Event Monitoring Based On Web Services for Heterogeneous Event Sources

Arne Koschel

***Abstract*—**This article discusses event monitoring options for heterogeneous event sources as they are given in nowadays heterogeneous distributed information systems. It follows the central assumption, that a fully generic event monitoring solution cannot provide complete support for event monitoring; instead, event source specific semantics such as certain event types or support for certain event monitoring techniques have to be taken into account.

Following from this, the core result of the work presented here is the extension of a configurable event monitoring (Web) service for a variety of event sources. A service approach allows us to trade genericity for the exploitation of source specific characteristics. It thus delivers results for the areas of SOA, Web services, CEP and EDA.

***Keywords*—** Event monitoring, ECA, CEP, SOA, Web services.

I. Introduction

THE world is nowadays distributed and heterogeneous and so are information systems that are often collections of existing information sources. When integrating such a set of existing information sources into a distributed information systems' architecture one has to deal with the technical (and semantical) heterogeneity of such sources.

Such information systems are today frequently based on Service-Oriented Architectures (SOA) [4], [6], [20]. SOA based information systems in turn often have to process events arising from different sources within them. Thus this is also particularly applicable to distributed services such as event monitoring, which are useful in the context of Event Driven Architectures (EDA, [21]) and Complex Event Processing (CEP [5], [21]). Web services standards [6] deal with this heterogeneity to some degree at a technical level. However, the only provide a little support for event processing (e.g. WS-Notification [22]). In particular they usually do not directly take the (technical and semantical) heterogeneity of event sources themselves into account.

Technical integration of heterogeneous sources is today reasonably supported by Enterprise Service Bus (ESB) systems (e.g. Apache ServiceMix), JBoss ESB etc., which usually support Web services [6]. However, Web services provide general support only and do not or take only quite limited source specific semantics into account. Not only does this apply to WS-* standards such as WS-Notification or WS-BPEL, but also to work on Complex Event Processing (CEP) systems [5] such as Esper [2]. For this reason, our objective is to enhance Web services by mechanisms that allow us to add event source specific application semantics. Thus we provide work in the area of Event Driven Architectures (EDA) [5] combined with work for (Web) Service-Oriented Architectures (SOA) [4].

Our key aim is to provide a particular flexibly configurable event monitoring Web service, which accepts source heterogeneity for a (Web) services environment. Flexibly configurable means the ability to generate code templates at compile time and to provide dynamic parameterization of parts of the event monitoring Web service. Moreover, the examination of options for several configuration options for full Event–Condition–Action (ECA) rule processing, for example options for parallel rule processing engines, are part of our work.

To allow for precisely defined semantics, our event passing follows as far as possible semantics, which were developed for established EDA systems. In particular, we propose the semantics from Active Database Management System (ADBMS) style ECA rules [1]. Our monitoring (Web) services extend earlier work on ECA rule based active information delivery [13], [16], [18] in heterogeneous information systems. This earlier work was limited in flexibility as well as in performance and was developed for CORBA [7] only. It details and extends work from [16] and discusses even more event information sources.

The remainder of this article starts in the next section with a discussion of related work. This is followed by the design and concepts for our event monitoring services (Section III) and followed implementation techniques (Section IV). Section V draws a conclusion and looks at some future work.

II. Related Work

Looking at work, which is related to the work presented here, it is mainly found in the areas of (distributed) event monitoring, Web services themselves, Active DBMSs, distributed ECA rule processing, and workflow systems.

A. (Distributed) Event Monitoring

In (distributed) monitoring systems (see [10] and [11] for overviews) event monitoring techniques are well understood. Such systems can contribute general monitoring principles to the work presented here. These systems mainly concentrate on primitive (often pure) event sources, such as operating system level signals. This is in contrast to the work here, which is concerned with event sources that are typically found in heterogeneous information systems.

Arne Koschel is with the Faculty IV, Department for Computer Science, University of Applied Sciences and Arts Hannover, Hannover Germany (e-mail: arne.koschel@hs-hannover.de).

B. Active DBMS (ADBMS)

ADBMSs offer several elements for use in our work (see [1] and [8] for overviews). Event monitoring techniques in ADBMSs are partially useful, but concentrate mostly on monitoring ADBMS internal events, and tend to neglect external and heterogeneous event sources. For the design of interfaces to our monitoring service, namely those for notifiable and monitored objects, we follow similar design patterns [3].

A major contribution of ADBMSs is their very well defined and proven semantics for definition and execution of ECA rules. This leads to general classifications for parameters and options in ADBMS core functionality [1]. Building upon these proven results, we capture options that are relevant to event monitoring in our general event model, especially event type, event binding and event processing semantics. However, since ADBMSs mostly do not concentrate on heterogeneity (and distribution), our work extends those done for ADBMSs and CEP/EDA into these directions.

C. Distributed ECA Rule Processing

There are (few) systems, which combine the worlds of monitoring and ECA rule processing together. Partially they even take some heterogeneity and distribution from real life information systems into account.

RIMM [9] focuses on reactive mechanisms for database interoperability, but only describes a simple event and ECA rule model with very limited semantics. In the ECA rules of the NCL/NIIIP approach [12] only method event types are supported. This takes no event source specifics into account. The Amalgame project [15] and the WHIPS/TSIMMIS project [14] use ECA rules to support data integration and view materialization in a warehousing environment. To this end, they only need simple ECA rules, which monitor a single source and update derived information. Some of the primitive event monitoring techniques of WHIPS/TSIMMIS seem to meet our needs. Both Amalgame and TSIMMIS take some source specific semantics into account. However, they have not been developed with any eye towards Web services.

D. Web Services

Nowadays Web services implementation platforms themselves give a solid implementation basis to integrate heterogeneous process and information sources in general. The lack however, support for source specific event monitoring. WS-* standards such as WS-Notification or WS-BPEL and similar rule techniques are just relatively generic in their approaches as mentioned above.

E. Web Services

Workflow/BPM systems are at a higher level than ours. They could utilize our work for the event monitoring of resources.

As a conclusion, what is missing in all the above approaches is a monitoring service, which does accepts heterogeneity and is designed for a Web services environment. There is no approach, which combines configurable WSDL based event type specification, monitoring interfaces, a classification scheme and algorithms for monitoring heterogeneous sources together with flexible dynamically parameterized definition of event types. Moreover, there is only a partially discussion on techniques and implementation aspects. To overcome these deficiencies, the goal of our work is to address this combination of aspects in flexibly configurable event monitoring Web services.

III. Concept: Event Monitoring (Web) Services

Our overall work addresses the following problems:

- Description (and detection) of arbitrary event types from heterogeneous event sources.
- Proper utilization of source category specific implementation support for the detection of events from such sources.
- An in depth examination of the supportable degree for parameters from Active DBMS style ECA rules such as event occurrence notification time (after, before, instead) or event–granularity (instance/set–oriented).

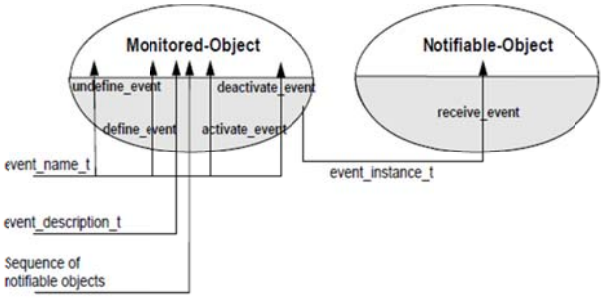

Fig. 1 Monitored-Objects and Notifiable-Objects

The main results of the work presented here are as follows:

1) Development of a WSDL–based, configurable and extensible event type model for arbitrary event sources.
2) Flexible event type descriptions using either direct coded WSDL event types or WSDL as an event description language, or a (simple) meta-model (name/value lists).
3) Provision of the WSDL specification of the service interfaces, which each Monitored-Object in the system has to be implemented. Fig. 1 illustrates the interplay of Monitored-Objects and Notifiable-Objects.
4) Contribution and detailed explanation of a classification scheme, which categorizes event sources by their specific implementation support for monitoring (see Fig. 2).
5) Implementation techniques for monitoring event sources from several categories are examined.
6) Compile time configurability of event monitors is achieved by generation of monitoring technique specific code templates.
7) Discussion of event semantics from ECA rule parameters that each of the monitoring techniques is able to support.

To illustrate the work provided here, first our event source classification scheme in Fig. 2 is described here.

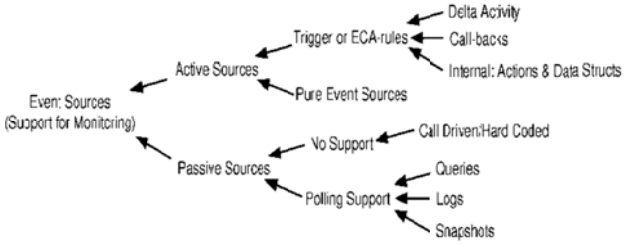

Fig. 2 Classification: Monitoring Support

Heterogeneous information sources differ dramatically in their functional capabilities. To study and provide various encapsulation strategies for event detection in such sources, we categorize the sources with regard to their support for event detection. Clearly, the more support a source provides, the simpler is the construction of a corresponding event monitoring capsule, which wraps or encapsulates the event source in the form of a Web service.

From left to right Fig. 2 starts in general with *event sources*. With respect to their monitoring capabilities such sources might be sub-classified into *active* and *passive* event sources.

Active Event Sources support at least some (active) event detection or event notification mechanism.

- *Pure Event Sources* have no notion of triggers or similar event detection mechanisms. They are purely able to notify about their events, but there is no source specific specialized detection mechanism.
- *Trigger/ECA-Rules Sources* have (at least) some mechanism to specify which events shall be detected (and possibly how and when). They might be further sub-divided as follows:

– *Call-back Sources* are the most versatile since they have a notion of event that has some external effect. Typically, they allow an interested party outside the source to register for an event and then go to sleep, only to be woken up by a call from the source after it detected one of the party's events. Examples are DBMSs with trigger mechanisms which allow as part of the associated action a call to the outside.

– *Internal Action Sources* resemble call-back sources insofar as they are still capable of detecting events. However, the events are entirely internal so that possible reactions remain hidden to the outside and just manipulate their own database. In order to qualify as an event source these systems must provide explicit means to make the events visible, e.g., by storing the events, or the entire trigger instance, in an internal data structure, which might be queried from outside the source.

– *Delta Activity Sources* have the ability to periodically send deltas of the net effect of updates since the previous transmission.

Passive Event Sources: All sources that provide no insight whatsoever into their inner dynamics are termed *passive sources*. Still mechanisms for event detection can be developed.

- *No Support:* There are sources, which provide no support for event detection at all by themselves. However, still a capsule or wrapper might be developed to access such sources as services. Whenever then a service call to such a source happens, this call could be detected as a service call event at the level of the event source capsule.
- *Polling Support:* Several sources have mechanisms, which allow to poll them for new information, for example, by examining a sources' log entry information. They can be further divided as follows:

– *Queryable Sources* are data sources which support a query interface. Interesting data items can periodically be polled (queried) and compared in order to detect state changes.

– *Protocolled Sources* record their actions in log files which can be analyzed to detect events. Examples are mail systems or the log files available with all DBMSs.

– *Snapshot Sources* provide their data in bulk, i.e., without any selective capabilities, thus comparisons must also be done in bulk. Traditional files are a typical as an example.

IV. Event Monitoring Techniques

In [16] we discussed implementation concepts of three types of event sources, in particular: "Event Sources with Triggers and Callbacks", "Event Sources with Internal Triggers and Polling", and "Protocolled Event Sources". As a particular complex example we slightly extend the discussion of "Event Sources with Triggers and Callbacks" as an active event source here. Moreover we discuss two additional types of sources, namely "Queryable Sources" and "Snapshot Sources" below.

A. Event Sources w. Triggers and Callbacks

Let us suppose an event source, which supports triggers and callbacks (here an Oracle relational DBMS, see Fig. 3). Oracle allows the communication with Oracle sessions using so called pipes. A pipe is a data structure, into which messages may be placed and from where they may be retrieved in FIFO order. For retrieval, a recipient process registers with the pipe. The recipient then reads one message from the pipe, processes it, and reads the next message. If the pipe is empty, the recipient will block. It becomes unblocked, after a new message has been placed in the pipe. Note that a message in a pipe becomes immediately visible independently of the status of the transaction that placed it there.

Now suppose that an event type is defined for the source, say update of a tuple in some relation. The wrapper declares a corresponding trigger which, when fired, places a message with all relevant information into the pipe. The wrapper thread then acts as the recipient, and hands the event – with negligible overhead – over to an appropriate receiver service (the Notifiable-Object from above).

The callback mechanism has the advantage of event detection without delay and incurs negligible overhead because no query processing is needed as in the other mechanisms discussed below. An issue however, is the lack of standardization for the callback mechanism and, thus, its limited availability.

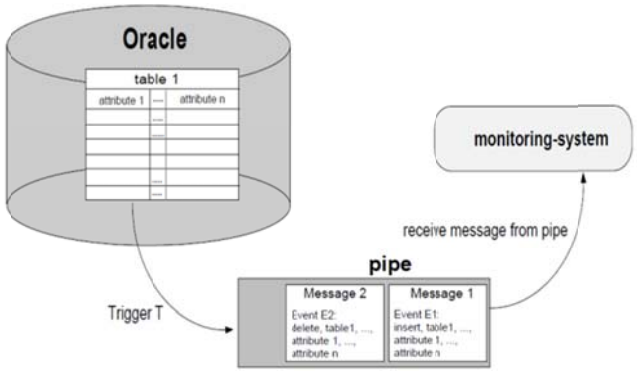

Fig. 3 Monitoring: Triggers and Callbacks

B. Queryable Passive Sources

Now let us have a look at passive information respectively "event" sources. A passive source does not have a notion of events and, thus, does not offer triggers or similar event detection mechanisms by itself. An event monitor for such a source must now somehow simulate the effects of triggers.

We will assume for this case, that we have a source that at least supports some kind of queries (see our classification scheme from above). Such a source could be a DBMS, which does not support triggers, but at least queries, e.g. smaller relational DBMS without triggers or other kinds of DBMS, for example NoSQL DBMS [19] without triggers.

The simulation to monitor such sources for events consists in polling (in the relational DBMS case) the table of interest and comparing its current state with some earlier state in order to detect changes that can be related to a defined event.

In a relational DBMS polling can again be based on SQL queries. Likewise NoSQL DBMSs' generally provide some query mechanism as well, which could be utilized here.

Given a defined event type, a suitable query must be constructed. The only event types possible are those that reflect – in the RDBMS case – tuple insertions, deletions or updates or likewise CUD operations in NoSQL DBMSs'. The query is invoked at each poll and its result compared with the result of an earlier query. The comparisons are based on keys (or identifying ids in the general case). For example, an INSERT event is recognized if tuples are detected with keys that did not exist in the database at the prior poll. A DELETE event is recognized if a key has disappeared between successive polls. An UPDATE event occurred if tuples exist with identical keys but different values in the remainder.

Unfortunately this solution causes much more time overhead – compared to most active sources. This is due to the much larger relations that must now be examined. Consequently, it does not seem to scale well either in database size or number of data sources. However, it is still better than not being able to monitor such sources at all.

Also, events may now be lost, since polling only observes the net effect of events that occurred during a polling interval. For example, an update may be the result of either an update event or a deletion event followed by an insertion event. An insertion event followed by a deletion event on the same tuple may not leave any trace. The choice of polling interval needs particular care and will depend on the application's time constants and tolerance for losses.

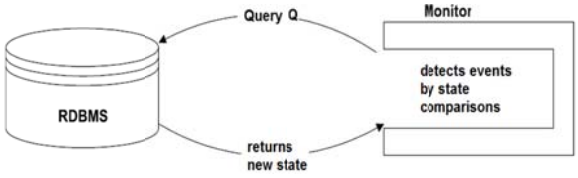

Fig. 4 Monitoring: Queryable passive sources

C. Passive Snapshot Sources

Encapsulation of passive snapshot sources follows similar concepts to the encapsulation of queryable passive sources (cf. Fig. 5). However, this time not even a discriminatory querying facility is provided, so that a structure such as a file must be inspected and compared to an older state in bulk.

Consequently, event detection in such sources involves a quite significant time overhead that limits scalability. A typical technical implementation example would be the periodic poll of a file with a state A, that is compared to a later state B of the file. Technical, for example, utilizing the diff utility is a possible option here. Again, as with the UPDATE case in queryable passive sources, events might get lost. However, frequently bulk comparisons are just "the option" to detect events in such types of sources at all.

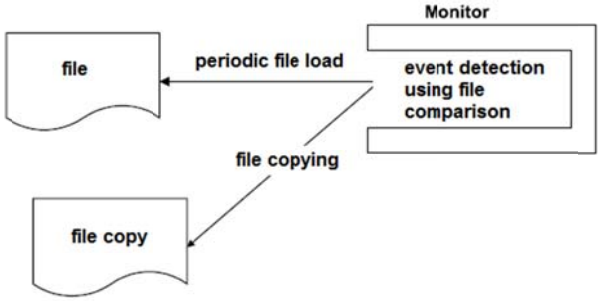

Fig. 5 Monitoring: Passive Snapshot Sources

V. Conclusion

This article presented the conceptual design for event monitoring Web services for particular heterogeneous event sources. It extends and refines earlier work in this area [13], [16], [18] by incorporating the concept of Monitored-Objects and Notifiable-Objects. Moreover, it explains and details the classification scheme for heterogeneous event sources and examined additional implementation concepts for further event sources.

In our current and future work we are exploring, how the developed concepts might be transferred and implemented within nowadays cloud computing environments [17]. Cloud computing environments are heterogeneous almost "by nature" and seem to be as such a good extension space for the presented work.

Acknowledgment

Acknowledgment goes to the co-workers and students from

HS Hannover, which helped to implement the event monitoring services in our research work. Also the author wants to acknowledge the work of my colleagues and cooperation partners in previous publications around this topic.

REFERENCES

[1] ACT-NET Consortium. The Active DBMS Manifesto. ACM SIGMOD Record, 25(3), 1996.
[2] ACT-NET Consortium. The Active DBMS Manifesto. ACM SIGMOD Record, 25(3), 1996.
[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns. Addison-Wesley Publishing Company, 1995.
[4] D. Krafzig, K. Banke, and D. Slama. Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall, 2005.
[5] D. Luckham. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman, 2002.
[6] E. Newcomer and G. Lomow. Understanding SOA with Web Services. Addison-Wesley, 2004.
[7] Object Management Group. CORBA Home Page. Technical Report, Object Management Group, Inc. (OMG). Available: http://www.corba.org/
[8] N. W. Paton, editor. Active Rules for Databases. Springer, New York, 1999.
[9] M. Young. The Technical Writers Handbook. Mill Valley, CA: University Science, 1989.
[10] B. Schroeder. On-Line Monitoring: A Tutorial. IEEE Computer, 28(6):72–80, June 1995.
[11] S. Schwiderski. Monitoring the Behavior of Distributed Systems. PhD thesis, Selwyn College, University of Cambridge, University of Cambridge, Computer Lab, Cambridge, United Kingdom, 1996.
[12] S. Su, H. Lam, T. Yu, S. Lee, and J. Arroyo. On Bridging and Extending OMG/IDL and STEP/EXPRESS for Achieving Information Sharing and System Interoperability. In Proc. 5th Annual Express User Group Int. Conf. (EUG), Grenoble, France, October 1995.
[13] G. v. Bultzingsloewen, A. Koschel, and R. Kramer. Active Information Delivery in a CORBA-based Distributed IS. K. Aberer and A. Helal, editors, In 1st IFCIS CoopIS. IEEE CS Press, 1996.
[14] J.Widom. Research Problems in Data Warehousing. In Proc. 4th Int. Conf. Information and Knowledge Management (CIKM), November 1995.
[15] G. Zhou, R. Hull, R. King, and J. Franchitti. Supporting Data Integration and Warehousing Using H2O. Data Engineering, 18(2):29-40, June 1995.
[16] A. Koschel, I. Astrova. Event Monitoring Web Services for Heterogeneous Information Systems. World Academy of Science, Engineering and Technology (WASET), Vol:19 2008-07-27, 50-52, 2008.
[17] M. Armbrust et al. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, EECS Department, University of California, Berkeley, 2009.
[18] A. Koschel, R. Kramer. Configurable Event Triggered Services for CORBA-based Systems. Proc. 2nd Intl. Enterprise Distributed Object Computing Workshop (EDOC'98), San Diego, California, U.S.A, November 1998.
[19] P. Sadalage, M. Fowler: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pearson, 2013.
[20] T. Erl: Service-Oriented Architecture: Concepts, Technology, and Design, 5th edt., Prentice Hall, NJ, 2005.
[21] R. Bruns, J. Dunkel: Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse, Springer, 2010.
[22] OASIS Open: Web Services Base Notification 1.3, OASIS Standard, Oct. 2006.