A Sensor-based Approach to Symptom Recognition for Autonomic Systems

Jeffery Li, Patrick Martin, Wendy Powley
School of Computing, Queen's University
Kingston, ON, Canada
{junli, martin, wendy}@cs.queensu.ca

Kirk Wilson, Chris Craddock
CA Labs, CA, Inc
Islandia, NY USA
Kirk.Wilson@ca.com

Abstract— **The increased complexity of today's distributed, composite, Web-based systems presents difficult and unique systems management problems. The way these systems interact, and the fact that they often span organizational boundaries, render them difficult to monitor and manage. Autonomic Computing has emerged as a promising approach to the management of complex systems. A key to realizing fully autonomic systems is the development of monitoring tools that provide the controllers with adequate and meaningful performance information, especially the identification of symptoms that indicate potential underlying problems. We present an event- driven sensor approach to a monitoring system whereby a hierarchy of dedicated, simple sensors monitors and correlates low level events into a meaningful representation of the system performance that can be used for problem determination. Our approach utilizes the OASIS Web Services Distributed Management (WSDM) standards.**

Keywords; autonomic computing, monitoring, symptom recognition, WSDM

I. INTRODUCTION

For decades, developers have been producing increasingly complex systems. Today, Web-based platforms provide standard protocols to connect these complex systems, regardless of the technologies or devices they use, or their locations. Today's systems consist of multiple distributed components running on heterogeneous platforms with multiple applications interacting in unpredictable ways. Management of such systems is as complex as the systems being managed. To facilitate management tasks such as problem determination, resource allocation and performance tuning, the use of Autonomic Computing has been proposed [1].

The goal of Autonomic Computing is to build self-managing IT infrastructures, that is, those that configure, heal, optimize, and protect themselves with little to no human intervention. A typical Autonomic Computing solution implements a feedback loop that includes system monitoring, system analysis, planning for action, and action execution. When the monitoring subsystem detects one or more events that indicate a decline in performance, corrective action, such as reallocation of resources, tuning of one or more components, or perhaps enforcement of workload control measures, may be initiated to attempt to solve the problem.

An event represents some activity that occurred in a system. A failed attempt to connect to a database, for example, is an event. A complex event represents an abstraction or aggregation of other events. In other words, it is an event that occurs because some series of other events occurred [2]. Certain event or complex event occurrences are indicative of problems within the system, that is, they are symptoms of potential failure.

A symptom is a form of knowledge that indicates a problem or some specific situation in a managed environment [3]. For instance, a response time decline in a Web server is a symptom that is indicative of a variety of potential problems. By detecting and recognizing symptoms occurring in a system, the scope of problem identification is narrowed. In an autonomic system, symptom occurrences are recognized by the monitoring system which must effectively monitor and filter collected data to support problem determination. Once identified, current symptoms can be compared against a database of known symptom patterns and successful resolutions to automatically find a solution to recurring problems [4].

In this paper, we propose a monitoring subsystem based on an event-driven sensor architecture. In our approach, software sensors recognize events and generate new events and hierarchies of these simple sensors cooperate to collect, aggregate, and correlate current events to ultimately generate a set of symptoms occurring in the system.

We propose a lightweight, flexible, standards-based approach to symptom recognition in an autonomic environment. The proposed framework for symptom recognition will ultimately assist in system problem determination. Although we focus the description of our approach on a Web Services environment in this paper, our approach is applicable to virtually any distributed environment.

The paper makes several research contributions including a) the specification of a highly distributed, policy-based sensor framework for management, b) the proposal of a novel use of complex event processing, and c) the adoption of the OASIS Web Services Distributed Management standard to illustrate the relevance and applicability of this standard to systems management.

The remainder of the paper is organized as follows. Section II provides an introduction to Web Services Distributed Management (WSDM) and outlines some related work from the literature. The architecture of our monitoring system is described in Section III. Section IV illustrates our approach using a sample scenario from our Autonomic Web

Services Environment (AWSE) implementation. Section V provides conclusions and suggestions for future work.

II. BACKGROUND AND RELATED WORK

Web Services Distributed Management (WSDM) is a key technology used in our approach. In this section we provide an introduction to WSDM along with a discussion of relevant work from the literature.

A. Web Services Distributed Management (WSDM)

A Web Service is defined by the World Wide Web Consortium (W3C) as "a software system designed to support interoperable machine to machine interaction over a network". In a typical Web Services environment, service providers and consumers communicate using XML messages that follow the SOAP [5] standard. In this model, the consumer sends the request to the provider for services and the service provider responds to the consumer with the requested services. Web Services try to maximize the interoperability and minimize the degree of coupling between a consumer and a service provider.

Numerous standards pertaining to Web Services have emerged over the past few years. The WS-Notification (WSN) framework [6] is based on event notification. This specification defines a mechanism for a service to distribute information to other services, without prior knowledge of the receivers. WSN includes WS-BaseNotification and WS-Topic. WS-BaseNotification defines the interface that WS-Notification consumers and producers should expose. WS-Topic addresses the definition and use of topics to which Web Services consumers can subscribe. WS-Notification provides the publish-subscription services for Web Services architectures.

Web Services Distributed Management (WSDM) [7] is a standard for the management of Web-based services. It allows a manager service to communicate with any other service that is WSDM-compliant. WSDM consists of two specifications, namely Management Using Web Services (MUWS) [8] and Management of Web Services (MOWS) [9]. MUWS defines the interfaces to represent and access the manageable functions of available resources. MUWS also provides a standard management event format, namely WSDM Event Format (WEF) to improve interoperability and correlation between events. The MOWS standard defines how to manage Web Services resources as well as how to describe and access the manageable functions using MUWS. WSDM supports notifications using WS-Notification and WEF messages.

Our monitoring system is based on WSDM, using WS-Notification and WEF messages for communication.

B. Related Work

Self-healing is identified as a key property of Autonomic Computing systems. Self-healing is the property that enables a system to perceive that it is not operating correctly and to make the necessary adjustments to return to a normal state of operation [10]. Research into self-healing has appeared for a number of classes of software systems including database management systems (DBMSs) and Web services.

Nehme [11] proposes a framework for self-healing DBMSs. The framework, which is intended to be integrated inside the DBMS, actively detects problem symptoms, matches against problems' signatures and performs corrective actions. Pernici and Rosati [12] propose a method and a tool based on pattern recognition to automatically learn repair strategies to support self-healing of Web services. WS-Diamond [13] is a framework supporting the execution of self-healing Web Services, monitoring, detecting and recovering failures and faults with QoS and dependability guarantees. In all these systems monitoring and symptom recognition are key steps in self-healing. Our proposed framework can be used to effectively perform these steps in a variety of situations and systems.

Considerable research effort has been spent on autonomic fault detection and self-healing techniques for networks. The AHSEN (Autonomic Healing-Based Self Management Engine for Network Management) is a policy-based management system that monitors and heals a network system using context-driven self-management functions [14]

We propose a hierarchical approach to symptom recognition where information is passed from lower level sensors and is correlated and analyzed by higher level sensors. Similar approaches have been used in autonomic computing employing hierarchies of autonomic elements to manage systems [15]. Lapouchnian et al. [16] describe a hierarchical autonomic architecture where leaf nodes correspond to the managed elements and non-leaf nodes are used to orchestrate the lower-level elements. This is similar to our approach where, at the lowest level of the hierarchy the sensors perform monitoring of the resources, whereas the higher levels refine the information and eventually produce a symptom report.

Our approach is policy-based and, unlike most of the current approaches, it does not require any changes to the monitored system. We assume a standard management interface for managed resources (using WSDM) and the monitoring system is layered on top of these components.

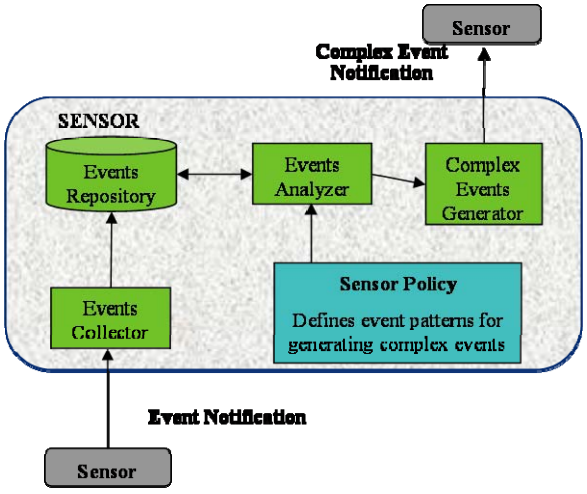

Figure 1. Sensor architecture

III. SYSTEM ARCHITECTURE

The monitoring system consists of a hierarchy of basic units called sensors (shown in Figure 1) that communicate via event notification. Sensors consume event notifications from other sensors, process the information, and emit new event notifications (complex events) that are consumed by higher level sensors, or other components. A notification emitted by a sensor identifies one or more symptoms that were recognized by the sensor. The behaviour of a sensor is controlled by its policy. The policy specifies how incoming events should be processed, when and if a complex event should be generated, and defines the contents of events produced by the sensor.

Each sensor is identical, with the exception of its policy. Individual sensors are implemented as WSDM entities, thus exposing a standard management interface in addition to providing the publish/subscribe capabilities that are used for communication amongst the sensors.

The publish/subscribe capability allows a sensor to subscribe to topics produced by other sensors (or other WSDM entities) and to receive notifications of events published to these topics. A topic is a way to organize and categorize items of interest for subscription. For example, a sensor that monitors response time for a Web server may provide a topic called "ResponseTime" to which it issues an event notification containing the symptom "Response Time Increase" whenever the response time of the server increases. Other sensors can subscribe to the "ResponseTime" topic so that they are notified of Web server response time increase events. Sensors may provide multiple topics to which they publish events, and they may subscribe to multiple topics, thus receiving notification messages from multiple sensors.

The behaviour of a sensor is defined by its policy. The policy defines the topics produced by the sensor and the topics subscribed to by the sensor as well as the contents of the complex events generated by the sensor. Complex event notifications contain a description of symptom(s) recognized by the sensor. Most importantly, the policy defines the conditions under which a complex event should be generated and published to the topics produced by the sensor. We refer to this as the condition/action pattern. An example of a condition/action pattern for the "ResponseTime" sensor described above is "the response time has increased more than 10 percent over the past 5 minutes". If the condition/action pattern is found to be satisfied then a complex event is generated by the sensor and a notification is published to the "ResponseTime" topic indicating that there has been an occurrence of the symptom "Response Time Increase".

Incoming event notifications are collected by the sensor's Events Collector and are saved in the original XML format in a repository that is unique to each sensor. The arrival of a new event in the repository triggers the Events Analyzer which evaluates the new event in light of the condition/action pattern. The condition/action pattern is stated using XQuery [17], a standard language for querying XML data. The XQuery returns a list of "matches" from the repository if the policy is satisfied. A null set is returned if the policy is not satisfied. If matches are returned, a complex event notification is generated by the Complex Events Generator and published to the appropriate topic, notifying subscribers of the event. In the case of the ResponseTime sensor, if the XQuery representing the conditional/action pattern "the response time has increased more than 10 percent over the past 5 minutes" returns a non-empty set when evaluated on the current contents of the sensor's repository, then a complex event is generated with the contents "Response time increase", a symptom that indicates possible performance degradation.

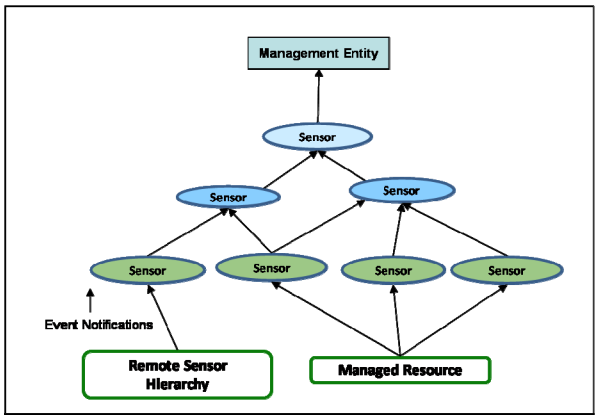

Figure 2. Sensor hierarchy

The monitoring system consists of a hierarchy of simple, cooperating sensors, each with a very narrow view of the system. The general architecture is shown in Figure 2. At the lowest level of the hierarchy, sensors consume event notifications either directly from a managed resource (via the resource's WSDM endpoint) or from another sensor hierarchy. The event notifications are stored and analyzed by the sensor and, if the condition/action pattern is satisfied, a complex event is generated and passed on to the sensors at the next level. As data flows up the hierarchy, the contents of the event notifications (ie. the symptoms) become more meaningful. At the top-most point in the hierarchy, an event notification is passed from the monitoring system to a management entity which will perform management functions. At this level, a significant amount of information will be known about the symptoms that have occurred in the system. Additional information, if required, can be obtained by querying the sensor repositories where the original events are stored.

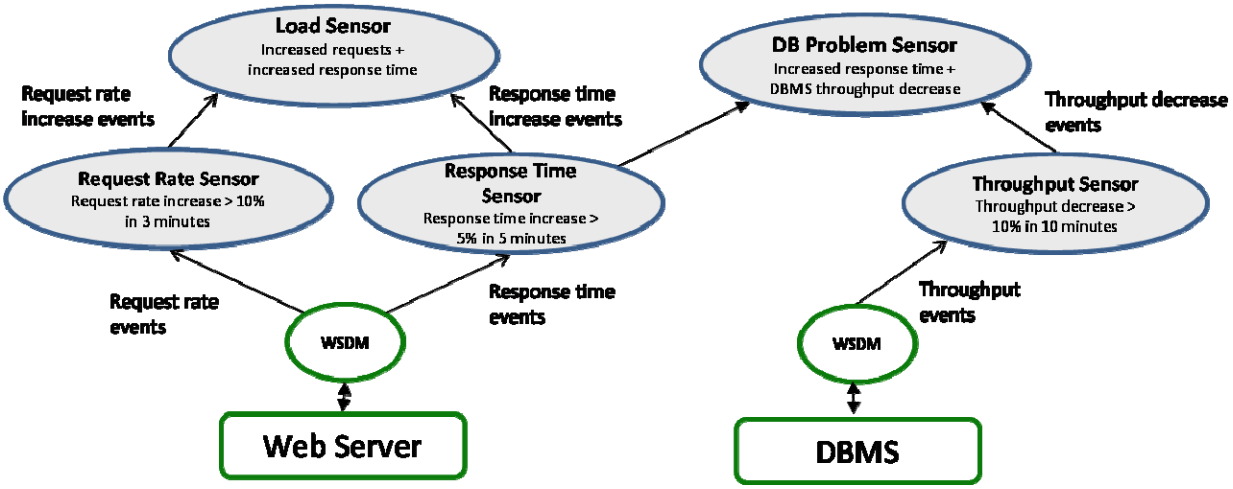

Figure 3. AWSE Scenario

IV. SCENARIO

A prototype of our symptom recognition system was implemented using Eclipse TPTP [18], Apache Muse [19] and IBM DB2 [20] and it has been integrated into a previously implemented system called Autonomic Web Services Environment (AWSE) [21]. AWSE is a framework that was developed for testing and showcasing ideas for autonomic management of computing systems. We have implemented the sensor-based approach to symptom recognition within this system to monitor various components such as the Web server, the Web Services, and the DBMS server.

To illustrate the functionality of the monitoring system, we consider a small portion of the AWSE monitor that monitors the Web server and a supporting Database Management System (DBMS) as shown in Figure 3. The Web server's WSDM interface provides, among other metrics, the average response time and the average number of requests (the Request Rate) that are issued to the server. The DBMS's WSDM interface provides an average throughput measurement, that is, how many queries are being processed per time interval.

An increased response time for the Web server is a symptom of a potential system problem. However, it may be caused by a number of factors such as an increase in the number of requests of a more complex nature, delays caused by supporting components (such as the DBMS) or perhaps an increased load on the system. The increase in response time combined with one or more other symptoms provides deeper insight into the actual cause of the system and narrows the scope of problem determination for the management components.

The sensor hierarchy for our scenario is shown in Figure 3. Two low level sensors receive events directly from the Web server's WSDM interface. The Response Time sensor receives events periodically regarding the current average response time of the server. This sensor's policy indicates that a complex event should be generated if the response time has increased more than 5 seconds over the past 5 minutes. This condition/action statement is specified by the following XQuery:

```
xquery declare namespace muws1="http://docs.oasis-
open.org/wsdm/muws1-2.xsd";declare namespace
muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd" ;
for $a in db2-
fn:xmlcolumn('RESPTIME.EVENT')/muws1:Managemen
tEvent, $b in db2-
fn:xmlcolumn('RESPTIME.EVENT')/muws1:Managemen
tEvent  where xs:dateTime(fn:string($b/@ReportTime)) >
xs:dateTime(fn:string($a/@ReportTime)) and
xs:dateTime(fn:string($b/@ReportTime)) &lt;
xs:dateTime(fn:string($a/@ReportTime)) +
xdt:dayTimeDuration("PT5M") and
fn:number($b/muws2:Situation/muws2:Message)-
fn:number($a/muws2:Situation/muws2:Message)>5
return &lt;x&gt;{$a/muws1:EventId/text()},
{$b/muws1:EventId/text()}&lt;/x&gt;
```

Each time the Response Time Sensor receives a notification containing new response time data, the event notification is saved in the Response Time Sensor repository and the current data is evaluated against the above condition/action pattern. If the policy is satisfied, that is, it is found that the response time has increased more than 5 seconds over the past 5 minutes, a complex event is generated and published to the "ResponseTimeIncrease" topic. Notifications published to this topic will be consumed by higher level sensors that have subscribed to the "ResponseTimeIncrease" topic. In this scenario, these sensors include the Load Sensor and the DB Problem Sensor.

Similarly, at the lowest level, the Request Rate sensor periodically receives events which contain the latest average Request Rate for the Web server. The Request

Rate Sensor's policy indicates that a complex event notification should be published if the Request Rate has increased more than 10% in the past 3 minutes. These messages are published to the topic "RequestRateIncrease". If the policy is satisfied, the sensor sends a complex event notification with the message "Request Rate Increase". This event is received by the Load Sensor that has subscribed to receive such events.

At the second tier, the Load Sensor subscribes to both the "RequestRateIncrease" and the "ResponseTimeIncrease" topics. The policy for the Load Sensor requires events to be received from both topics. The condition/action statement for the Load Sensor is as follows:

```
xquery declare namespace muws1="http://docs.oasis-
open.org/wsdm/muws1-2.xsd";declare namespace
muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd";
for $a in db2-
fn:xmlcolumn('LOADSENSOR.EVENT')/muws1:Manage
mentEvent, $b in db2-
fn:xmlcolumn('LOADSENSOR.EVENT')/muws1:Manage
mentEvent where
$a/muws2:Situation/muws2:Message/text() = 'Request
Rate Increase' and
$b/muws2:Situation/muws2:Message/text() = 'Response
Time Increase' return
&lt;x&gt;{$a/muws1:EventId/text()},
{$b/muws1:EventId/text()}&lt;/x&gt;
```

Thus, for example, if the Load Sensor receives a message that the Web server response time has increased and the request rate has also increased then the policy has been satisfied. Although not shown in our XQuery (for brevity), this policy would also include a specification of time, that is, the events should be received within a certain time frame for the policy to be satisfied. If its condition/action statement is satisfied, the Load Sensor sends a notification to a higher level entity (a management entity, or perhaps another sensor hierarchy) indicating that the load on the server has increased and delays are occurring; two symptoms that may or may not be related.

Similarly, the DB Problem Sensor waits for events from the Response Time Sensor and the Throughput Sensor. If events are received from both sensors within a defined time frame, the DB Problem Sensor sends a notification to a higher level entity indicating that the DBMS throughput has increased and the Web server is experiencing delays.

The top level management entity receives a notification from a sensor containing a list of current system symptoms. In this scenario, the symptoms will either include <Response Time Increase, Request Rate Increase> or <Response Time Increase, DBMS Throughput Increase>.

V. CONCLUSIONS

Today's information systems are becoming increasingly complex and difficult to manage. The Web Services standards and development platform provides a method to loosely decouple systems and simplify interoperability among systems and components. The WSDM standards provide a standard interface for the monitoring and management of Web services enabled systems which employ events to represent the situation data. We utilize these events to monitor components using a sensor-based approach. Using complex events processing, sensors detect patterns in a sequence of events or in a time window. Using a hierarchy of sensors that communicate using standard Web protocols, we are able to monitor resources and provide suggestions for decision-making systems by processing events.

In this paper, we provide an exploration of sensor-based symptom recognition that incorporates complex event processing and policy based design as the two key technologies. A sensor implements a complex events processing model, aggregating events, running rules against them and taking action by producing a complex event indicating that one or more symptoms have been identified. Hierarchies of sensors are formed to aggregate events from multiple sources. A sensor's behavior is abstracted to a pattern which is expressed as a policy. One of the key research issues remaining is how to effectively design and build potentially very complex sensor networks. Scalability is another issue that requires investigation; what will be the overhead of sensor communication in a hierarchy consisting of large numbers of sensors? In addition, we need to address component failure; how will we detect and handle sensor failure?

The framework presented in this paper represents the initial stage of a comprehensive approach to the development of a self-healing system including monitoring, symptom recognition, and problem resolution for a distributed system. Our vision is to extend the sensor hierarchy to include management entities based on the same architecture as the sensors. The management entities will be event-driven WSDM entities that process and analyze event notifications to make management decisions based on the symptoms recognized by the monitoring system.

REFERENCES

[1] An architectural blueprint for autonomic computing. 2006, IBM; April 12 2008, http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf.

[2] D. Luckham, "The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems", Addison-Wesley, 2002.

[3] Marcelo Perazolo, "Symptoms Deep Dive, Part 1: The Autonomic Computing Symptoms Format", October 18, 2005, http://www.ibm.com/devloperworkds/autonomic/library/ac-symptom1/.

[4] Mark Brodie, Sheng Ma, Guy Lohman, Laurent Mignet, Natwar Modani, Mark Wilding, Jon Champlin, Peter Sohn, "Quickly Finding Known Software Problems via Automated Symptom Matching", *Proceedings of the Second International Conference on Autonomic Computing (ICAC '05)*, Seattle, WA, 13-16, 2005, pp. 101-110.

[5] Simple Object Access Protocol (SOAP) 1.2. 2007, World Wide Web Consortium; June 20 2008, http://www.w3.org/TR/soap/.

[6] WS Notification. 2004, Organization for the Advancement of Structured Information Standards; July 2 2008; http://www.oasis-open.org/specs/index.php#wsnv1.3.

[7] WSDM v1.1. 2008, Organization for the Advancement of Structured Information Standards; May 18 2008, http://www.oasis-open.org/specs/index.php#wsdmv1.1.

[8] WSDM Management Using Web Services v1.0 (WSDM-MUWS). 2005, Organization for the Advancement of Structured Information Standards; July 1 2008, http://www.oasis-open.org/specs/index.php#wsdm-muwsv1.0.

[9] WSDM Management Of Web Services v1.0 (WSDM-MOWS). 2007, Organization for the Advancement of Structured Information Standards; May 20 2008, http://www.oasis-open.org/specs/index.php#wsdm-mowsv1.0.

[10] D. Ghosh, R. Sharman, H.R. Rao and S. Upadhyaya. "Self-Healing Systems – Survey and Synthesis". *Decision Support Systems 42*, pp. 2164 – 2185, 2007.

[11] R. Nehme, "Database Heal Thyself", *Proc of 3rd International Workshop on Self-Managing Database Systems (SMDB 2008),* April 2008.

[12] B. Pernici and a Rosati. "Automatic Learning of Repair Strategies for Web Services". *Proc of 5th European Conf. on Web Services*, pp. 119 – 128, 2007.

[13] L. Console and WS-Diamond Team. "WS-DIAMOND: an approach to Web Services, DIAgnosability, MONitoring and Diagnosis", *Proceedings of the International e-Challenges Conference*, The Hague, Oct. 2007.

[14] J.A. Chaudhry and S. Park. "AHSEN – Autonomic Healing-Based Self-Management Engine for Network Management in Hybrid Networks", C. Cerin and K.-C. Li (Eds), LNCS 4459, pp. 193-203, 2007.

[15] W. Tian, F. Zulkernine, J. Zebedee, W. Powley and P. Martin. "An Architecture for an Autonomic Web Services Environment". Proceedings of the Joint Workshop on Web Services and Model-Driven Enterprise Information Systems WSMDEIS (ICEIS 2005), May 2005, Miami, Fl., pp. 54-66.

[16] A. Lapouchnian, S. Liaskos, J. Mylopoulos, Y.Yu, "Towards Requirements-Driven Autonomic Systems Design", Proceedings of Design, and Evolution of Autonomic Application Software (DEAS) 2005, St. Louis, Missouri, May 21, 2005.

[17] XQuery 1.0: An XML Query Language. 2007, World Wide Web Consortium; May 12 2008; http://www.w3.org/TR/xquery/.

[18] Eclipse TPTP. 2008, The Eclipse Foundation; July 2 2008, http://www.eclipse.org/tptp/.

[19] *Apache Muse*. 2007, Apache Software Foundation,; April 25 2008; http://ws.apache.org/muse.

[20] *DB2 9*. 2007, IBM; July 2 2008, http://www-306.ibm.com/software/data/db2/9/.

[21] W. Tian, F. Zulkernine, J. Zebedee, W. Powley and P. Martin. "An Architecture for an Autonomic Web Services Environment", *Proceedings of the Joint Workshop on Web Services and Model-Driven Enterprise Information Systems WSMDEIS (ICEIS 2005)*, May 2005, Miami, Fl., pp. 54-66.