EVENT PROCESSING IN THE CLOUD ENVIRONMENT WITH WELL DEFINED SEMANTICS

Marc Schaaf, Arne Koschel
University of Applied Sciences and Arts, Ricklinger Stadtweg 120, 30459 Hannover, Germany

Stella Gatziu Grivas
University of Applied Sciences Northwestern Switzerland, Riggenbachstr. 16, 4600 Olten, Switzerland

Keywords: Active DBMS, Activity service, Event condition action (ECA) rules, Cloud computing, Enterprise architecture frameworks, Event driven architectures, Complex event processing.

Abstract: The paper presents the OM4SPACE project, which aims to provide an integration of active functionality into cloud environments to enable applications to further benefit from the agility of this new environment. In particular, we propose an activity service that incorporates well defined interfaces and the clear semantic of Active Database Management System (Active DBMS) style event processing combined with the concepts of modern Event Driven Architectures and the approach of Complex Event Processing. This gives the potential to provide active functionality across the boundaries of one Cloud and to decouple the usage of active functionality from the concrete cloud provider. In this paper we present our initial prototypic implementation of this activity service.

1 INTRODUCTION

Cloud computing (Armbrust et al., 2009) is a 'trendy new kid on the block' as many recent activities in research and industry show. Companies and open source players almost constantly announce new features for their cloud platforms.

Event-based active mechanisms are an important feature for the cloud, be it just in form of messaging or in more elaborated event- or rule-driven behavior (Rozsnayi et al., 2007). Although the necessity of supporting active behavior is clear, the open issue is the lack of a well-defined semantic.

The overcome of this drawback is the contribution of our work. For our OM4SPACE (Open Mediation 4 SOA and P2P Based Active Cloud Components) project, we proposed an activity service for cloud computing (Schaaf et al., 2010). The activity service adopts its semantics from the well-proven and clear semantics of Active Database Management System (Active DBMS) style (Widom and Ceri, 1996; Paton, 1999) event-condition-action (ECA) rules. This semantic is modified and extended for the cloud. The design of the activity service is based on proven principles, patterns, and technologies from service oriented architectures (SOA). Eventually, we will deliver an activity service with a precisely defined Active DBMS style ECA rule and execution model for the cloud.

Event-Driven mechanisms have been extensively investigated in the 1990s starting with the approach of active databases supporting ECA-rules. One step beyond go approaches concerning ECA rule processing in distributed environments. In (Koschel and Lockemann, 1998), active functionality was extended into an ECA rule service for CORBA-based distributed environments. Actually, this approach is one initial step in our direction. Recently Event Driven Architecture (EDA) have been proposed as an architecture style for the creation of distributed systems with the aim to create agile applications. Event Driven Architectures are tightly linked to Complex Event Processing (CEP), which is an emerging enabling technology to apply context-aware knowledge from and against large amounts of event data in near real-time (Luckham, 2001).

With the activity service we will extend the set of available cloud services, which can be used by the developers to rapidly create or adapt their applications for the cloud. Moreover, it will bridge the gaps be-

Schaaf M., Koschel A. and Gatziu Grivas S..
EVENT PROCESSING IN THE CLOUD ENVIRONMENT WITH WELL DEFINED SEMANTICS.
DOI: 10.5220/0003393001760179
In *Proceedings of the 1st International Conference on Cloud Computing and Services Science* (CLOSER-2011), pages 176-179
ISBN: 978-989-8425-52-2

tween several cloud specific messaging or notification mechanisms by providing an abstraction across the vendor specific API's. This will enable developers of event-driven cloud applications to utilize a powerful middleware for active functionality without the risk of a vendor lock-in. Application areas include, for example, cloud vendor independent complex event processing scenarios or the processing of events, which occur in logistics or finance applications.

To achieve our goals we need to deal with several challenges including: *Well-defined semantics for the activity service, architectural issues, highly dynamic runtime environments, highly heterogeneous system environments, vendor specific event or message communication mechanisms* etc.

While (Schaaf et al., 2010) sketched the idea and key high-level architectural concepts of the activity service, this paper summarizes them only briefly. Its core contribution is our meanwhile developed initial prototypic implementation of the activity service.

The remainder of this paper starts with an overview description of the activity service with its components. Afterwards we describe our prototypic implementation followed by an outline of the next steps for the implementation. Eventually we conclude with a summary and an outlook on the next steps for the whole project.

2 AN ACTIVITY SERVICE FOR THE CLOUD

The aim of our work is to bring active functionality into the cloud in the form of a simple and easy to use service, which features a clear semantic. Instead of defining the semantic for the service from scratch, we base our approach on the proven and well-defined semantics of active database systems and aim to adapt them to the new, highly dynamic and massively distributed environment. The architectural and semantical base of our work uses the unbundling approach for active functionality from Active DBMS concepts (Gatziu et al., 1998) for distributed and heterogeneous environments. Based on these results and in order to achieve a high flexibility, we base our activity service on two components: the event service and the rule execution service (Figure 1).

The communication between the components is realized based on the concept of a SOA thus utilizing services with well-specified interfaces with clear semantics. Thereby the concrete implementation of the different components is interchangeable.

We consider the interaction of applications located in one cloud with applications located in other clouds as a common use case. Thus the event producers and consumers of the activity service are not limited to the components in the cloud where the activity service is located.

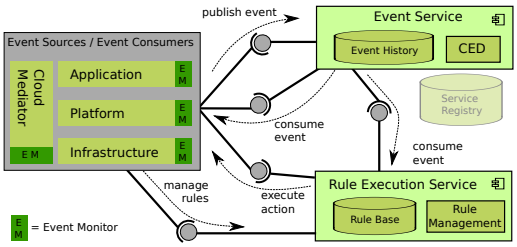

Figure 1: The components of the activity service and their interactions.

2.1 Activity Service: Components

Figure 1 illustrates the different components of the activity service and their interactions. Their functionality is explained in more detail in (Schaaf et al., 2010) and thus only summarized in this section.

The event service component provides the means to be informed about the occurrence of events from different event sources, to pre-process those events and to deliver them to appropriate event consumers. Therefore it provides a service, which can be used by event producers to send their events to the event service via a *sendEvent* method. For each incoming event, the event service determines if there are event consumers that are interested in this particular event and delivers the event to them.

In addition the incoming events are stored into an event history to support the monitoring of complex/composite events. A complex event detector (CED) evaluates the events and derives new complex events, which are fed back into the processing mechanism. Consequently they are handled again as if they were incoming events.

To receive events from the event service, an event consumer has to implement an appropriate event handler service, which needs to be published to the service registry. The event service discovers those services through the service registry. To extend the mechanism of the event service, we utilize the concept of an event monitor which capsules data sources that are not able to actively notify the event service (Schaaf et al., 2010).

The rule execution service receives events from the event service to evaluate them against sophisticated ECA rules. Therefore it acts as an event consumer of the event service by registering an event handler service. The rules result in the execution of action handlers. Such an action handler needs to be imple-

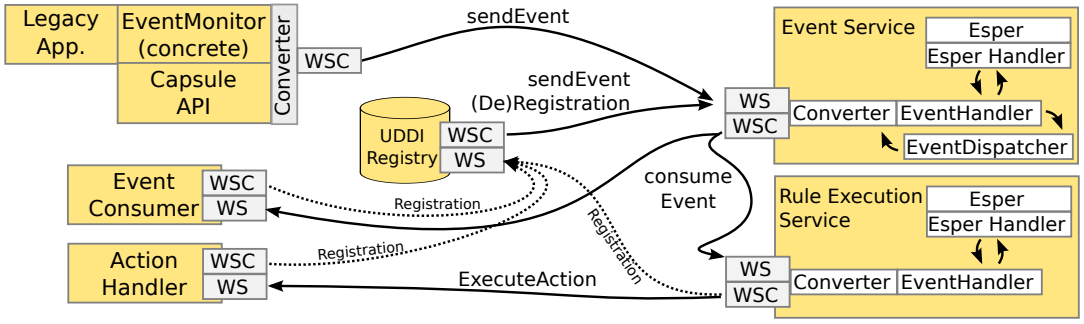

Figure 2: The Technical Architecture of the Activity Service

mented by each of the components that are intended to be called from within rules.

2.2 Putting the Activity Service into Context

Most of the cloud architectures that were reviewed focus on just technical stacks that lack interoperability with other providers. In the area of event processing mechanisms, many existing offerings include the means for message passing like for example Amazon with their Simple Queuing Service. However these offerings also lack common interoperable and well-defined interfaces that prevent a vendor lock-in and enable applications to use active mechanisms across the boundaries of a single provider. With our activity service we fill this gap by providing a well-defined service with a clear semantic, which is independent of the concrete cloud provider. Furthermore we will extend the activity service across the boundaries of one cloud so that it not only provides a general interface but also provides the means to communicate with application parts located in different clouds. The long term objective of our project is to *extend the platform for cloud based applications* so that it provides the means to create applications that can benefit from the dynamics of cloud environments by utilizing active mechanisms in a provider independent way.

2.3 Prototypic Implementation

Based on our conceptual work a prototypical implementation of the activity service was developed. Its technical system architecture is illustrated in Figure 2 and described in the following subsections.

2.3.1 Technical Architecture

Technically the activity service uses Web Services to receive events from event producers. After the event processing, it delivers the events to Web Services, which are provided by the event consumers. The event processing in the event service itself is currently based on the complex event processing engine Esper (esper.codehaus.org). However, all details regarding the integration of Esper are hidden from the event producers and consumers. Thus the processing engine could be replaced with another rule processing engine such as CLIPS or Drools .

As a starting point we work with an event type which basically consists of the event type name, the event source name and a set of timestamps to record the event occurrence, detection and processing times. The mentioned fields provide the header information of the event, the body of the event contains a flexible set of key value pairs, which are specific for each event type and can be used as needed.

The discovery of event consumers is realized as a facade in front of an UDDI repository, which informs the event service of new event consumers by sending an event whenever a service appears or disappears. The reason for the introduction of a facade in front of the repository is the lack of a notification mechanism in UDDI to inform a subscriber about newly added services

For the notification of the event service, the UDDI facade uses the active mechanisms that are provided by the event service in the first place. Thus the facade acts as an event producer and informs the event service of registered or deregistered event hander services by raising an event. The event contains the WSDL of the event handler service. Furthermore it contains a filter string to specify events, which the handler is interested in. The event service receives the event and thus gets all the required information to contact the event handler if a matching event has to be delivered. Due to this mechanism, the event service does not require any knowledge about the UDDI registry.

As the implementation provides all required components and functionalities, a complete roundtrip from the event producer through the (complex) event processing to the event consumers including the automatic discovery of event handlers is possible.

One of the next steps for the implementation is to move from the specialized Web Service interfaces to a dedicated communications layer that is outlined in the next section. Moreover we are working on the realization of the rule execution service to support the decoupled processing of more elaborated and time consuming rules that result in the execution of an external action instead of the generation of new events.

2.3.2 Communication Infrastructure Abstraction

Our previous descriptions, as well as our current implementation, are tightly linked to Web services. However our general concept is not limited to one communication mechanism. Instead we will extend our implementation with a layer of abstraction from the actual communication mechanism to allow the activity service to interact with cloud provider specific solutions like the messaging facilities provided as part of Amazon's cloud offerings.

Moreover the activity service will provide the possibility to use multiple communication mechanisms in parallel. This provides the capability to mediate between different mechanisms in order to combine event based applications, which use otherwise incompatible (provider specific) communication mechanisms.

For this purpose we are currently working on an abstraction layer, which hides the different provider specific methods from the activity service and the event produces and consumers. This layer will then be implemented for each specific communication mechanism like Amazons Simple Queuing Service, a JMS client or a WS-Eventing based system and can thus provide an optimal mapping of the event structure to the transport mechanism.

Discovery. With regard to the integration of an abstraction layer from the transport mechanisms, a similar technology independent approach is required for the discovery.

Our approach for the discovery of new event handlers is based on events and thus decoupled from the actual discovery component. For the current implementation the technology specific parts of the discovery are handled by the mentioned UDDI facade. This facade creates events with a technology independent description of the requested event types, which is needed by the event service itself and a technology specific description of the endpoint. This mechanism allows the event service to discover event handlers that use any of the communication mechanisms that are supported by the abstraction layer.

3 CONCLUSIONS

With the OM4SPACE project we extend the cloud platform with active functionality so that cloud based applications can easily integrate active mechanisms and therefore react to events just in time. With this we will assist cloud applications to become even more dynamic and to reach their full potential. Furthermore we provide this active behavior with the clear and well-proven semantic of Active DBMS and well-defined interfaces across the border of a single cloud and without the limitations given by vendor specific messaging mechanisms.

After the definition of the architecture and a prototypic realization of the activity service, we now focus on the extension of the concept across the border of a single cloud and the adaption to the new highly dynamic environment. Furthermore we focus on the evaluation of our concept in the scope of real live application scenarios.

REFERENCES

Armbrust, M. et al. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Technical report, EECS Department, University of California, Berkeley.

Gatziu, S., Koschel, A., et al. (1998). Unbundling active functionality. *ACM SIGMOD Rec.*, 27(1):35–40.

Koschel, A. and Lockemann, P. C. (1998). Distributed events in active database systems: letting the genie out of the bottle. *Data Knowl. Eng.*, 25(1-2):11–28.

Luckham, D. C. (2001). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Paton, N. W., editor (1999). *Active Rules for Databases*. Springer, New York.

Rozsnayi, S. et al. (2007). Event Cloud - Searching for Correlated Business Events. *9th IEEE International Conference on E-Commerce Technology*.

Schaaf, M., Koschel, A., Grivas, S. G., and Astrova, I. (2010). An active dbms style activity service for cloud environments. Lisbon, Portugal. XPS (Xpert Publishing Services).

Widom, J. and Ceri, S., editors (1996). *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann Publishers, San Francisco, CA, U.S.A.