Non-Intrusive Monitoring and Service Adaptation for WS-BPEL

Oliver Moser, Florian Rosenberg and Schahram Dustdar
Distributed Systems Group, Technical University Vienna
Argentinierstr. 8/184-1, 1040 Vienna, Austria
lastname@infosys.tuwien.ac.at

ABSTRACT

Web service processes currently lack monitoring and dynamic (runtime) adaptation mechanisms. In highly dynamic processes, services frequently need to be exchanged due to a variety of reasons. In this paper we present VieDAME, a system which allows monitoring of BPEL processes according to Quality of Service (QoS) attributes and replacement of existing partner services based on various (pluggable) replacement strategies. The chosen replacement services can be syntactically or semantically equivalent to the BPEL interface. Services can be automatically replaced at runtime without any downtime of the overall system. We implemented our solution with an aspect-oriented approach by intercepting SOAP messages and allow services to be exchanged during runtime with little performance penalty costs, as shown in our experiments, thereby making our solution suitable for high-availability BPEL environments.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Programming Environments; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures, Languages*

General Terms

Design, Languages, Reliability, Performance

Keywords

BPEL, Quality of Service, Monitoring, Service Selection, Message Mediation

1. INTRODUCTION

Increasingly, organizations try to automate their business processes using coarse-grained services that implement specific parts of their business functionality. Services are self-describing, platform-agnostic computational elements that support rapid low-cost composition of distributed applications [22]. Recently, Web services are one popular technology for implementing such services. Composing different services into a structured process can be done by using an orchestration language (also called composition language) such as WS-BPEL, the Web Service Business Process Execution Language (BPEL for short) [19]. It is an XML-based language that provides the user with variables, conditionals, loops, asynchronous messages, process correlation and facilities for transaction and exception handling. The language itself was originally designed by BEA, Microsoft and IBM. Finally, WS-BPEL 2.0 is now standardized by OASIS.

When leveraging BPEL as a process execution language, there are some major concerns that have to be considered when using it in high-availability environments. As a motivating scenario, consider an online store where the back-end business logic is implemented using a BPEL based solution, and the front-end for the customers is a Web application which gathers data and hands it over to the BPEL engine for processing the order. The order processing requires some communication with the local back-end services (such as stock services) or partner services offered by other organizations (e.g., credit card verification and payment). A failure or downtime of one of these services in the process can cause a downtime in the overall process execution. This may result in a considerable loss of money due to the fact that the front-end is relying on a fully operational back-end system.

In this work, we address two issues we identified when using BPEL in enterprise systems where monitoring and high-availability play a crucial role: Firstly, one major drawback of BPEL is its inherently static nature. Basically, if a process definition is deployed into a BPEL environment – the BPEL engine – it cannot be changed dynamically at runtime. Every information is hard-wired after a process is deployed, for example, references to other services (called *partner links*) used in the BPEL process cannot be changed and exchanged without editing and redeploying the process which implies a downtime of the overall system. Although it is possible for the process to bind to partner links at runtime, the process definition would contain a tremendous amount of code that is not related to the business process (see Section 4 for more details on runtime partner link binding). A dynamic replacement of partner Web services in the process is, therefore, a necessary approach when the service provider quality in terms of response time (or other QoS) aspects is not good enough and affects the overall quality of the process.

Secondly, the BPEL standard does not provide any means for monitoring a running process. Nevertheless, monitoring of business processes is a very important issue in enterprise systems as they are a key factor for running a business. Currently, it is up to the BPEL engine to provide monitoring interfaces. However, to the best of our knowledge, available BPEL engines lack this ability. The monitoring of QoS is a necessary foundation to decide whether a replacement of

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

a service should be performed and also what kind of QoS-based selection criteria should be used.

We address these aforementioned issues in our VieDAME (Vienna Dynamic Adaptation and Monitoring Environment for WS-BPEL) system[1]. It is an extension to the ActiveBPEL engine [1] and allows monitoring of various QoS attributes of running BPEL processes and to perform a fully dynamic service adaptation of existing processes in a non-intrusive way by providing a number of alternative services for a given service.

Additionally, we provide a mechanism to handle *service interface mismatches* by allowing transformations to be applied to incoming and outgoing SOAP messages to adapt them to the currently used interface in the BPEL process. We show that the performance overhead introduced by our extension is minimal and the overall system performance is sufficient for high performance service based applications. Furthermore, we provide a Web-based administration interface allowing access to monitoring data and the configuration of the adaptation and its possible transformations.

This paper is structured as follows: In Section 2 we discuss the basic model of our approach as well as the detailed system architecture and its components. Section 3 describes the monitoring strategy for BPEL processes and Section 4 presents the service adaptation and the necessary message transformation if the service interfaces do not match. In Section 5 we briefly describe the system implementation and the technologies we used to build the extension. Section 6 presents the case study that we use in our approach and depicts an in-depth performance evaluation to show that our system is well-suited for high-availability environments. Related work is discussed in Section 7 and Section 8 concludes this paper and presents some future work.

2. VIEDAME APPROACH

In this section, we firstly discuss the design goals for the VieDAME system, followed by a system overview that briefly introduces the architectural approach. An in-depth discussion of the VieDAME system architecture concludes this section.

2.1 Conceptual Goals

In general, dynamic process adaptation deals with a multitude of different aspects concerning the adapted parts of a process and the applied adaptation techniques. The main idea for VieDAME was to achieve *non-intrusive behavior* with regard to dynamic service adaptation, which enables the runtime exchange of partner links within a BPEL process, without any changes to the BPEL process or the involved partner services.

Such a dynamic service adaptation only leverages one possible fragment of a process model that can be adapted at runtime. In general, there is no restriction what fragments of a process need to be adapted and why. Service adaptation is one of the most obvious requirements that stems from the fact that BPEL has no efficient possibility of dynamically exchanging partner links or adding alternative services to the process at runtime. Unlike other approaches that modify the process definition to redirect partner link communication to a proxy service that is capable of adding additional features such as logging or partner service selection, we propose a lightweight adaptation layer to intercept and control incoming and outgoing (SOAP) message flow (refer to Section 2.3 for details). As an example, the work presented in [8] discusses the implementation of software fault tolerance for Web services by mapping various fault tolerance patterns to WS-BPEL, using standard BPEL constructs only. While being an interesting and standard compliant approach, the main advantage of our solution is the fact that the process model itself does not have to be changed, and, therefore, does not introduce any logic that is not related to the business process itself.

Other adaptation techniques coming from the workflow area allow to change different process fragments at runtime mainly based on execution monitoring [28]. Certainly, such process adaptation techniques can be applied also to BPEL, nevertheless, it would require a tighter coupling between the BPEL engine and the adaptation layer, which is not the chosen approach in this paper. Our contribution includes lightweight techniques to truly adapt services in a process at runtime. We currently do not adapt the process control flow itself, which is, for example, necessary when a service has to be replaced by two other services which fulfill the same functional requirements but require adaptions in the control flow to achieve the same behavior.

In Figure 1 we have depicted our conceptual model we use for the dynamic service adaptation. The core part is the BPEL process which has a certain control flow and invokes a number of partner services. The partner services are generally hosted on different machines distributed over the Web. In VieDAME each service in a BPEL process can be marked as *replaceable* to indicate that alternative services can be configured and invoked instead of the original service that is hard-wired in the process. An alternative service can either be *syntactically* or *semantically* equivalent. The former indicates that the interfaces of the original and the alternative services match. This is, for example, the case, when multiple instances of the same service are hosted on different machines to provide increased reliability. The latter indicates that the services only have the same functionality but expose it using different interfaces which mainly occurs when services needs to be exchanged that come from completely different providers on the Web.

Each service and all of its alternative services' endpoints are stored in the VieDAME service repository. If a service should be dynamically replaced with an alternative service during process execution, the original partner service captured by the VieDAME adaptation layer has to be marked replaceable in the VieDAME UI (right side of Figure 1). Alternative services that can replace the original service defined by the BPEL process may be added at any time by providing their interface description in the VieDAME UI. Once they are linked to the original service, a replacement policy can be selected to control which of the available alternatives will be used (see Section 4.1). Additionally, the VieDAME UI allows the definition of mediation rules that allow alternative services to be used where the interfaces do not match the original interface of the partner service.

2.2 System Overview

The VieDAME system is split into the VieDAME core and the VieDAME engine adapters. The VieDAME core ties together the monitoring, service selection and message trans-

[1]Online demo available at `http://www.vitalab.tuwien.ac.at/prototypes/viedame`

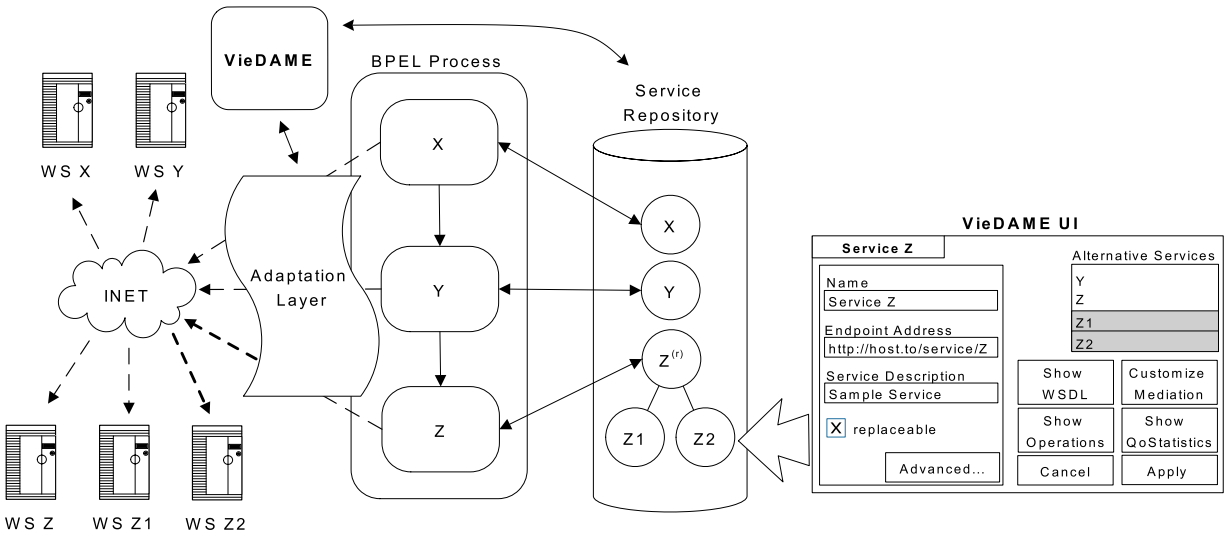

Figure 1: VieDAME enhanced BPEL environment

formation facilities as well as provides services such as data store access, scheduling and configuration data, whereas the engine adapters represent the connector to the BPEL engine. Thus, to support new BPEL engines, it is (only) necessary to implement an engine adapter specifically to the desired engine implementation. The VieDAME environment currently supports ActiveBPEL 3.0 [1], support for Apache ODE [2] and JBoss jBPM WS-BPEL runtime [27] is planned once they are reliable enough to be used in production environments.

The VieDAME engine adapters are implemented using aspect-oriented programming (AOP). Although extending existing (object oriented) software systems is traditionally achieved by applying *subclassing* to add additional functionality, we would fail to reach our goal to keep the base system (i.e., the BPEL engine) and the VieDAME system as separate as possible by taking the OOP approach. The main drawback with subclassing is that the code is dependent on its base classes. Changes in the base system always imply changes in the extension, even if the changes in the new release are unrelated to the extension. Moreover, if the number of classes to be modified is high, it would be likely that one would end up with code that is hard to maintain. These issues, together with the possibility to enable and disable certain parts of the VieDAME system as needed by using *load time weaving*, were the main reasons for the decision towards AOP.

2.2.1 Aspect-Oriented Programming

The following paragraph gives a very short introduction to the key concepts in AOP. For a complete introduction to AOP consider [14] and [15].

An *aspect* in AOP can be seen as a concern that is related to one or more places in existing code. In AOP, those places are called *joinpoints*. To tell the AOP framework where exactly it should add the additional functionality (called *advices* in AOP jargon) it requires the definition of *pointcuts* – expressions that identify arbitrary events in the runtime system such as method invocations or field access. Finally, the AOP framework combines the base system with the additional code. This step is called *code weaving* and affects the performance and ease of deployment of the altered code. Whereas compile-time weaving requires an intermediate step to generate the modified classes, *load-time weaving* adds the advice code during class loading time. As an additional benefit, aspects can be deployed and undeployed during runtime. Another approach is *source-level weaving*, which is not popular any more since it requires access to the base system's source code. Section 3 explains the `ServiceStatisticsCollectorAspect`'s advice code and shows sample pointcut expressions for JBossAOP.

2.3 Architecture

Figure 2 depicts the architectural approach taken as well as the system dependencies. Firstly, the flow of events in a standard BPEL environment is described, without any interaction with the VieDAME system. Secondly, the additional steps performed in a full-featured VieDAME environment are explained. It includes service monitoring, alternative service selection and message transformation, which are explained in detail in the subsequent sections. Please note that only a simple scenario – without process correlation and other advanced BPEL features – is used for explanation. Nevertheless, the VieDAME system does not impose any limitations in this regard.

After deployment of a process definition (1), the BPEL processor (2) is ready to create new process instances. A new BPEL process instance (2a) is created when one of its start activities is triggered, e.g., by an incoming message. Interaction with a partner link is initiated by invoke activities (2b) that create SOAP calls (3a). These SOAP calls are executed by a SOAP engine (10) that returns the result of the invocation of an arbitrary partner service (11) upon completion of the request. The invoke activity reports the result to the process instance which in turn proceeds to the next activity.

When the VieDAME system is enabled, an additional level of processing is introduced, manifested in the *Interception and Adaptation Layer* (5b), hereinafter referred to as the *IAL*. Basically, the IAL is created by *aspects* that are

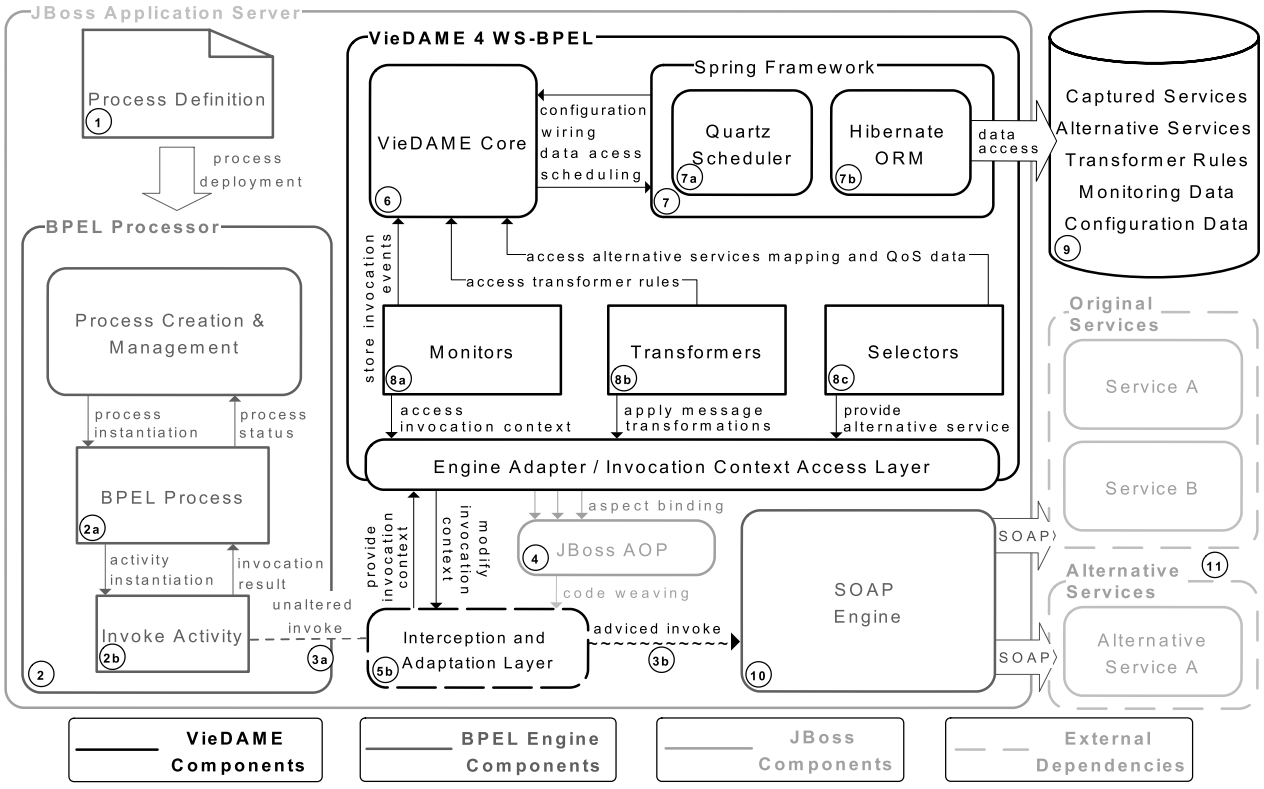

Figure 2: VieDAME Overall System Architecture

bound to specific *joinpoints* in the BPEL engine's code by the definition of *pointcuts*. The *advice* code is then *woven* into the original method invocations by the AOP framework (4) at load time. The IAL provides a bidirectional interface for the *engine adapter* (5a) to tap the communication between the invoke activity (2b) and the SOAP engine (10). The engine adapter in turn provides read-write access to the *invocation context*, enabling other VieDAME components – such as `Monitor` (8a) or `Selector` (8c) – to access and modify invocation parameters and other runtime data.

The first VieDAME component that is called after interception of a partner link invocation by the IAL is the `Monitor` component. It examines the invocation context to find the service name, endpoint address and operation name in order to load a previously persisted service reference or to persist a new service reference for future requests. The `Monitor` leverages the VieDAME core (6) and the ORM framework (7b) respectively to persist objects to a data store (10). Furthermore, the `Monitor` activates a timer to measure the time elapsed during the actual SOAP call and stores this information together with a reference to the involved service and success/failed flag. A scheduling framework (7a) is used to bulk-insert invocation events in order to optimize data store access. The `Monitor` component is explained in more detail in Section 3.

If the service reference loaded by (8a) is marked as replaceable, the next VieDAME component takes control. The `Selector` component (8c) determines an alternative partner service by applying some selection algorithm to a list of configured alternative services (9). If an alternative service is found, the invocation context is updated with the alternative's endpoint parameters. Section 4.1 explains the `Selector` concept in more detail.

Like the `Monitor` component, `Selector`s access the data store by using (6) and (7b). The same applies to the last VieDAME component that can be called, the `Transformer` component. A `Transformer` (8b) is responsible for compensating the interface mismatch between the original service and the alternative. The `Transformer` uses transformation rules (e.g., XSLT stylesheets) stored in (9) to perform the required transformations. Consider Section 4.2 for more information on `Transformer`s.

After all required modifications are applied to the invocation context, the SOAP call is finally proceeded, probably invoking an alternative partner service instead of the original service. The difference between the unaltered invoke (3a) and the advised invoke (3b) is called the *Invocation Context Delta*, or *ICD*. A big ICD indicates many differences between the original service interface and the alternative service interface, whereas a small ICD indicates a replica of the original service (i.e., the original partner service and the alternative partner service only differ in their endpoint address). A zero ICD indicates that neither a service replacement nor message transformation was applied. The ICD measured value can be used as an indicator for determining the degree of adaptation the VieDAME system has performed and whether the environment running VieDAME uses the adaptation facilities at all.

3. SERVICE MONITORING

As the BPEL standard does not specify any details about process monitoring a possible work-around to accomplish

this in BPEL is a combination of `<throw>` and `<catch>` activities that can be utilized to trigger additional activities (e.g., invoking an alerting service) in case of an error.

We think that there are two severe problems with this approach: On the one hand, this solution is not very flexible. If a partner service becomes unavailable or faulty, the process could call an alerting service which in turn could inform the operators about the situation. Fixing the problem, which is only possible if the operators have access to the broken partner service will result in a service downtime. Moreover, using a Web service for sending out alerts about service problems does not seem to be an appropriate solution as it introduces a new single point of failure. In the worst case, the alerting service is down and the operators would not even notice what is going on. On the other hand, exception handling is only useful if a service fails completely to respond to a request, either by returning malformed responses or by being unavailable. If one or more partner services become slow in terms of responsiveness, the impact on business processes could turn out even worse, since the approach discussed above is not applicable in such a case. Imagine a customer clicking the finalize order button, expecting a response in a reasonable amount of time. How long the customer has to wait depends on how fast each of the involved partner services responds. It is not only a matter of "succeeded or failed", but rather fast enough or too slow. If statistical monitoring data about previous service invocations is available, calculating the overall execution time of the BPEL process is possible. The VieDAME system can provide this information, which can be used to optimize the business process in terms of partner service invocations. The `Monitor` component (8a in Figure 2) is responsible for storing the partner service invocation events (i.e., SOAP calls that result from `<invoke>` activities in the BPEL process), that in turn are aggregated to calculate performance related QoS values such as average response time, accuracy or availability of some particular operation.

QoS Attribute	Formula	Category
Response Time	$\frac{1}{\#requests} \sum rt_i$	Performance
Availability	1 - $\frac{downtime}{uptime}$	Dependability
Accuracy	1 - $\frac{\#failedrequests}{\#totalrequests}$	Dependability

Table 1: Monitored QoS Attributes

Table 1 lists the monitored QoS attributes and their underlying formulas. These values serve as input for some `Selector` components that are explained in Section 4.1. Furthermore, other issues such as handling peak-time loads can be addressed by adding additional alternative partner services based on long-term observations.

Listing 1 shows the skeleton of the VieDAME Monitor component, exemplifying the structure of the `Selector` and `Transformer` components and explaining how AOP is used to examine the invocation context. On Line 2–3, an invocation context attribute (`EndpointReference`, in this case) is accessed and the name of the invoked partner service is examined (line 4). The call to `invocation.invokeNext()` on line 9 hands over control to the next applicable aspect (i.e., to the `Selector` component), effectively returning the result of the actual method invocation after all other VieDAME components have succeeded. Line 15 shows the `storeInvocation()` method call that persists the operation invocation event.

```
1  void handleInvoke(MethodInvocation invocation) {
2    EndpointReference endpointRef
3      = (EndpointReference) invocation.getArgs[0];
4    String name = endpointRef.getServiceName();
5    ServiceEndpoint endpoint
6      = endpointDao.loadEndpoint(name, url);
7    try {
8      monitor.start();
9      Object result = invocation.invokeNext();
10     monitor.stop();
11     return result;
12   } catch (Exception e) {
13     success = false;
14   } finally {
15     storeInvocation(operation, duration,
16                     success)
17   }
18 }
```

Listing 1: Monitor Component Structure

The QoS statistics (e.g., availability) for a particular operation are calculated by a scheduled job based on the list of saved operation invocation events. Since the VieDAME system relies on a relational database for persistence, immediately flushing the invocation events to the database can cause serious performance issues when dealing with a lot of concurrent requests. Thus, the VieDAME system leverages a scheduling framework to implement *write-behind* semantics in this regard, allowing to efficiently batch insert a configurable number of events at once. Besides the buffer size used for temporary storage of invocation events, a resize factor can be defined that the VieDAME system uses to dynamically change the buffer size if the backlog of events waiting to be flushed exceeds a configurable threshold. These parameters allow for tuning the VieDAME system to deal with different workload scenarios as well as optimizing database access.

We will now explain how the AOP framework is configured *where* and *when* the code from Listing 1 is applied. Listing 2 shows pointcut definitions and advice-to-pointcut bindings for JBoss AOP that enable the VieDAME system to intercept partner link communication in order to monitor partner service behavior.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <aop>
3  <aspect class="StatisticsCollectorAspect"/>
4  <bind pointcut=
5      "execution(* invokers.*->invoke(..))">
6    <advice name="interceptInvoke"
7      aspect="StatisticsCollectorAspect"/>
8  </bind>
9 </aop>
```

Listing 2: Pointcut definition and Aspect binding

The `<aspect>` tag on line 3 tells the AOP framework where to look for advice methods, whereas `<bind>` (line 4) instructs JBoss AOP to call advice method `interceptInvoke` of aspect `StatisticsCollectorAspect` (lines 6–7) when method `invoke` (`->`) of an arbitrary class (`*`) found in the `invokers` package of the ActiveBPEL engine implementation is executed with any number of arguments of arbitrary type (`..`), returning an arbitrary object (`*`). For more information on JBoss AOP consider [26].

4. DYNAMIC ADAPTATION AND MESSAGE MEDIATION

As aforementioned in Section 2, the VieDAME system takes a lightweight approach to dynamically adapt partner service invocations based on the IAL. This allows to efficiently exchange partner links at runtime. This section shows first, how runtime partner link binding can be performed with standard BPEL constructs, followed by a wrap-up of limitations inherent to this approach and finally explains the two VieDAME components responsible for conquering these limitations.

A partner link endpoint reference in BPEL is represented as a `wsa:EndpointReference` XML element defined by the WS-Addressing [30] standard. It is possible to declare a variable of type `wsa:EndpointReferenceType` and dynamically assign its value to a particular partner link. Juric et al. [13] refer to this technique as *dynamic partner link assignment*. Although it is possible to use this approach to force the process to dynamically *bind* to partner services at runtime, it does not allow *adding* new partner services at runtime, let alone additional partner services that differ from the original in their interface description. Furthermore, it is not possible to change the way *how* and *when* a particular service is picked, i.e., the *selection* criterion is statically hardwired into the process definition (where it conceptually does not belong to).

Taken together, these limitations indicate the need for a more flexible and viable solution to the problem of *dynamically adapting* the process at runtime. The VieDAME environment allows to add *alternative services* while the process is running and choose `Selector`s to define the replacement algorithm. The replacement algorithm in turn determines the alternative service by matching some predefined criteria (i.e., QoS attributes such as availability or response time) against the data gathered by the monitoring component. Thus, the selectors do not solely rely on predetermined data but rather uses up-to-date information about the quality of the alternative service. Following [17] we distinguish between *deterministic* and *non-deterministic* QoS attributes. Deterministic QoS attributes, on the one hand, indicate that their value is known before a service is invoked, including price or penalty rate. On the other hand, their non-deterministic counterpart include all attributes that are uncertain at service invocation time, for example service availability [29]. Dealing with non-deterministic attributes is more complex since it requires to perform calculations based on data gathered during runtime observations (see Section 3). Thus, we will only take these non-deterministic attributes into consideration for the evaluation. Nevertheless, the VieDAME system also supports service selection based on deterministic attributes.

The fact that not all alternative services provide the same interface as the original partner service makes the situation even more challenging. The VieDAME system addresses this issue by supporting *basic* and *extended* transformation components, called `Transformers`. Both the `Selectors` and `Transformers` are key components that need further explanation.

4.1 Service Selectors

A `Selector` (8c in figure 2) in the VieDAME system context is an implementation of some particular selection algorithm that determines which of the available alternative services matches one or more selection criteria best. The VieDAME systems provides a variety of selection algorithms, ranging from simple randomized and round-robin selectors that can be used for load balancing, to more sophisticated selectors that combine several QoS attributes such as performance and dependability for the selection criterion. Moreover, the VieDAME system offers fault-compensating selectors that retry failed service invocations, either with the original service or with an appropriate replacement. To meet further requirements, the VieDAME system can easily be extended with additional `Selector` implementations.

Figure 3 gives an overview of the selector hierarchies. Basically, the *load balancing selectors* are used for choosing a service out of a group of services that are deployed on different nodes to distribute the load and increase the fault-tolerance in a *round-robin* or *randomized* way.

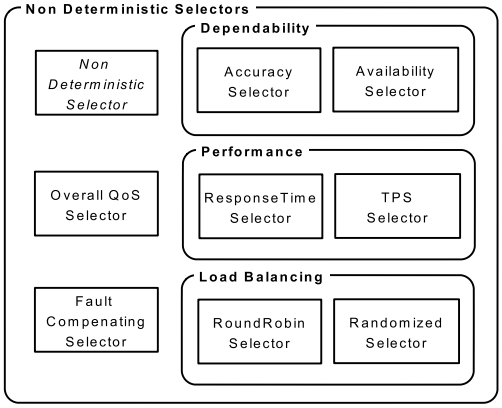

Figure 3: VieDAME Non Deterministic Selectors

The *dependability selector* are used for choosing a service that either has the highest *availability* or the best *accuracy* based on its execution history measured by the monitoring module. The *performance selectors* choose a service based on its *average response time* or it *transactions per second* rate. These different selectors provide a very powerful mechanism to select services based on certain QoS criteria that typically arise in enterprise systems.

4.2 Message Transformers

A `Transformer` in the VieDAME system is a mediation component that compensates the interface mismatch between the original service and an alternative service by applying transformation rules to incoming and outgoing messages. Thus, only *semantic equivalence* is necessary for a service to qualify itself as a replacement for another service. The VieDAME system offers two different types of `Transformer`s that are described below. Figure 4 illustrates the dependencies between partner services, `Transformer`s and transformation rules and the IAL. `BasicTransformer`s accomplish the interface mismatch compensation by substitutions based on regular expressions. While creating transformation rules by leveraging regular expressions is straightforward, this approach is only viable if both interfaces differ only in tag or attribute names. `ExtendedTransformer`s, on the other hand, support full XSLT 2.0 [31] transformations and therefore allow to use alternative services even if their interface descriptions are rather different from the original partner service WSDL. Furthermore, it is possible to assign

a list of transformation rules to an alternative service, which allows the `Transformer` to apply different rules to different parts of the incoming and outgoing messages or implement chained transformations.

In the rare situation that both `Basic-` and `Extended-Transformers` are not applicable (i.e., if the transformation requirements cannot be met by neither regular expression nor XSLT), the VieDAME system allows to configure an external transformation engine such as Apache Synapse [3]. This would also allow to attach other messaging back-ends such as the Java Messaging Service (JMS) or REST based services [11].

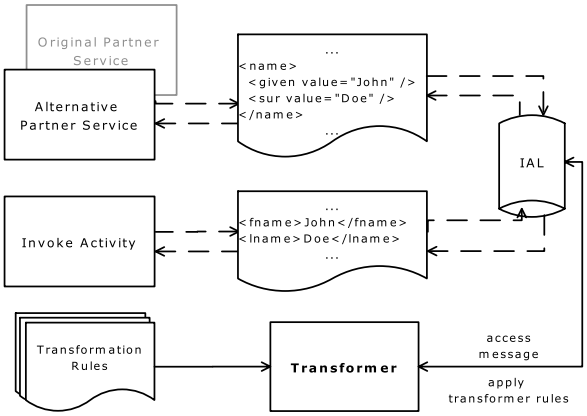

Figure 4: VieDAME `Transformers`

5. IMPLEMENTATION DETAILS

For evaluating our approach, ActiveBPEL 3.0.1 [1] (2) is used (The number in braces references the respective component in Figure 2). The prototype for the VieDAME system was implemented based on the Java 5 platform, configuration, dependency management, and component wiring are provided by the Spring Framework [12] (7). Object-Relational mapping is implemented using Hibernate [23] (7a), which offers sophisticated caching mechanisms the VieDAME system uses to avoid unnecessary database access. JBoss Application Server 4.2.1 [24] (shown in the background of Figure 2) is used for deployment of both the ActiveBPEL Engine and the VieDAME system, JBoss AOP 2.0 [26] (4) seamlessly integrates AOP capabilities into the application server deployment concept. JBoss Seam 2.0 [25], an application framework that integrates JSF and EJB 3.0, was used to implement a Web-based user interface for configuring the VieDAME system, allowing the user to add alternative services by providing the service's WSDL, choose the desired `Selectors` and define/upload `Transformer` rules. Moreover, the partner services QoS statistics are displayed using JFree [21] charts on a per operation basis. We are using Postgres 8.0.13 for the RDBMS component (9).

6. EVALUATION

In this section the VieDAME system performance is compared to the performance achieved with a plain ActiveBPEL setup (without using any VieDAME components) using a load-test scenario with a different number of virtual users. We use a multistage comparison regarding the VieDAME system features to show the additional overhead each component (e.g., a `Transformer`) adds to the plain ActiveBPEL system. In the `Selector` tests, a simple round robin selector is applied, whereas for the transformer tests, an *Extended Transformer* was used that uses an XSLT transformation on the incoming and outgoing SOAP message of the service that needs to be adapted.

6.1 Case Study

The case study we have implemented for evaluation of our approach describes the business process behind an externally available *PurchaseOrder* Web service that is implemented as a BPEL process. It enables partner companies to perform *one-shot orders*, meaning that all activities that are required for placing an order are performed at once. Figure 5 illustrates the process using BPMN (Business Process Modeling Notation) [20].

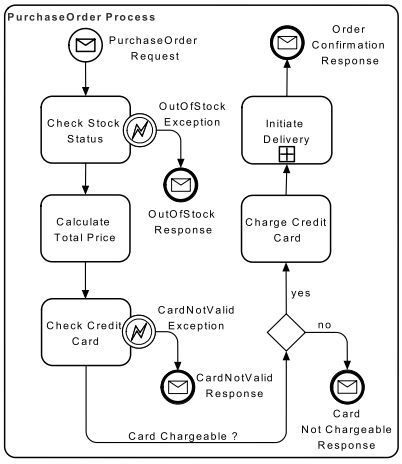

Figure 5: PurchaseOrder Process

The process is triggered upon receipt of the `Purchase-Order` request from the service requester. Then it will check if the required order items are in stock. If one or more items are out of stock or currently not available in the desired quantity, the requester has to be informed and eventually given an alternative. If all items are in stock, then a total price, including additional fees like shipping costs and taxes, has to be calculated. Furthermore, this activity triggers a shipping address validation, which is necessary to calculate the shipping fee. The next step is the validation of the customers payment option. To simplify matters, it is assumed that only major credit cards are accepted for payment. This requires a check to assure that the credit card data supplied by the business partner is valid and the card can be charged. The final steps include the actual charging of the supplied credit card and the initiation of the delivery process which also confirms the successful order placement by sending an email to the customer.

The process is built upon five different Web services, the service names resemble the activity names in Figure 5. Moreover, the following operations have to be invoked to complete the *PurchaseOrder* process: `checkStockStatus`, `calculate-TotalPrice`, `isChargeable`, `chargeOrder` and `deliverOrder`. All requests used in this scenario follow the synchronous request-response message pattern. Besides using internal back-end services in the process (e.g., for checking the stock),

external services from other organizations are used for credit card data validation and payment, the *CheckCreditCard* service and the *ChargeCreditCard*.

For the evaluation, we replace the *CheckCreditCard* service with other alternative credit card services where the services are only semantically equivalent, thus a transformation using XSLT has to be applied.

6.2 Setup and Results

The evaluation was carried out using two different physical machines, connected using a 100 Mbit LAN. One machine was running ActiveBPEL to execute the process and to host VieDAME, the other machine was used to host the different Web services used in the process. The VieDAME/ActiveBPEL host was powered by an Intel Core2Duo CPU clocked at 2.66 Ghz, whereas the Web service host had an AMD64 2.0 Ghz CPU installed. Both machines were equipped with 2 GB of RAM and fast SATA disks, running Linux 2.6.22.

For running the load tests we use a commercial tool called Mercury LoadRunner [18] that allows to create real world load scenarios that can be used to simulate and predict system behavior in production environments under heavy load. The VieDAME system was stressed in three different setups: Firstly, only the `Monitor` component was enabled. For the next test run, the `Selector` component was additionally enabled, and finally, in the third step, the `Transformer` component was turned on too. Each of these four setups (including the VieDAME-disabled scenario) had to undergo a 50 minute load test, serving 50, 100 and 200 concurrent virtual users, respectively. A randomized pacing time between one and two seconds for consecutive requests was used to create a realistic high-load scenario. Additionally, an initial ramp up phase was defined to start 5, 10 or 20 users (depending on the number of virtual users) every 15 seconds to not overload the system in the very beginning.

We measured the average response times for `PurchaseOrder` requests and the number of `PurchaseOrder` transactions per second for each of the four aforementioned scenarios. The decision to use response times and transactions per second as evaluation metrics is based on the fact that they are the two most important factors of performance in production systems in the domain of process execution environments.

The results of the load test are illustrated in Figure 6. The first three upper diagrams show the response times for 50, 100 and 200 virtual users for the BPEL overall process execution. The three lower diagrams show transactions per second for the different virtual users.

Stage	50 Users	100 Users	200 Users
VieDAME Disabled	65/30	182/55	1747/56
Monitor only	73/29	298/52	2283/49
+RRobin Selector	91/29	679/43	3725/41
+XSLT Transformer	105/27	694/42	3484/38

Table 2: Response Times (ms) / Transactions per Second

The results demonstrate that the VieDAME system provides good performance for all three scenarios, whereas the 50 user test results for the VieDAME enhanced system are almost identical to the results measured in the VieDAME-disabled test. Only the results of the `Transformer` tests show that the XSLT transformations have an considerable impact on system performance. The next release of VieDAME will introduce `Transformer` *caches* to address this issue and speed up the transformation performance.

A summary for both average response times (left value) and transactions per second (right value) is provided in Table 2. Please note that our discussion and illustration of the system performance only shows the small penalty the VieDAME system adds to an existing BPEL engine. If we would consider the case where several alternative services are configured for a particularly slow original service, the overhead introduced by the base system is negligible because the overall process performance increase is much higher than the penalty of the VieDAME.

7. RELATED WORK

Ezenwoye et al. provide an approach [10] to transparently adapt BPEL processes to compensate runtime faults and to improve the process performance with respect to partner services. In their previous work [9], they presented the RobustBPEL framework that can generate an *adapt-ready* version of an existing BPEL process that is capable of monitoring the invocation of partner Web services. Upon failure, a static proxy service is invoked, which tries to find a replacement service for the failed one. In this static proxy approach, information about the replacement services is hardcoded at proxy generation time. With RobustBPEL2 [10], a new concept, called *dynamic proxies*, is introduced allowing runtime discovery of replacement services. Furthermore, RobustBPEL2 adds self-optimizing behavior to existing BPEL processes. While their work is similar to ours with respect to their aim to improve reliability and performance in the context of BPEL and Web services, they are using a proxy based approach to monitor process execution and improve process performance whereas the VieDAME system leverages a lightweight adaptation and monitoring layer based on AOP to achieve these goals. RobustBPEL2 uses UDDI to discover alternative services upon failure, but it does not incorporate selection criteria when multiple services are found, while the VieDAME system chooses the most adequate (in terms of QoS) service *in advance* to optimize process performance.

In [5], Baresi et al. present a solution for self-healing of BPEL processes. Their approach is based on *Dynamo*, a supervision framework proposed in [4] together with an AOP extension to ActiveBPEL, and a monitoring and recovery subsystem that leverages JBoss Rules. This assertion based solution provides the user with two domain specific languages (WSCoL, the *Web Service Constraint Language* and WSReL, the *Web Service Recovery Language*) to declaratively define Web service monitoring and recovery rules, respectively. WSCol and WSReL allow to create complex recovery strategies that are beyond the capabilities of VieDAME. However, their solution does not explicitly address the problem of selecting alternative services and dealing with possible interface mismatches when forwarding a request to an alternative endpoint during recovery. Our solution provides a viable way to select alternative services and to compensate these mismatches by using `Selector`s and `Transformer`s. Additionally, their recovery rules cannot be changed dynamically as they need to be compiled

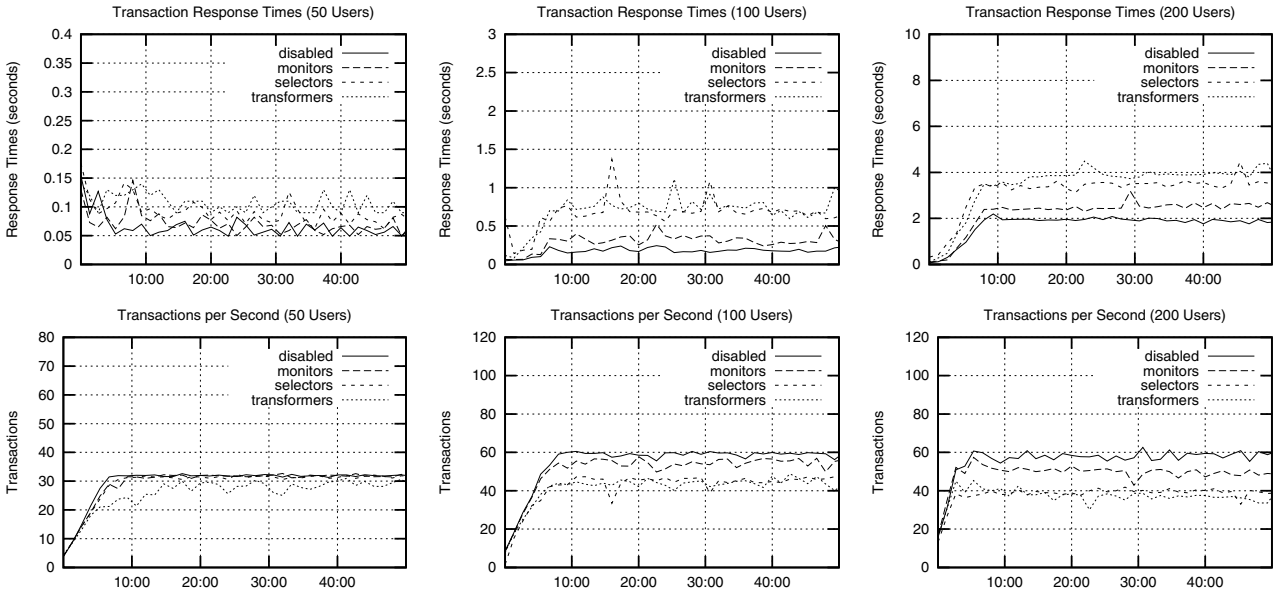

Figure 6: Response Times and Transactions per Second during VieDAME Loadtest

offline, whereas our system can configure all alternative services and selectors during runtime of the system.

AO4BPEL [7] is an aspect-oriented extension to BPEL that supports the definition of workflow aspects for BPEL processes. It is a powerful framework to define cross-cutting concerns like logging, security or transactional processing for given business processes. Aspects are defined in XML documents that in turn define one or more pointcuts and advices. An advice in terms of AO4BPEL is a BPEL activity that implements a crosscutting concern or a workflow change. AO4BPEL also allows for dynamically change the deployed process by simply activating or deactivating aspects. In contrast to Charfi's work, our approach focuses on enhancing process performance and flexibility specifically with regard to partner service interaction. The implementation of AO4BPEL is based on an aspect-aware engine approach, thus limiting its application to a particular BPEL engine (in this case IBM's BPWS4J), whereas the VieDAME system takes a non-intrusive approach to intercept interaction between the BPEL engine and its partner links.

While not strictly related to our approach, the work presented in [6] shows that the idea of interface mismatch compensation can be taken one step further to solve *behavioral* mismatch among BPEL processes. Their proposed adaptation process, given two communicating BPEL processes whose interaction may lock, builds (if possible) a BPEL process that allows the two processes to successfully interoperate. In [16], Kongdenfha et al. propose an aspect-oriented framework to provide service adaptation. Their approach uses aspect-based templates to automate the task of handling interface mismatches. Moreover, their solution is capable of handling *protocol* mismatches to compensate differences between the external specification and the implementation of arbitrary BPEL processes. However, both [6] and [16] do not aim for neither partner service monitoring nor service selection like our solution does.

8. CONCLUSIONS

In this paper we introduced an aspect-oriented extension for existing BPEL environments that allows: (i) the monitoring of existing BPEL processes according to certain QoS criteria and (ii) an adaptation strategy to replace existing partner services based on various selectors that implement different replacement strategies. The replacement services can either be syntactically or semantically equivalent to the interface used in the BPEL process. In case of a interface mismatches, a set of `Transformer`s can be specified to handle theses mismatches on a SOAP message level.

These mechanisms allow a non-intrusive adaptation of partner services within a BPEL process without any downtime of the overall system. No modifications to the process definition or the partner services are required. Therefore, VieDAME is a feasible candidate for transparently enhancing BPEL engines used in high-availability environments. Our evaluation demonstrates that the system scales very well even when the number of users is high and the overhead introduced by the monitoring, selection and transformation is minimal.

Our future work includes extending and improving our VieDAME environment by implementing more advanced `Selector` algorithms that can learn from previous execution traces to choose the best selection strategy for different time periods, thereby allowing an optimal transaction per seconds rate. Another important issue is the development of algorithms for assisting developers to create `Transformer`s between semantically equivalent replacement services. For increasing the performance, we will implement additional batch processing and storage concepts to allow even more concurrent requests. Moreover, we will extend our evaluation scenario to examine the real performance gain in process execution times when replacing particularly slow services with high-performance pendants, as these results will be essential to the decision whether the VieDAME is ready for application in production environments.

9. REFERENCES

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

[2] Apache Software Foundation. *Apache ODE*, 2007. http://ode.apache.org/ (Last accessed: Oct 21, 2007).

[3] Apache Software Foundation. *Apache Synapse*, 2007. http://ws.apache.org/synapse/ (Last accessed: Oct 21, 2007).

[4] L. Baresi and S. Guinea. Dynamo: Dynamic Monitoring of WS-BPEL Processes. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'05), Amsterdam, The Netherlands*, pages 478–483. Springer, 2005.

[5] L. Baresi, S. Guinea, and L. Pasquale. Self-healing BPEL Processes with Dynamo and the JBoss Rule Engine. In *International Workshop on Engineering of Software Services for Pervasive Environments (ESSPE '07)*, pages 11–20. ACM, 2007.

[6] A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'06), Chicago, USA*, pages 27–39. Springer, 2006.

[7] A. Charfi. *Aspect-Oriented Workflow Languages: AO4BPEL and Applications*. PhD thesis, TU Darmstadt, Fachbereich Informatik, 2007.

[8] G. Dobson. Using ws-bpel to implement software fault tolerance for web services. In *EUROMICRO '06: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 126–133, Washington, DC, USA, 2006. IEEE Computer Society.

[9] O. Ezenwoye and S. M. Sadjadi. Enabling robustness in existing BPEL processes. In *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS'06), Paphos, Cyprus*, 2006.

[10] O. Ezenwoye and S. M. Sadjadi. RobustBPEL2: Transparent Autonomization in Business Processes through Dynamic Proxies. In *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems (ISADS'07), Sedona, Arizona*, 2007.

[11] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[12] Interface21. *Spring Framework*, 2007. http://www.springframework.org (Last accessed: Oct 24, 2007).

[13] M. B. Juric, B. Mathew, and P. Sarang. *Business Process Execution Language for Web Services*. Packt Publishing, second edition, 2006.

[14] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. *Lecture Notes in Computer Science*, 2072:327–355, 2001.

[15] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[16] W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati. An Aspect-Oriented Framework for Service Adaptation. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'06), Chicago, USA*, pages 15–26. Springer, 2006.

[17] Y. Liu, A. H. Ngu, and L. Zeng. QoS Computation and Policing in Dynamic Web Service Selection. In *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*, 2004.

[18] Mercury Interactive. *LoadRunner*, 2007. www.mercury.com/us/products/loadrunner/ (Last accessed: Oct 25, 2007).

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

[20] Object Management Group – Business Process Management Initiative. *Business Process Modeling Notation (BPMN) Specification, Version 1.0*, 2006. http://www.bpmn.org/ (Last accessed: Oct. 21, 2007).

[21] Object Refinery Limited. *JFreeChart*, 2007. http://www.jfree.org (Last accessed: Oct 24, 2007).

[22] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 11, 2007.

[23] Red Hat. *Hiberante ORM*, 2007. http://www.hibernate.org (Last accessed: Oct 24, 2007).

[24] Red Hat. *JBoss Application Server*, 2007. http://www.jboss.org (Last accessed: Oct 24, 2007).

[25] Red Hat. *JBoss Seam*, 2007. http://www.jboss.org (Last accessed: Oct 24, 2007).

[26] RedHat. *JBoss AOP*, 2007. http://labs.jboss.com/jbossaop/ (Last accessed: Oct 21, 2007).

[27] RedHat. *JBoss jBPM WS-BPEL Extension*, 2007. http://docs.jboss.com/jbpm/bpel/ (Last accessed: Oct 21, 2007).

[28] M. Reichert and P. Dadam. ADAPTflex: Supporting dynamic changes of workflow without loosing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

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

[30] W3C. *Web Service Addressing (WS-Addressing)*, 2007. http://www.w3.org/Submission/ws-addressing/ (Last accessed: Oct 21, 2007).

[31] W3C. *XSL Transformations (XSLT) Version 2.0*, 2007. http://www.w3.org/TR/xslt20/ (Last accessed: Jan 23, 2007).