Publish/Subscribe in the VRESCo SOA Runtime

Anton Michlmayr, Philipp Leitner, Florian Rosenberg, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Wien, Austria
lastname@infosys.tuwien.ac.at

ABSTRACT

Event-based systems and the publish/subscribe style are widely used to notify subscribers when certain events of interest occur. In the context of Service-oriented Architecture (SOA) and Web services, event notifications can be used to address one issue inherent to the SOA paradigm: Services and Quality of Service attributes are changing regularly but service consumers cannot react automatically. In current service registry standards, notifications are mainly used to inform about changes in the registry data, which does not include service runtime information. In this paper, we present a SOA runtime environment that leverages event processing for Web services to support the full service lifecycle, including runtime information concerning service discovery and invocation, as well as Quality of Service attributes.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; C.2.4 [**Computer-Communication Networks**]: Distributed Systems — Distributed Applications

Keywords

Service-oriented Architecture, Publish/Subscribe, Event Processing

1. INTRODUCTION

Following the Service-oriented architecture (SOA) paradigm, service providers register services and corresponding descriptions in registries. Service consumers can then find services in a registry, bind to the services which best fit their needs, and finally execute them. Web services are one widely adopted realization of SOA and build upon the main standards SOAP, WSDL and UDDI. Over the years, a complete Web service stack has emerged that provides rich support for multiple higher level functionality (e.g., business process execution, transactions, metadata exchange, etc.).

DEBS '08, July 1-4, 2008, Rome, Italy

One of the issues inherent to the SOA paradigm is represented by the fact that services and associated metadata and Quality of Service (QoS) attributes change regularly. However, service consumers are not aware of these changes. In this regard, the lack of appropriate event notification mechanisms limits flexibility because service consumers cannot automatically react to service and environment changes.

Current service registry standards such as UDDI [9] and ebXML [8] introduce limited support for event notifications. Both standards have in common that users are enabled to track 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 standards. 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 paper, we present an approach that addresses such runtime event notification support which was integrated into the VRESCo runtime [2, 7]. The remainder of this paper is organized as follows. Section 2 briefly mentions related work in this area. Section 3 gives an architectural overview of the VRESCo event notification engine whereas Section 4 demonstrates how this software is used in practice.

2. RELATED WORK

Event-based systems in general, and the publish/subscribe style in particular have been the focus of research within the last years. This research has led to different event-based architecture definition languages (e.g., Rapide [4]) and QoS-aware event dissemination middleware prototypes [5].

Cugola and Di Nitto [1] give a detailed overview of research approaches combining SOA and publish/subscribe. The most popular examples are WS-Notification [10] and WS-Eventing [12]. While WS-Eventing uses content-based publish/subscribe, WS-Notification provides topics according to WS-Topics as a means to classify events.

There are several approaches that address search in historical events [3, 11]. Rozsnyai et al. [11] introduce the *Event Cloud* system which aims at searching for business events. Their approach uses indexing and correlation of events by using different ranking algorithms. In contrast to our approach, the focus is on building an efficient index for searching in vast numbers of events whereas subscribing to events and getting notified about their occurrence is not addressed.

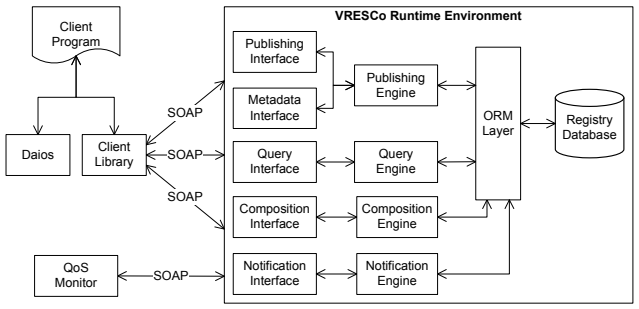

Figure 1: VRESCo Overview

Li et al. [3] present a data access method which is integrated into the distributed content-based publish/subscribe system PADRES. The system enables to subscribe to events published in both the future and the past. In contrast to our work, the focus is on building a large-scale distributed publish/subscribe system that provides routing of subscriptions and queries.

3. SYSTEM DESCRIPTION

The approach presented in this paper was implemented as part of the VRESCo (Vienna Runtime Environment for Service-Oriented Computing) project [2, 7].

3.1 Overview

The basic architecture of VRESCo is shown in Figure 1. To be platform-independent, the VRESCo services are provided as Web services which can be accessed either directly using the SOAP protocol, or via the client library that provides a simple API for accessing these services.

Services and associated metadata are stored in the registry database that is accessed using the object-relational mapping (ORM) layer. The services are published and found in the registry using the publishing and querying engine, respectively. The VRESCo runtime uses a QoS monitor which continuously monitors the QoS values of services and keeps the QoS information up to date. Furthermore, the composition engine aims at providing support for QoS-aware service composition which is part of our ongoing work. Finally, the event notification engine is responsible for notifying subscribers when events of interest occur.

To carry out Web service invocations the Daios framework integrated in the VRESCo client library decouples clients from the services to be invoked by abstracting from service implementation issues such as encoding styles, operations or endpoints. Therefore, clients only need to know the WSDL interface of the target service, and the corresponding input message; all other implementation details are handled transparently.

Web services evolve over time, which raises the need to maintain multiple service revisions concurrently. VRESCo supports service versioning by introducing the notion of service revision graphs that define successor-predecessor relationships between different revisions of a service and support multiple branches. Revision tags are thereby used to distinguish the different service revisions. Service consumers are then enabled to decide which service to use (e.g., invoke specific revisions, or always invoke the latest revision).

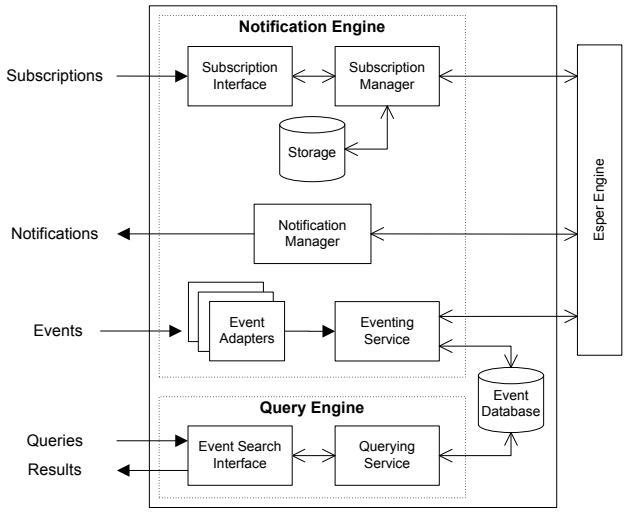

Figure 2: VRESCo Eventing Architecture

3.2 VRESCo Eventing

This section gives a high-level overview of the VRESCo notification support while the details are described in a companion paper [6]. The basic idea can be summarized as follows: Notifications are published within the runtime if certain events occur (e.g., service is added, user is deleted, etc.). In contrast to current Web service registries, this also includes events concerning service binding and invocation, changing QoS attributes, and runtime information. Clients can then subscribe to get notified about the occurrence of these events using different notification mechanisms (e.g., email, listener Web service, etc.)

Figure 2 depicts the architecture of the notification engine which is part of the VRESCo runtime in Figure 1 and, therefore, also implemented in C# on the .NET platform. The event processing functionality is based on NEsper, which is a port of the event processing engine Esper[1].

Event Types and Representation. The events supported by VRESCo are represented by C# classes that form a type hierarchy where events inherit the properties of the parent type. All events inherit from the base type *VRESCoEvent* which contains a unique event sequence number and a timestamp measured during event publication.

The VRESCo runtime provides several event types. First of all, service management events are triggered when services or service revisions and their associated metadata or QoS values change. Other event types include runtime information concerning binding and invocation (e.g., *ServiceInvokedEvent*), querying information (e.g., *RegistryQueriedEvent*) and user information (e.g., *UserAddedEvent*).

Event Publication. Within the notification engine, events are published using the eventing service. We distinguish between *internal events* which are produced within the SOA runtime and *external events* which are published from components outside the runtime. Internal events are directly produced by the corresponding VRESCo services (e.g., service management events are fired by the publishing service

[1] `http://esper.codehaus.org`

while querying events are fired by the querying service). In contrast to this, external events related to binding and invocation are produced by the service proxies located in the client library, whereas QoS events are regularly fired by the QoS montitor to publish the current QoS values. External events are fired using the notification interface where event adapters are used to transform incoming events into the internal event format which can be processed efficiently. The eventing service then forwards both internal and external events to the event persistence component that is responsible for storing events in the event database by using the ORM layer. Finally, the eventing service feeds all incoming events into the Esper engine.

Subscriptions. Similar to event producers, we distinguish between *internal* and *external subscribers*. Internal subscribers reside within the VRESCo runtime and register listeners at the Esper engine which are invoked when subscriptions match incoming events. External subscribers outside the runtime are notified depending on the notification delivery mechanism defined in the subscription request.

The external subscription interface is used for subscribing to events of interest according to the methods proposed in the WS-Eventing specification. The subscription manager is responsible for managing subscriptions which are put into the subscription storage. In addition, subscriptions are translated for further processing. This is done by converting the WS-Eventing subscriptions into Esper listeners which are attached to the Esper engine.

In both cases, the Esper Query Language (EQL) [13] is used as subscription language. The structure of EQL is similar to SQL but EQL is formulated on event streams whereas SQL uses database tables. Therefore, EQL also supports aggregate functions, grouping functions, ordering structures, and joining of event streams. Furthermore, EQL provides a powerful mechanism to integrate temporal relations of events using sliding event windows, supports statistical functions over event properties and enables to define event patterns.

Event Notifications. The Esper engine performs the actual event processing and is, therefore, responsible for matching incoming events received from the eventing service to listeners attached by the subscription manager. On a successful match, the registered listener informs the notification manager that is responsible for notifying interested subscribers. Depending on the listener type, the notification manager knows which notification type to use (e.g., email, listener Web service).

Event Search. Event notifications are often used when subscribers want to quickly react on state changes. Additionally, in many situations it is also important to search in historical event data. To support such functionality, the VRESCo notification engine persists all events in an event database. The events can be queried using the event search interface which is part of the querying interface. The querying service then returns a list of events that match the given query. Since we use NHibernate[2] for implementing the ORM layer, our query language is based on the Hibernate Query Language (HQL).

[2]`http://www.nhibernate.org`

4. SOFTWARE DEMONSTRATION

In this section we demonstrate the VRESCo runtime using a case study from the telecommunications domain. Consider a telecommunications company (TELCO) hosts several services, and consumes services from competitor TELCOs (e.g., number porting service) and other partners (e.g., credit card payment service, shipping service, etc.).

Figure 3 shows this case study in the VRESCo Runtime Manager GUI. In VRESCo, services are grouped into categories of services that perform the same task (e.g., PhoneNumberPorting). These categories and services are illustrated in the left part of the GUI which also provides a search interface for querying services within the registry database. The service revision graph of the selected service is depicted in the middle, showing Id and tags (e.g., INITIAL, STABLE, etc.) of all service revisions. The initial revision is always placed on the top of the graph and the edges define the predecessor-successor relationship. The details of the selected service revision are shown in the right part including revision tags, URL of the WSDL document, current QoS parameters, and all events related to this revision.

```
1  IVRESCoSubscriber subscriber =
2    VRESCoClientFactory.CreateSubscriber(
3      "rome.vitalab.tuwien.ac.at", 11111);
4
5  // subscribe using email notifications
6  Identifier sid = subscriber.SubscribePerEmail(
7   "select * from RevisionPublishedEvent " +
8     "where Service.Id = 11",
9   "anton@infosys.tuwien.ac.at",
10  60*10
11 );
12
13 // subscribe using Web service notifications
14 sid = subscriber.SubscribePerWS(
15  "select * from QoSEvent "+
16  "where Revision.Id = 17 "+
17  "and ResponseTime > 500",
18  "net.tcp://localhost:8005/SubscriptionEndTo",
19  "net.tcp://localhost:8006/OnVRESCoEvents",
20  24*60*60
21 );
22
23 // use sliding window and statistics
24 sid = subscriber.SubscribePerEmail(
25  "select * from QoSEvent(Revision.Id=47) "+
26  "win:time(6 hours).stat:uni('QoS.Latency') "+
27  "where average > 200",
28  "anton@infosys.tuwien.ac.at",
29  30*24*60*60
30 );
```

Listing 1: Subscription examples

The TELCO case study has several scenarios where notifications are useful. Listing 1 gives some examples and illustrates how subscriptions are defined using the VRESCo client library. First of all, a subscriber proxy is generated by the client factory which takes server name and port number of the VRESCo subscription manager service (Line 1–3).

The first example shown in Line 6–11 represents an email notification. The first parameter defines the subscription in EQL which in this example means that notifications should be sent every time a new revision of service 11 is published (Line 7–8). The second paramter specifies the email address the notification manager should use for the notifications, while the third parameter defines the duration of the subscription in seconds. The subscription manager returns a unique subscription identifier *sid* which can be used to unsubscribe or renew the subscription.

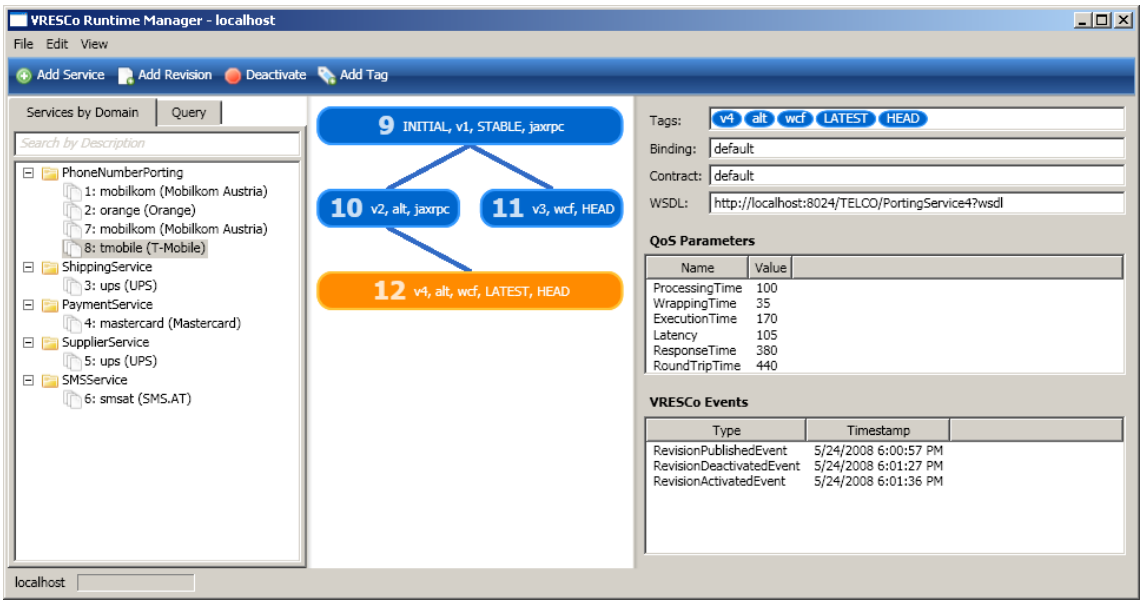

Figure 3: VRESCo Runtime Manager

The second example declares interest in *QoSEvents* where the *ResponseTime* of revision 17 is greater than 500 ms (Line 14–21) since this might violate the Service Level Agreement. This time the notification should be sent using Web service notifications following the WS-Eventing specification. The second parameter defines where the *subscriptionEnd* messages should be sent, while the third parameter specifies the destination of the actual notification messages.

Finally, the third example shows the use of the sliding window operator and statistical functions over multiple events. More precisely, it defines that notifications should be sent per email if the average *Latency* of *QoSEvents* concerning service revision 47 that occurred within the last six hours is greater than 200 ms (Line 24–30).

The performance evaluation of our first prototype has shown that the throughput of internal events is several hundreds of events per second (depending on the number of subscribers), while the throughput of external events is between 50 and 200 (depending on the binding). A detailed evaluation of our work, including the expressiveness of the subscription language, performance evaluation and possible application scenarios can be found in [6].

5. REFERENCES

[1] G. Cugola and E. Di Nitto. On adopting content-based routing in service-oriented architectures. *Information and Software Technology*, 50(1–2):22–35, Jan. 2008.

[2] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. End-to-End Versioning Support for Web Services. In *Proceedings of the International Conference on Services Computing (SCC 2008)*. IEEE Computer Society, July 2008.

[3] G. Li, A. Cheung, S. Hou, S. Hu, V. Muthusamy, R. Sherafat, A. Wun, H.-A. Jacobsen, and S. Manovski. Historic Data Access in Publish/Subscribe. In *Proceedings of the Inaugural International Conference on Distributed Event-Based Systems (DEBS'07)*, pages 80–84. ACM, 2007.

[4] D. C. Luckham and J. Vera. An Event-Based Architecture Definition Language. *IEEE Transactions on Software Engineering*, 21(9):717–734, 1995.

[5] S. P. Mahambre, M. K. S.D, and U. Bellur. A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware. *IEEE Internet Computing*, 11(4):35–44, 2007.

[6] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Advanced Event Processing and Notifications in Service Runtime Environments. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*. ACM, 2008.

[7] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar. Towards Recovering the Broken SOA Trianlge – A Software Engineering Perspective. In *Proceedings of the Second International Workshop on Service Oriented Software Engineering (IW-SOSWE'07)*, pages 22–28, Sept. 2007.

[8] OASIS International Standards Consortium. *ebXML Registry Services and Protocols*, 2005.

[9] OASIS International Standards Consortium. *Universal Description, Discovery and Integration (UDDI)*, 2005.

[10] OASIS International Standards Consortium. *Web Services Notification (WS-Notification)*, 2006.

[11] S. Rozsnyai, R. Vecera, J. Schiefer, and A. Schatten. Event Cloud - Searching for Correlated Business Events. In *Proceedings of the 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, pages 409–420. IEEE Computer Society, 2007.

[12] World Wide Web Consortium. *Web Services Eventing (WS-Eventing)*, 2006.

[13] Esper Reference Documentation, 2008. `http://esper.codehaus.org/`.