Self-tuning BPEL Processes

Adina D. Mosincat
Faculty of Informatics, University of Lugano
Lugano, Switzerland
adina.diana.mosincat@lu.unisi.ch

Walter Binder
Faculty of Informatics, University of Lugano
Lugano, Switzerland
walter.binder@unisi.ch

ABSTRACT
The performance of BPEL processes depends on the composing web services. Monitoring web service performance and adapting to changes in service performance are essential for creating self-tuning BPEL processes. Unfortunately, BPEL supports neither performance monitoring nor dynamic adaptation mechanisms. In this paper, we present a monitoring and rebinding infrastructure for BPEL that transparently enhances existing BPEL processes with self-tuning behavior.

Categories and Subject Descriptors

General Terms
Performance

Keywords
BPEL processes, dynamic service selection and binding, performance monitoring, self-optimization

1. THE BPEL ADAPTABILITY PROBLEM

Web service compositions are usually defined as processes in the Business Process Execution Language (BPEL1). A process defines business logic by modeling message exchange sequences in an executable manner, invoking service functionalities described in WSDL2. Processes are executed by BPEL engines, which create a process instance when one of the receive activities (a start activity) in the process is triggered, and end the instance after completion of the corresponding reply activity. The process instance interacts with services (partner links) through invoke activities.

BPEL processes are a means of building new applications leveraging existing services. However, the web as a highly dynamic environment brings with it challenges such as reliability and performance. The ability of a process to adapt to change is vital [2]. Furthermore, a service must provide a continuously good performance level to be competitive.

BPEL faces two shortcomings: its inherent static nature and the lack of any means of monitoring a running process. While it is possible to use the dynamic partner link assignment to change a partner link that represents a service in a process at runtime, this involves coupling the dynamic binding with process business logic and does not allow adding new available services after the process has been deployed. The issue of monitoring is crucial when performance is considered since it leaves the process unaware of, and thus dependent on the changes in the environment.

We present an infrastructure that overcomes these shortcomings, providing a means to select the service to bind at runtime and to monitor the execution of the bound service, learning about the changes in the environment and reacting on the collected information. Our selection algorithm handles the balance between service load and service performance, leveraging the information provided by our monitoring mechanism. Our approach achieves compatibility with standard BPEL engines.

Our goal is to assure the users experience a good performing service in highly loaded environments, where a large number of requests must be fulfilled concurrently. In our service selection we use a load-balancing algorithm that distributes the requests between available services based on their monitored performance. The work presented in this paper builds on our generic architecture described in [1]. As original scientific contributions, we introduce a novel mechanism that enhances the process with self-optimization capabilities.

2. A TRANSPARENT APPROACH TO SELF-ADAPTIVE PROCESSES

Figure 1 shows the architecture of our system (Bind System). Functionally equivalent services that can substitute for each other are grouped into service types. All services of the same service type must offer the same interface.

The Service Monitor collects information on service response times from the dynamic proxies and provides statistics to other parts of the system.

The Service Ranker ranks services according to their monitored performance. Services with fast response times will have a higher probability to be selected. The ranking is updated periodically.

The Worked arrows in the figure show the process transformation. At deployment time the process is transformed by the Transformation Tool, which replaces every service used in the original process with the service type provided by the
and receives back the concrete service bindings.

Bind Manager types the process instance may invoke to the T o this end, the process instance sends the list of all servic e

( Service Manager and the resulting transformed process is

or it may refer to a dynamic proxy in Figure 1). In the former case, the invocations of the service by the process instance will not be monitored (Unmonitored invocation in Figure 1), whereas in the latter case, the service invocations will be monitored by the dynamic proxy (Monitored invocation in Figure 1).

2.1 Service Ranking and Selection

In order to improve process performance, it is of paramount importance to select the right service to bind. The target of the selection algorithm is to help the system continuously converge to and maintain good performance. The selection needs to assure that the service which best fulfills this purpose is selected. While a na"ıve solution would be to always select the service with the best performance, in reality this strategy would put the performance of the system in jeopardy by excessively overloading the best available service. Therefore, the algorithm must find an equilibrium between service performance and service load.

Services are ranked according to their performance. For every service, the Service Ranker computes a selection probability based on the monitored service performance. Assuming $n$ services of the same service type, the probability $sp_i$ of a service $i (1 \leq i \leq n)$ to be selected is computed as follows: $sp_i = \left[\frac{1}{n-1}\right] + \left[1 - \frac{rt_i}{\sum_{j=1}^{n} rt_j}\right]$, where $rt_i$ represents the monitored average response time of service $i$. For all $i$, $0 < sp_i \leq 1$, and $\sum_{i=1}^{n} sp_i = 1$. The service probabilities are updated after a configurable time interval based on the new monitoring information.

The Service Selector executes the following algorithm to select the service to bind:

1. Let $r$ be the next pseudo-random value from a uniform distribution, $0 \leq r \leq 1$.

2. Select the service $m$, where $m = \min\{i : 1 \leq i \leq n\} \land (r \leq \sum_{j=1}^{i} sp_j\}$.

Service selection in our probabilistic approach is related to lottery scheduling [3].

2.2 Evaluation and Conclusion

We conducted a preliminary evaluation for a group of 10 equivalent services, each performing according to a discrete time Markov-chain model with 10 states, each state corresponding to a distinct service response time between 100ms and 1000ms. The probability for keeping the current state is 1/2, the probability of each state change is 1/18, and the timeslot$^3$ is 60 seconds. The service ranking is updated every 15 seconds and all service invocations are monitored. We base our evaluation on a process with 3 service invocations. As reference, we use an equivalent process that implements a hard-coded round-robin service selection mechanism (without Bind System). Figure 2 shows the system throughput measured for each 200 process instances. On average, the Bind System achieves a 40% increase in system throughput.

Summing up, self-optimization is crucial in the lifetime of a process because it allows the process to control its performance and gain independence over the evolution of the composing services. Our infrastructure enhances the process with self-adaptive and self-optimization behavior. The contributions of this paper are (1) a transparent approach to automatic adaptation and self-optimization of BPEL processes, and (2) a probabilistic service selection algorithm that balances the requests between services according to their recent performance.

3. REFERENCES


$^3$A time slot is the moment in time when a decision regarding the next state is randomly made based on the current state and the state change probabilities.