International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:7, 2008

Event Monitoring Web Services for Heterogeneous Information Systems

Arne Koschel, and Irina Astrova

***Abstract*—**Heterogeneity has to be taken into account when integrating a set of existing information sources into a distributed information system that are nowadays often based on Service-Oriented Architectures (SOA). This is also particularly applicable to distributed services such as event monitoring, which are useful in the context of Event Driven Architectures (EDA) and Complex Event Processing (CEP). Web services deal with this heterogeneity at a technical level, also providing little support for event processing. Our central thesis is that such a fully generic solution cannot provide complete support for event monitoring; instead, source specific semantics such as certain event types or support for certain event monitoring techniques have to be taken into account. Our core result is the design of a configurable event monitoring (Web) service that 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*—**ECA, CEP, SOA, and Web services.

I. Introduction

THE world is nowadays distributed and heterogeneous and so are information systems that are often collections of existing information sources. Technical integration of heterogeneous sources is today reasonably supported by Enterprise Service Bus (ESB) systems (e.g. Apache ServiceMix), 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 service, which accepts source heterogeneity for a (Web) services environment. By configurable, we mean that we are able to generate code templates at compile time and to provide dynamic parameterization of parts of the service. Moreover, several configuration options for full Event–Condition–Action (ECA) rule processing, e.g., parallel rule engines are part of our work.

To allow for well defined semantics, our event passing follows as far as possible semantics, which were developed for well 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] in heterogeneous information systems, which was limited in flexibility as well as performance and was developed for CORBA [7] only.

Arne Koschel is with the Department of Computer Science, Faculty IV, University of Applied Sciences and Arts, Ricklinger Stadtweg 120, 30459 Hannover, Germany (phone: +49 511 9296 1839; fax: +49 511 9296 1810; e-mail: Arne.Koschel@gmx.de).
Irina Astrova is with the Institute of Cybernetics, Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia (e-mail: irinaastrova@yahoo.com).

II. Related Work

Work related to ours is mainly found in the areas of event monitoring techniques, Web services themselves, ADBMSs, distributed ECA rules, and workflow systems.

A. *Event Monitoring Techniques*

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

B. *Web Services*

Web services themselves today provide us with a solid basis, but lack support for source specific event monitoring. As mentioned, WS-BPEL and similar rule techniques are just relatively generic in their approaches.

C. *ADBMSs*

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 the 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.

D. Distributed ECA Rules

Some systems combine the worlds of monitoring and ECA rule processing together, and partially take some heterogeneity and distribution from real life information systems into account. RIMM [9] concentrates 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, but on the other hand, they have not been developed with an eye towards Web services.

E. Workflow Systems

Workflow 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 event monitoring service.

III. Event Monitoring (Web) Services

Our work addresses the following problems:

1) Description (and detection) of arbitrary event types from heterogeneous sources.
2) Utilization of source category specific implementation support for event monitoring.
3) Examination of the supportable degree for parameters from ECA rules such as event occurrence notification time (after, before, instead) or event–granularity (instance/set–oriented).

The main results of our work 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.
4) Contribution of a classification scheme, which categorizes event sources by their specific implementation support for monitoring (see Fig. 1).
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 our work, we will give implementation examples of three sources from our event source classification scheme in Fig. 1.

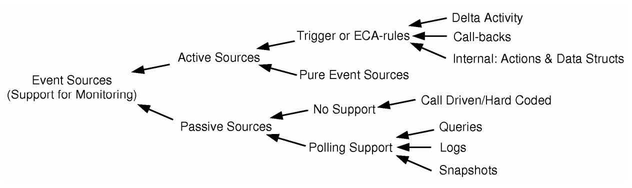

Fig. 1 Classification: Monitoring Support

A. Event Sources with Triggers and Callbacks

Let’s suppose an event source, which supports triggers and callbacks (here an Oracle DBMS) (see Fig. 2). The system allows the communication between Oracle sessions by means of 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 insertion of a tuple into 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.

Furthermore, it shows that our implementation flexibly allows for dynamic specification of event types to be monitored.

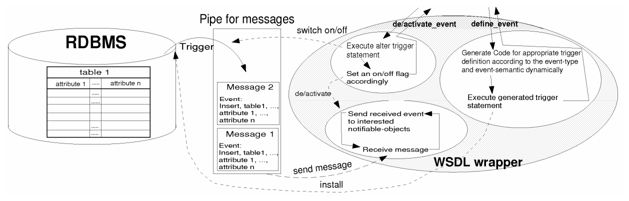

Fig. 2 Monitoring: Triggers and Callbacks

B. *Event Sources with Internal Triggers and Polling*

Next, let's suppose that a source allows only triggers with purely database internal effects (see Fig. 3). This somehow requires a solution that simulates the above pipe and writes trigger results to an externally visible structure, e.g., a mirror table.

As before, if an event type is defined, a corresponding trigger will be declared. As its action, it places a description of the event in the mirror table. E.g., if the source is a relational DBMS the mirror table may now be queried like any other relation by suitable SQL statements. Because of lack of a callback mechanism, the wrapper thread must poll the mirror table to detect any new events, and read and delete the corresponding tuple (if any).

Of course, the mirror table has the disadvantage of space and time overhead compared to callbacks. An unsuitable polling frequency easily results in either large delays in event detection or – if it is too high – the overhead may become unbearable. At least, there is a guarantee that no events are lost.

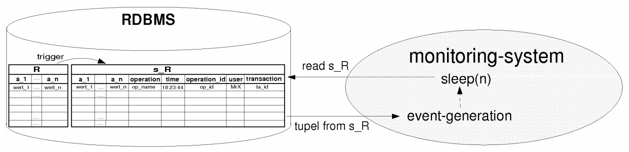

Fig. 3 Monitoring: Triggers and Polling

C. *Protocolled Event Sources*

Protocolled event sources write a trace of all their actions into a log file (see Fig. 4). The log file is then served as the above mirror table. Of course, the log file must also be polled to determine whether events of a given type occurred. Since all actions are protocolled, careful inspection of the log file ensures that no event will be lost.

Mail systems or DBMSs are typical protocolled sources. A single log file might result in large time overhead because events are indiscriminately recorded no matter whether they have been defined to be of interest or not. If a source allows us to freely declare log files, a new file may be created with each definition of a new event type. This will reduce time overhead, but induce space overhead.

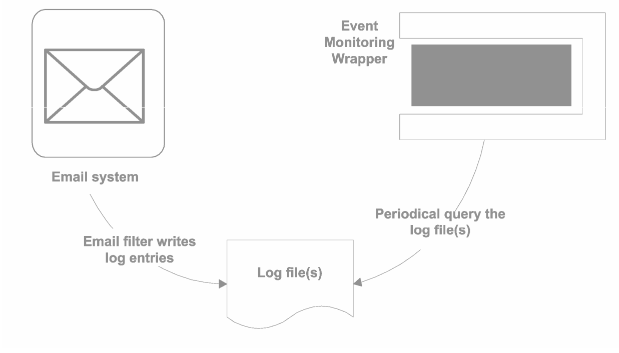
Event
Monitoring
Wrapper
Email system
Periodical query the log file(s)
Email filter writes log entries
Log file(s)

Fig. 4 Monitoring: Protocolled event sources

References

[1] ACT-NET Consortium. The Active DBMS Manifesto. *ACM SIGMOD Record*, 25(3), 1996.

[2] EsperTech. *Esper Reference Documentation*, version 2.0.0. Technical Report. Available: http://esper.codehaus.org/

[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.