An Active DBMS Style Activity Service for Cloud Environments

Marc Schaaf*, Arne Koschel*, Stella Gatziu Grivas†, Irina Astrova‡
*Department of Computer Science
University of Applied Sciences and Arts, Hannover, Germany
marc@marc-schaaf.de,akoschel@acm.org
†Institute for Information Systems
University of Applied Sciences Northwestern Switzerland, Olten, Switzerland
stella.gatziugrivas@fhnw.ch
‡Institute of Cybernetics
Tallinn University of Technology, Tallinn, Estonia
irina@cs.ioc.ee

***Abstract*—We propose an activity service as a component in cloud computing with the particular novelty that we base this service on the well-defined and proven semantics of Active Database Management Systems (Active DBMS). In addition, we utilize well-known principles of service oriented architectures. Furthermore we aim to provide an integration with a cloud service mediation component to automatically react to changes occurring in the cloud environment and in this way to implement agility and self management of cloud applications. As contribution of this paper we provide the high-level design of this activity service. This includes architecture, core interfaces and a semantically well-defined rule and execution model, based on extended Active DBMS semantics.**

***Keywords*-Active Database Management System (Active DBMS); Activity Service; Event Condition Action (ECA) Rules; Cloud Computing.**

I. INTRODUCTION

Cloud computing [1] 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 [2]. 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. We propose an activity service for cloud computing that adopts its semantic from the well-proven and clearly defined semantic of Active Database Management System (Active DBMS) style [3], [4] event-condition-action (ECA) rules and extend them for the cloud. Moreover, the design of the activity service is going to be based on proven principles and patterns from service oriented architectures (SOA, [5], [6]). Eventually, we will deliver an activity service with a precisely defined Active DBMS style ECA rule and execution model for the cloud.

The remainder of this article is organized as follows: The next Section will discuss related work. Afterward, Section III introduces Active DBMS style ECA rule processing. This is followed by an introduction of relevant cloud computing concepts. Section V joins both concepts to eventually provide an Active DBMS style ECA rule activity service for the cloud. We contribute the high-level design of this activity service, including architecture, core interfaces, and a semantically well-defined Active DBMS style rule and execution model, that is extended into the cloud.

II. RELATED WORK

Work, which is related to ours, occurs in different areas. Distributed event monitoring, which is an important part of our system, is an excellent instrument for (distributed) monitoring systems, see [7], [8] for overviews, and can contribute general monitoring principles to our work. However, these systems mainly concentrate on primitive event sources. Our work deals with event sources that are typically found in quite heterogeneous cloud computing environments. Event modeling aspects and semantics often lack precision [8] when compared to systems such as Active DBMS. Nevertheless, general work on the design of monitoring services for distributed systems is valuable for transfer into a cloud-based environment. Some event monitoring and propagation within the cloud in conjunction with complex event processing (CEP, [9]) is discussed in [10]. However, ECA rule processing with precisely defined semantics is not its focus.

The precise semantical foundation of our work is based on proven research work from the area of Active DBMS [3], [4]. In particular, we can utilize the Active DBMS manifesto [11]. This manifesto provides a proven ECA rule and execution model with a well-defined, clear semantic. Active DBMS style ECA rule processing will be discussed further in Section III and utilized in Section V.

One step beyond the work in Active DBMS go approaches concerning ECA rule processing in distributed environments. In [12], active functionality was extended into an ECA rule service for CORBA-based distributed, environments. Actually, this approach is one initial step in our direction.

A first step into ECA rule processing within cloud computing is done in [13]. At least some combination of event driven and service-oriented architecture for the cloud is discussed there. However, the work remains quite high-level and in particular focuses on policy-driven event processing for the cloud. It does not really address an activity service for the cloud at all, in particular not with well-defined Active DBMS style semantics.

Web services development standards such as the business process execution language WSBPEL [14], usually operate on a higher level than our approach. However, they are an excellent example for Web Services-based systems, that can generate events such as 'process' or 'activity' started/ended etc. In our approach, we have to deal with events across the heterogeneous cloud services and we must monitor and handle events generated by Web Services-based systems.

III. AN ACTIVE DBMS STYLE ACTIVITY SERVICE

The semantic foundation of our work is based on well-established earlier work from Active DBMS [3], [4], [11]. An Active DBMS is a standard 'passive' DBMS that has the capability to react to events based on event- condition-action (ECA) rules. The Active DBMS monitors the relevant events and notifies the component responsible for executing the corresponding rules (event signaling), which triggers these rules into execution. Rule execution incorporates condition evaluation as a first step and, if successful, action execution as the second step. A variety of execution models exist for the coupling of the transactions that raise events, evaluate conditions and execute actions.

An Active DBMS provides a rule definition language as a mean to specify event types, conditions, actions, and their assembly into ECA-rules. Execution constraints determine the coupling of events, condition evaluation and action execution within and across transactions. Binding information determines the granularity of the data items with which an event is associated. Information on event consumption determines how component events contribute to composite events and how event parameters enter into the computation of the composite event parameters. The rule base of an Active DBMS contains meta information on defined ECA-rules.

Our present work, in particular, follows the Active DBMS manifesto [11], which provides an established ECA rule and execution model with a well-defined semantic. Certainly however, this model requires extensions to take a cloud-based, distributed environment into account. However, the model already includes parameters such as event granularity information or event consumption policies, specifies different coupling modes etc. Such parameters can be summarized as *ECA semantic parameters*.

For traditional Active DBMS the mentioned functionality is usually closely tied to the DBMS. This is due to the usually monolithic system architecture of Active DBMS. Therefore it is quite hard to use their active functionality standalone in other contexts. For this reason, the active functionality was unbundled from Active DBMS to be usable as an activity service in other contexts [15]. It provides connectors for event detection, condition evaluation, action execution and an activity service exposes this as an overall functionality for active ECA rule processing. Now this unbundled active functionality is going to form a solid starting point for our present work as well.

IV. CLOUD COMPUTING CONCEPTS

Cloud computing has emerged as a technology becoming quite popular among companies and business. Computing resources like infrastructure, middleware or database functionality but also applications are provided over the Internet rapidly to users according to actual demands. The delivered resources are governable to ensure requirements like high availability, security, and quality. The key factor is that they are rapidly scalable up- and downwards, therefore the right amount of needed resources can be provided to the users.

Cloud Computing is a new paradigm, a new model based on known technologies like virtualization. What's new is the fast development and deployment of cloud applications. This is the contribution of the cloud computing to agility (fast response/reaction to new requirements, changes in the customer environment). The central element is the predictive management of the whole life cycle of a cloud environment, which is also the challenge. This subsumes all tasks like configuration, scale up/down and charge back.

We consider our work on the activity service as an important contribution towards the support of the predictive management of the cloud environment. For this, we propose a cloud broker or cloud mediator supporting functionality as reported in [16]. In [17], we discuss the extension of the functionality of a cloud broker, which has the intelligence to react to the changes of the business processes or their environment in order to change the cloud configuration (to scale up and down or to choose a new provider). The basis for the implementation of this functionality is the activity service we present in this paper, which could be, for example, used to monitor the utilization of the used services by evaluating corresponding business events. Based on the monitoring results the activity service could inform the cloud mediator that a service is at its capacity limits. In turn the cloud mediator could decide to switch to another service provider to increase the capacity.

V. AN ACTIVE DBMS STYLE ACTIVITY SERVICE FOR THE CLOUD

The aim of our work is to adapt the notion of an Active DBMS like activity service to the cloud. Therefore, we propose to place the required components for active mechanisms in the cloud and to thereby provide the mechanisms of

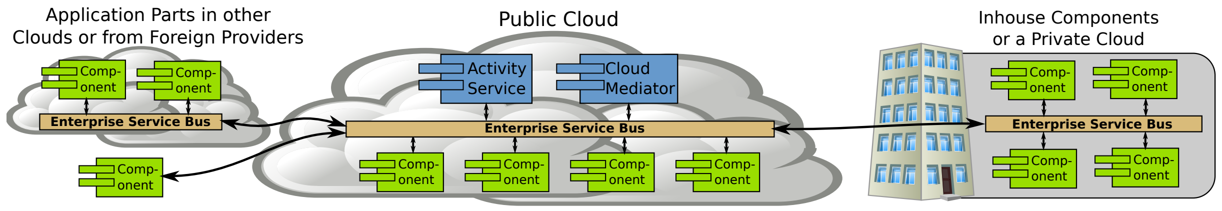
Application Parts in other Clouds or from Foreign Providers
Public Cloud
Inhouse Components or a Private Cloud
Activity Service
Cloud Mediator
Comp-onent
Enterprise Service Bus

Figure 1. The High Level Architecture

an Active DBMS style *activity service* in this environment (Figure 1).

In order to achieve a high flexibility, the active mechanisms follow the unbundling approach mentioned in Section III. They are thus separated into different components. Each of the components provides one or more well-defined interfaces with clear semantics. Thereby the concrete implementation of the different components is interchangeable.

The communication between the components is realized based on the concept of a service oriented architecture (SOA). An Enterprise Service Bus provides means for the communication between the components in the cloud.

In our approach the event producers and consumers are not limited to the components in the cloud where the activity service is located. It can also gather information from other environments like from components in a private cloud of a company or from other clouds provided by other vendors.

Possible application areas for the activity service include the processing of vast amounts of events, which occur, for example, in logistics or finance applications. As mentioned the activity service can also be used for cloud monitoring purposes like for example for the automatic monitoring and scaling of a cloud application where the monitoring would be based on the evaluation of events from the different application parts.

A. The Components of the Activity Service

Figure 2 illustrates the different components of the activity service and their interactions. Their functionality is explained in the following subsections.

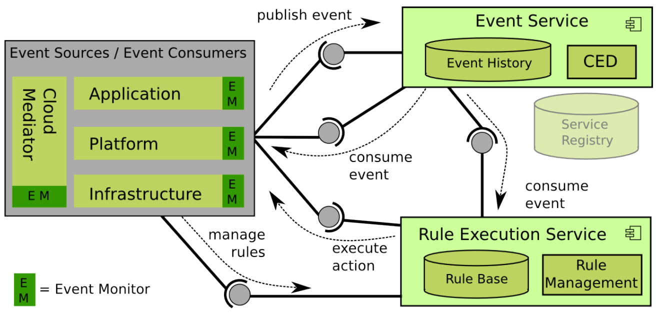

Figure 2. The Components of the Activity Service

1) The Event Service: The event service component implements all activities necessary for the cooperation between event producers and event consumers. It provides a service with the following interface, which can be used by event producers to send their events to the event service:

```
interface EventService{
  void sendEvent (Event)
}
```

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 (see below), which are fed back into the processing mechanism. Consequently they are handled again as if they where 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 inform the event service about the events a handler service is interested in, a filtering criteria has to be added to the WSDL description, which will be extracted by the event service.

Detection of Complex Events: Much work has been done in Active DBMS research regarding the detection of so called complex events ([8], [18]). Complex events are expressions of an event algebra, which are formulated over primitive or complex event types by means of algebraic operators. Say E1 and E2 are event instances. Complex events are then for example:

- disjunction (E1 $\vee$ E2), thus E1 or E2 occurred;
- conjunction (E1 $\wedge$ E2): E1 and E2 occurred, independent of their sequence;
- sequence (E1, E2): First came E1 then E2 occurred.

To detect complex events, basically, two techniques can be distinguished [18]:

- Backward discovery. In this technique upon arrival of a new primitive every event in the history of currently available events are checked, whether they together with the new event form a new complex event.

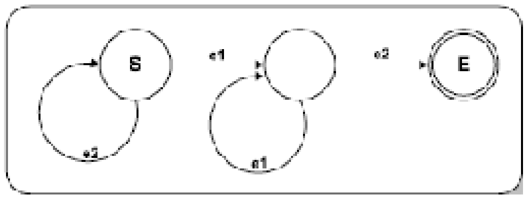

Figure 3. Complex event detection with a finite state automate

- Forward discovery: For forward discovery of complex events a sub-detector exists for every admissible complex event. Complex events are – without going back to the event history – detected in stages. The sub-detectors each have discovery status of 'their' complex event. The arrival of a primitive event leads to another step or a state change within individual sub-detectors. A complex event is detected, whenever a sub-detector reaches its final state.

In Active DBMS, a number of technologies for the discovery of complex events are used, such as finite state automates, Petri nets and event trees. As an illustrative example, Figure 3 shows a finite state automate. It detects a complex event: sequence E = (e1, e2) with start state S and final state E.

We currently aim to integrate backward discovery into the Event Service. A decision on the concrete technology for a prototypic implementation is yet to be made. However, the complex event detection process is hidden from the service consumers and can thus easily be changed to a forward discovery based approach if required.

2) The Rule Execution Service: 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 implemented by each of the components that are intended to be called from within rules. The rule can also provide the action handler with parameters, which can be derived from the rule execution.

The rules that are evaluated are stored in a rule base, which can be managed by a special *rule management service*. Using the rule base, the rules are implemented by the rule execution service.

3) Event Monitors: Normally, not all components are build for active notification by the event service. For other components a *monitor capsule* mechanism is possible. Therefore a small application that acts as a capsule around the actual event source can be realized that obtains the event from the source and transfers it to the event service. In addition, the conversion between different event types can be realized by the capsule.

To further illustrate the event monitoring, a concrete example for a particular kind of event source will now be

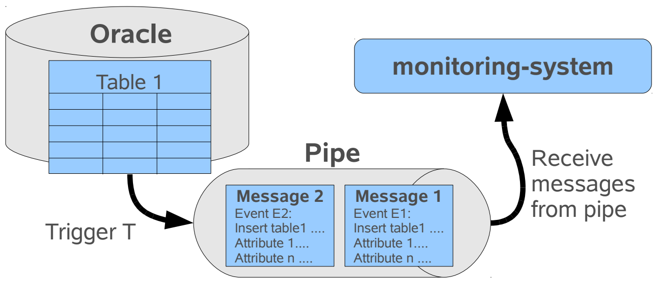

Figure 4. Monitoring using Triggers and Pipes

given. In particular, we utilize earlier work from us [19] to take a look at the monitoring of an 'active' event source, here the Oracle 10 relational DBMS (Figure 4).

The system allows for communication between Oracle sessions by means of so called Oracle Pipes. A pipe is a data structure into which messages may be placed and from where they may be retrieved in FIFO order. If the pipe is empty, a recipient is blocked until a new message is available.

Note that a message in a pipe becomes immediately visible independent of the status of the transaction that placed it there. Even if the transaction that produced the event is aborted, the event may is already passed onto further processing.

A pipe is a communication structure, which naturally fits into cloud computing. In particular larger cloud providers, e.g., Amazon's EC2/S3 or Microsoft's Azure provide cloud messaging queues. Our event monitor thus would read the events from Oracle pipes and place them into Cloud messaging queues, which are connected to the ESB from our activity service, for further processing.

Now suppose that an event type is defined for the source, say insertion of a tuple into some relation. Our monitor capsule internally declares a corresponding Oracle trigger which, when fired, places a message with all relevant information into the pipe. The capsule thread then acts as the recipient and hands the event over to the activity service.

The call-back mechanism has the advantage of event detection without delay and incurs negligible overhead because no query processing is needed in contrast to several other mechanisms, which are discussed in [19]. A significant drawback though, is the lack of standardization for the call-back mechanism and, hence, its limited availability.

Looking at the Active DBMS style ECA semantic parameters (see below), the natural coupling mode for this type of event sources is 'immediate decoupled' due to the independence of pipes from transactions.

All other modes require some additional effort. For example, to support the 'immediate coupled' mode, a second pipe must be installed, which receives the event signaling completion of the sub-transaction. A trigger is defined as the recipient and takes the appropriate action for the main transaction. In order to support the 'deferred decoupled' mode, a wrapper thread must observe a second structure in which the DBMS must make note of the completion of the transaction. Unfortunately, this solution entails some overhead due to the need for polling.

B. The Rule and Execution Model

Since our work follows the Active DBMS style rule and execution model from the Active DBMS manifesto we plan to investigate the semantic parameters for their suitability for cloud environments. It is expected that some modifications need to be made. For example, we can only provide certain transaction coupling modes, since we can't assume all cloud event sources to be able, to participate in 2-phase-commit transactions. Similarly, some event sources might only be able to send events as a whole rather than individually, so the event (sending) granularity might be event source dependent.

Beside evaluating, which ECA semantic parameters still fit for the cloud, we are going to investigate extensions of those parameters, to fit even better into cloud computing. Additional parameters include for example: Options to execute ECA rules across different rule processors within the cloud, configurable reactions to deferred events due to network issues, options to deal with not responding event or data sources and several more like cost parameters, which allow to choose for example an especially cheap activity service or an especially good one, etc.

VI. CONCLUSION

With this work, we aim to create a complete and versatile concept, which reaches from the event monitoring over the preprocessing to the evaluation of sophisticated rules and as a reaction the execution of actions on cloud components. Furthermore our whole concept is based on the well-defined and proven semantics of Active DBMS style ECA rule processing, which other event processing approaches lack (cf. Section II). To provide a maximum of flexibility we utilize the well-known principles of service oriented architectures and provide the different functionalities in different interchangeable components with well-defined interfaces and clear semantics and aim to support heterogeneous event sources by the concept of monitoring capsules. Furthermore we intent to provide an integration of the activity service with a cloud mediator so that the mediator can utilize event information to react immediately to changes in an applications environment.

Our next steps will be the detailed specification of the components, their interactions and interfaces and especially the adoption of the proven, semantically rich Active DBMS style rule and execution model to the new world of cloud computing. Moreover we will work on the specification of an appropriate rule and event definition language. Finally we will provide implementations of the different components of the activity service and evaluate their potential in real world application scenarios.

ACKNOWLEDGEMENT

Irina Astrova's work was supported by the Estonian Centre of Excellence in Computer Science (EXCS) funded mainly by the European Regional Development Fund (ERDF).

REFERENCES

[1] M. Armbrust *et al.*, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep., 2009.

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

[3] J. Widom and S. Ceri, Eds., *Active Database Systems: Triggers and Rules for Advanced Database Processing*. San Francisco, CA, U.S.A: Morgan Kaufmann Publishers, 1996.

[4] N. W. Paton, Ed., *Active Rules for Databases*. New York: Springer, 1999.

[5] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices*. Upper Saddle River, NJ: Prentice Hall, 2004.

[6] T. Erl, *SOA Design Patterns*. USA: Prentice Hall, 2009.

[7] B. Schroeder, "On-Line Monitoring: A Tutorial," *IEEE Computer*, vol. 28, no. 6, pp. 72–80, Jun. 1995.

[8] S. Schwiderski, "Monitoring the Behaviour of Distributed Systems," Ph.D. dissertation, Selwyn College, University of Cambridge, United Kingdom, 1996.

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

[10] G. Wishnie and H. Saiedian, "A complex event routing infrastructure for distributed systems," vol. 2. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 92–95.

[11] The ACT-NET Consortium, "The Active Database Management System Manifesto: A Rulebase of ADBMS Features," *ACM SIGMOD Rec.*, vol. 25, no. 3, pp. 414–471, Sep. 1996.

[12] A. Koschel and P. C. Lockemann, "Distributed events in active database systems: letting the genie out of the bottle," *Data Knowl. Eng.*, vol. 25, no. 1-2, pp. 11–28, 1998.

[13] P. Goyal and R. Mikkilineni, "Policy-based event-driven services-oriented architecture for cloud services operation & management." Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 135–138.

[14] WSBPEL TC, "Web Services Business Process Execution Language Version 2.0," OASIS, OASIS Standard, 2007.

[15] S. Gatziu, A. Koschel *et al.*, “Unbundling active functionality,” *ACM SIGMOD Rec.*, vol. 27, no. 1, pp. 35–40, 1998.

[16] L. Leung, “Cloud computing brokers: A resource guide,” 2010, url: http://www.datacenterknowledge.com/archives/2010/01/22/cloud-computing-brokers-a-resource-guide/ Visited: 10.06.2010.

[17] S. Gatziu, T. U. Kumar, and W. Holger, “Cloud Broker: Bringing Intelligence into the Cloud An Event-Based Approach,” in *Proc. of the 3rd IEEE Intl. Conf. on Cloud Computing*, Miami, Florida, July 2010.

[18] K. Dittrich and S. Gatziu, *Aktive Datenbanksysteme, Konzepte und Mechanismen*. Int. Thomson Publishing GmbH, Bonn, Albany, Attkirchen, 1996.

[19] A. Koschel and I. Astrova, “Event monitoring web services for heterogeneous information systems,” *Proc. World Academy Of Science, Engineering And Technology*, vol. 43, pp. 50–52, 2008.