Chapter 12
Event Processing in Web Service Runtime Environments

Anton Michlmayr
Vienna University of Technology, Austria

Philipp Leitner
Vienna University of Technology, Austria

Florian Rosenberg
CSIRO ICT Centre, Canberra, Australia

Schahram Dustdar
Vienna University of Technology, Austria

ABSTRACT

Service-oriented Architectures (SOA) and Web services have received a lot of attention from both industry and academia. Services as the core entities of every SOA are changing regularly based on various reasons. This poses a clear problem in distributed environments since service providers and consumers are generally loosely coupled. Using the publish/subscribe style of communication service consumers can be notified when such changes occur. In this chapter, we present an approach that leverages event processing mechanisms for Web service runtime environments based on a rich event model and different event visibilities. Our approach covers the full service lifecycle, including runtime information concerning service discovery and service invocation, as well as Quality of Service attributes. Furthermore, besides subscribing to events of interest, users can also search in historical event data. We show how this event notification support was integrated into our service runtime environment VRESCo and give some usage examples in an application context.

INTRODUCTION

Following the Service-oriented Architecture (SOA) paradigm shown in Figure 1, service providers register services and corresponding descriptions in registries. Service consumers can then find services in the registry, bind to the services that best fit their needs, and finally execute them. Web services (Wegerwaraana, Curbera, Leymann, Storey, & Ferguson, 2005) are one widely adopted realization of SOA that build upon the main stan-
Event Processing in Web Service Runtime Environments

Standards SOAP (communication protocol), WSDL (service description) and UDDI (service registry). Over the years, a complete Web service stack has emerged that provides rich support for multiple higher level functionalities (e.g., business process execution, transactions, metadata exchange etc.).

Practice, however, has revealed some problems of the SOA paradigm in general and Web services in particular. The idea of public registries did not succeed which is highlighted by the fact that Microsoft, SAP and IBM have shut down their public registries in the end of 2006. Moreover, there are still a number of open issues in SOA research and practice (Papazoglou, Traverso, Dustdar, & Leymann, 2007), such as dynamic binding and invocation, dynamic service composition, and service metadata.

One reason for these issues stems from the fact that service interfaces, service metadata and Quality of Service (QoS) attributes change regularly. Furthermore, new services are published, existing ones might be modified, and old services are finally deleted from the registry. This is problematic since service providers and consumers are usually loosely coupled in SOA. Thus, service consumers are not aware of such changes and, as a result, might not be able to access changed services any more. In this regard, the lack of appropriate event notification mechanisms limits flexibility because service consumers cannot automatically react to service and environment changes.

The current service registry standards UDDI (OASIS International Standards Consortium, 2005a) and ebXML (OASIS International Standards Consortium, 2005b) introduce basic support for event notifications. Both standards have in common that users are enabled to track newly created, updated and deleted entries in the registry. However, additional runtime information concerning service binding and invocation as well as QoS attributes are not taken into consideration by these approaches.

We argue that receiving notifications about such runtime information is equally important and should, therefore, be provided by SOA runtime environments. Furthermore, complex event processing mechanisms supporting event patterns, and search in historical event data are needed for keeping track of vast numbers of events. In this chapter, we focus on such runtime event notification support. Our contribution is threefold: firstly, we present the background of this work and describe the motivation based on a case study from the telecommunications domain. Secondly, we introduce the VRESCo runtime environment (Michlmayr, Rosenberg, Platzer, Treiber, & Dustdar, 2007) and describe its notification support in detail. This includes event types, participants, ranking, correlation, subscription, and notification mechanisms, as well as event persistence, event search, and event visibility. Finally, we show some usage examples and point to further application scenarios enabled by our work.

BACKGROUND

This section consists of two main parts. In the first part, we summarize several research approaches that are related to our work. In the second part, we briefly introduce the open source event processing engine Esper which we use as technical background for our prototype.