. We can do this by either defining an additional partner link in the WSBPEL process description or by introducing a new attribute or element in the WSBPEL schema. In our opinion, it would be useful to allow multiple auditing web services in a WSBPEL process definition.

8.3 Auditing settings in WSBPEL processes

The WSBPEL process requires additional information about which activities should be audited. Due to performance reasons, it is often desirable to only audit process activities which are relevant and of interest. For this reason, we propose two extensions.

The extensions include auditing settings which are included in the process description in two parts. A *centralised part* which allows to define and control the general auditing configuration such as the location of the auditing web service(s) (in case it is not specified as a partner link), log levels (which level of details should be captured from the activities) and settings for exceptional auditing situations (e.g. what should happen if a request to the auditing web service fails). A *decentralised part* for each WSBPEL activity which includes attributes for defining whether auditing is enabled and overwriting the default auditing settings.

Figure 8 shows the proposed extensions for a WSBPEL process. The <*auditing*> element captures all centralised auditing settings and includes information about

- Available *auditing web services* which are linked with partner links.
- The *default settings for the auditing* such as the default auditing web service, the auditing scope (indicating pre or/and post-auditing), the auditing correlation settings (indicating whether information about correlation sets should be captured) and auditing request settings (indicating whether the request to the auditing web service should be executed synchronously or asynchronously).
- A *fault handler* which will be invoked if a request to the auditing web service fails. The fault handler can include any WSBPEL activity.

Furthermore, every WSBPEL activity can overwrite the default settings defined in the <*auditing*> element. With this approach, it is possible to centrally define default auditing behaviour, but still have the possibility to define individual auditing settings for certain WSBPEL activities (e.g. skipping some activities from the auditing).

With the auditing setting in Figure 8, all activities of the process description are audited with full details on scope and correlation settings to auditing service ‘Service1’. For the *invoke* activities with the name ‘a1’ and ‘a2’, an audit trail is captured after the execution of the activities. Furthermore, instead of using the default auditing service (=‘Service1’) for activity ‘a2’, the auditing service ‘Service2’ is used. For activity ‘a3’, no audit trail information is captured. With the proposed WSBPEL extensions, companies can define auditing settings on a coarse level (e.g. process level, sequence level, scope level) as well as on a very detailed level, such as for invoke, receive and reply activities.

Figure 8 Extension of a WSBPEL process

```
<process ... xmlns:audit="auditNsURL">
    ...
    <extensions>
        <extension namespace="auditNsURL"
              mustUnderstand="no"/>
    </extensions>
    ...
    <audit:configuration>
      <audit:auditingServices>
       <audit:service audit:name="Service1"
      audit:partnerLink="myAuditingServiceLink1"/>
        <audit:service audit:name="Service2"
      audit:partnerLink="myAuditingServiceLink2"/>
      </audit:auditingServices>
      <audit:defaultAuditing
         audit:audit="true"
         audit:service="Service1"
         audit:scope="full"
         audit:correlationSettings="true"
         audit:asyncRequest="true" />
      <audit:faultHandler>
         <empty/>
      </audit:faultHandler>
    <audit:configuration>
    ...
    <invoke name="a1" ... audit:audit="true"
     audit:scope="post"/>
    ...
    <invoke name="a2" ... audit:audit="true"
     audit:scope="post" audit:service="Service2"/>
    <invoke name="a3" ... audit:audit="false"/>
</process>
```

8.4 Standard for correlating probe points

Section 6 explains the necessity of grouping incoming auditing requests in order to provide the required information for a later analysis. To avoid building a framework on the side of the auditing web service only to keep track of this correlation problem, the information should be obtained from the WSBPEL engine. The approach used and the information needed to correlate each audit request is another candidate for standardisation.

9 Conclusion and future work

In large organisations, huge amounts of data are generated and consumed by business processes. Conventional

BPMSs do not provide an open infrastructure for capturing process audit trails, and the aim of this paper was to show an approach to fill this gap. This paper has presented an approach for a shareable, web service-based auditing for WSBPEL processes that collects runtime information about business processes and makes them available for a broad range of applications. Additionally, we discuss various cross-cutting concerns, also called aspects that can be used orthogonal to a BPEL process definition. We go into further detail with the proposed extensions for WSBPEL in order to include auditing mechanisms on coarse and detailed granularity levels combined with an auditing web service for logging data during process execution.

The work presented in this paper is part of a larger, long-term research effort aiming at developing a service-oriented Business Process Monitoring platform. Future research and development efforts will focus on building a monitoring system that continuously receives, processes and augments process events and transforms these events in near real-time into performance indicators and intelligent business actions. Our goal is to use the proposed auditing approach for capturing changes of process states in a way that enables the monitoring system to automatically discover and analyse business situations or exceptions and take reactive and proactive actions, such as generating early warnings, preventing damage, loss or extensive cost, exploiting time-critical business opportunities or adapting business systems with minimal latency.

References

Baresi, L. and Guinea, S. (2005) 'Towards dynamic monitoring of WS-BPEL processes', *Proceedings of the 3rd International Conference of Service-Oriented Computing*, Vol. 3826, Amsterdam: Springer, pp.269–282.

Bruckner, R.M., Jeng, J.J. and Schiefer, J. (2003) 'Real-time workflow audit data integration into data warehouse system', *Proceedings of the 11th European Conference on Information Systems*, Naples.

Business Process Execution Language for Web Services version 1.1 (2003) Specification, May.

Gavirneni, S., Kapuscinski, R. and Tayur, S. (1999) 'Value of information in capacitated supply chains', *Management Science*, Vol. 45, No. 1, pp.16–24.

Hacigümüs, H. (2006) 'Middleware support for auditing service process flows', *Proceedings of the 1st Workshop on Middleware for Service Oriented Computing*, Vol. 184, Melbourne: ACM, pp.24–29.

Holton, R., Dreiling, A., Zur Muehlen, M. and Becker, J. (2002) 'Enabling technologies for supply chain process management', *Proceedings of the Information Resource Management Association International Conference*, Seattle, International Resources Management Association, pp.864–868.

Jeng, J.J., Schiefer, J. and Chang, H. (2003) 'An agent-based architecture for analyzing business processes of real-time enterprises', *Proceedings of the 7th International Enterprise Distributed Object Computing Conference*, Brisbane: IEEE, pp.86–97.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J-M. and Irwin, J. (1997) 'Aspect-oriented programming', *Proceedings of the 11th European Conference on Object-Oriented Programming*, Vol. 1241, Berlin: Springer, pp.220–242.

Lazovik, A., Aiello, M. and Papazoglou, M. (2004) 'Associating assertions with business processes and monitoring their execution', *Proceedings of the 2nd International Conference on Service Oriented Computing*, New York: ACM,
pp.94–104.

Leymann, F. and Roller, D. (2000) *Production Workflow: Concepts and Techniques*, Upper Saddle River: Prentice Hall.

McGregor, C. and Kumaran, S. (2002) 'An agent-based system for trading partner management in B2B e-commerce', *Proceedings of the 12th International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems*, San Jose: IEEE, pp.84–89.

McGregor, C. and Schiefer, J. (2003) 'A framework for analyzing and measuring business performance with web services', *International Conference on E-Commerce*, Newport Beach: IEEE, pp.405–412.

Oppenheimer, D. and Patterson, D.A. (2002) 'Architecture and dependability of large-scale internet services', Vol. 6, No. 5, IEEE, pp.41–49.

Sayal, M., Casati, F., Dayal, U. and Shan, M-C. (2002) 'Business process cockpit', *Proceedings of the 28th VLDB Conference*, Hong Kong: Morgan Kaufmann, pp.880–883.

Workflow Management Coalition (1998) *Audit Data Specification.*

Zur Muehlen, M. (2001) 'Process-driven management information systems – combining data warehouses and workflow technology', *Proceedings of the 4th International Conference on Electronic Commerce Research*, Los Alamitos: IEEE, pp.550–566.

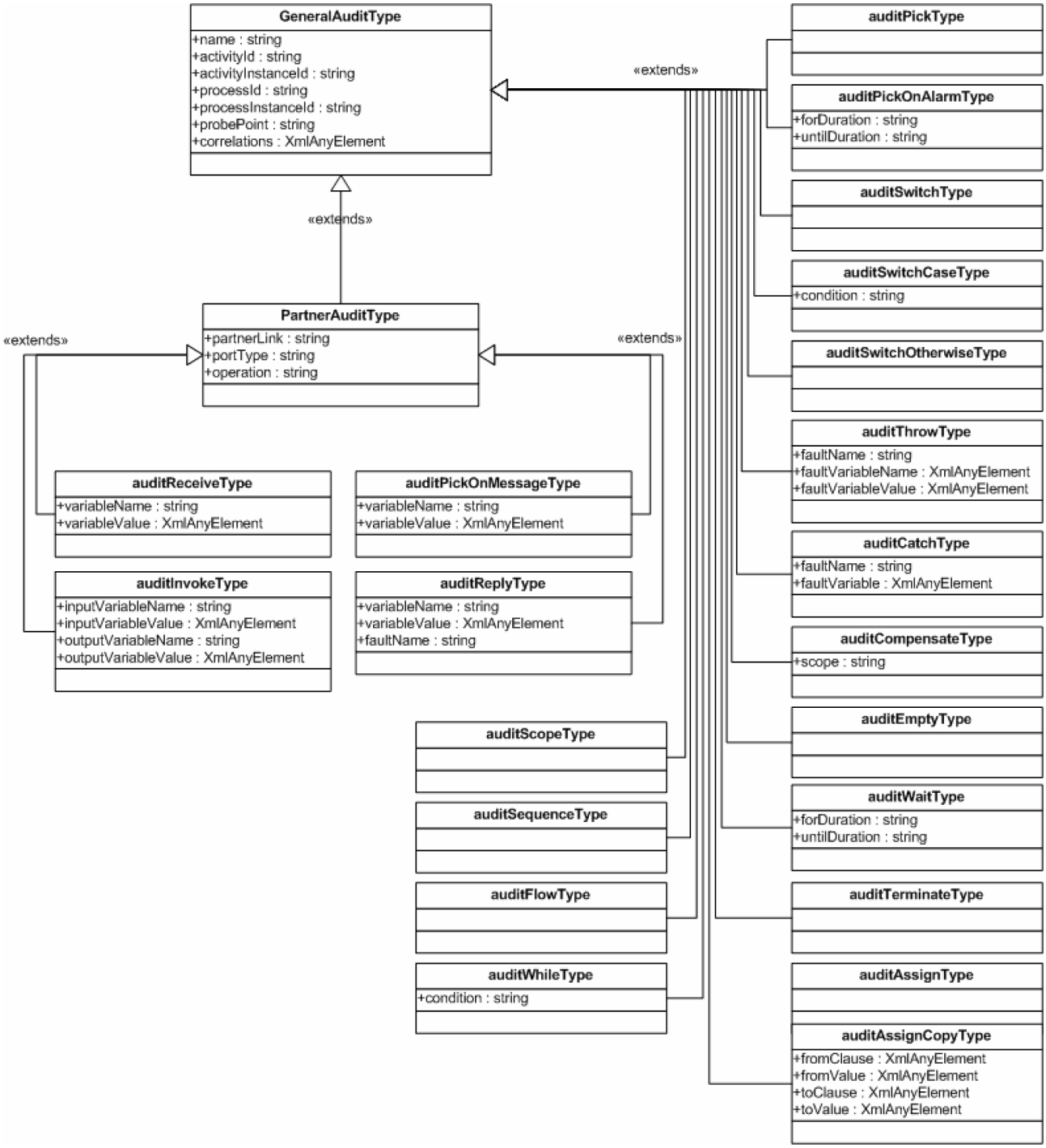
Appendix

Figure A1 List of audited attributes

Activity	Name	partner-Link	portType	operation	(input-) Variable-Name	(input-) Variable-Value	output-Variable-Name	output-Variable-Value	correlations	*Additionally reported items*
Receive	✓	✓	✓	✓	✓	✓	x	x	✓	
Reply	✓	✓	✓	✓	✓	✓	x	x	✓	faultName
Invoke	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Pick	✓	x	x	x	x	x	x	x	✓	
Pick – onMessage	✓	✓	✓	✓	✓	✓	x	x	✓	
Pick – onAlarm	✓	x	x	x	x	x	x	x	✓	durations
Throw	✓	x	x	x	faultVar	faultVal	x	x	✓	faultName
Switch	✓	x	x	x	x	x	x	x	✓	
Switch – Case	✓	x	x	x	x	x	x	x	✓	condition
Switch – Otherwise	✓	x	x	x	x	x	x	x	✓	
Catch	x	x	x	x	faultVar	faultVal	x	x	✓	faultName
Compensate	✓	x	x	x	x	x	x	x	✓	scopeName
Empty	✓	x	x	x	x	x	x	x	✓	
Wait	✓	x	x	x	x	x	x	x	✓	durations
Terminate	✓	x	x	x	x	x	x	x	✓	
Assign	✓	x	x	x	x	x	x	x	✓	
AssignCopy	x	x	x	x	fmClause	fromValue	toCluase	toValue	✓	
Scope	✓	x	x	x	x	x	x	x	✓	
Sequence	✓	x	x	x	x	x	x	x	✓	
Flow	✓	x	x	x	x	x	x	x	✓	
While	✓	x	x	x	x	x	x	x	✓	condition

Appendix (continued)

Figure A2 Auditing web service operation message types



Appendix (continued)

Figure A3 WSBPEL auditing tool – choosing which activities to audit