Towards Autonomic Web Services: Achieving Self-Healing Using Web Services

Sherif A. Gurguis
American University in Cairo
Computer Science Department
sgurguis@gmail.com

Amir Zeid
American University in Cairo
Computer Science Department
azeid@aucegypt.edu

ABSTRACT

Autonomic Computing was introduced to reduce the complexity of managing computing systems; however, the heterogeneous nature existing in most computing systems introduces some difficulty to achieve this target. Moreover, the notion of service as a computing component that seamlessly collaborates with other services in a loosely-coupled manner to perform complicated tasks was introduced by Service-Oriented Architecture (SOA); and then, fertilized by Web Services that added open standards to different roles and operations involved in a community of services; however, in order to gain the expected benefits of Web Services, the latter should be able to survive in normal and abnormal conditions. Our research aims at finding a hyper solution to that two-dimensional problem by allowing both Autonomic Computing and Web Services paradigms to lend each other their distinct features. First, Web Services lend Autonomic Computing the concept of platform-independency; second, Autonomic Computing lends Web Services the attributes providing self-management. The focus of this paper will be on how the self-healing autonomic attribute can be implemented and applied using Web Services.

General Terms

Management, Performance, Design, Reliability.

Keywords

Autonomic Computing, Web Services, Autonomic Web Services, Self-Healing Web Services, MAPE-cycle.

1. INTRODUCTION

IBM introduced the concept of Autonomic Computing by defining four attributes to enable computing systems to be self-managed with minimum human intervention [1]. The first attribute is *self-configuring* that is concerned with enhancing responsiveness by dynamically adapting to changing environments. The second attribute is *self-healing* intends to improve resiliency by avoiding disorders through discovery, diagnosis, and reaction to unexpected working conditions. *Self-optimizing* is the third attribute that aims at achieving better operational efficiency by maximizing the utilization of available resources. Finally, *self-protecting* deals with advancing information and resources security by anticipating, detecting, identifying, and protecting against different attacks [2]. In the context of autonomic computing, each computing component is divided into two integrated parts*: a managed element*, which is the resource itself, and *an autonomic manager*, which is responsible for providing autonomic behavior to that component [3]. An autonomic manager is implemented as a control cycle that is composed of four integrated phases: (1) *Monitor* collects information from the managed element, (2) *Analyzer* correlates reported events to identify the current situation of the managed element; and then, suggests some recovery actions in the case of failure, (3) *Planner* specifies what actions to be taken in order to recover from any malfunction while meeting business requirements and (4) *Executive* dispatches recommended actions in the managed elements [4]; the control cycle may be referred to as MAPE-cycle. The four phases share the knowledge they need, such as policies and logs in some repository. The vision of Autonomic Computing that has been discussed is shown in figure 1:

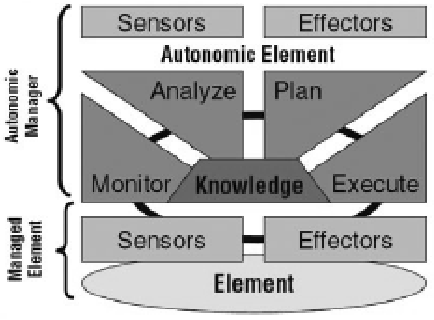

Figure 1. Autonomic Computing Vision [4]

In SOA, the functional unit is the service that is exposed to external users through publishing a set of defined interfaces. Therefore, SOA can be thought about as a normal extension of *Object-Oriented Architecture*, such that a service can be considered an object but with a special interface [2]. However, unlike *Object-Oriented Architecture*, SOA supports *loosely-coupled* relationships; and therefore, SOA supports applications that dynamically adapt to expected and unexpected changes. Web Services are services that are shared across the Internet by using open standards. For example, *XML* is used to support common understanding among computers; and therefore, allows seamless spanning across different domains. Web Services are shared by publishing the their interfaces in public service registries co that requesters of Web Services use service registries to find out the available Web Services that meet *their* needs; then, bind to their providers; and finally, invoke the required functions over the Internet. As shown in figure 2, roles and operations involved in the life-cycles of Web Services can be described with a triangle with roles represented by vertices and operations represented by sides.

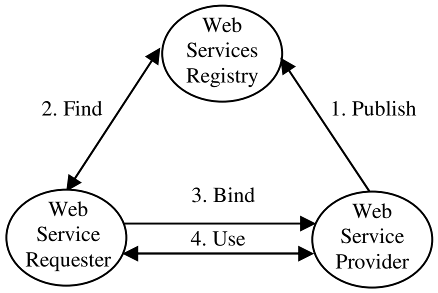

Figure 2. WS Triangle: Roles and Operations

All the Web Services' operations are based on exchanging messages using Internet communication protocols, such as *HTTP*. Moreover, XML is used for representing data that are exchanged among services in each operation. IBM considers the following standards the base technologies for developing Web Services: (1) *Simple Object Access Protocol (SOAP)* implements the bind and use operations by containing XML messages in a standardized envelope (2) *Web Services Description Language (WSDL)* implements the publish operations by defining abstract interfaces and bindings and (3) *Universal description, Discovery and Integration (UDDI)* implements the find-operations by providing a public registry that is accessed or queried for either searching or publishing [5].

2. RESEARCH TOPIC

Our research considers Autonomic Computing together with Web Services for the following reasons: First, autonomic computing provides the basics for achieving the notion of self-management. However, most of the modern computing systems are usually composed of solutions from different vendors. Because of this heterogeneous nature, there would be some difficulty to achieve the goals of autonomic computing. Second, Web Services provide the basics for building computing units that can be used dynamically by using open standards; however, Web Services will not be able to survive in unexpected operational environments if they rely on human intervention. Therefore, the promises and needs of both Autonomic Computing and Web Services have emphasized the relationship between them; therefore, they can be blended in a composite one, which is *Autonomic Web Services*.
According to our approach, Web Services could be divided into two categories. First, *Functional Web Services* are those services that provide computing functionalities over the Internet. Second, *Autonomic Web Services* are those services that encapsulate autonomic attributes to provide autonomic behavior over the Internet. Consequently, a functional Web Service can behave autonomically by discovering and using autonomic Web Services over the Internet without implementing autonomic attributes. This is illustrated in the figure below:

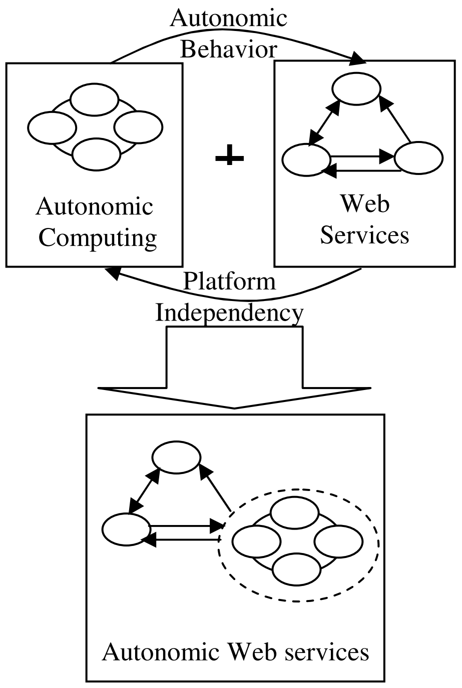

Figure 3. The Concept of Autonomic Web Services

Each attribute of autonomic computing can be implemented as a separate autonomic Web Service as follows: *Self-Configuring Web Service*, *Self-Healing Web Service*, *Self-Optimizing Web Service*, and *Self-Protecting Web Service*. In the heart of each autonomic Web Service, a MAPE-cycle is implemented; moreover, in order to allow each phase of the MAPE-cycle to be discovered and used separately over the Internet, the cycle is implemented as four integrated Web Services as follows: *Monitoring Web Service, Analyzing Web Service*, *Planning Web Service* and *Executing Web Service*. Through collaboration among the MAPE-cycle Web Services, autonomic Web Services perform their functions.

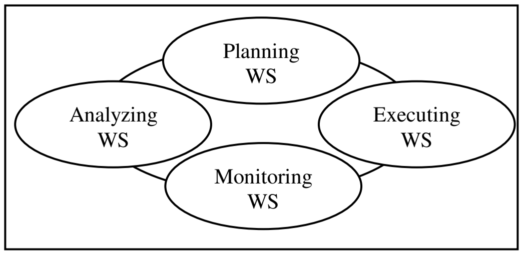

Figure 4. Web-based MAPE-Cycle

3. SELF-HEALING WEB SERVICE

This autonomic Web Service is responsible for providing the self-healing autonomic attribute to enable functional Web Services to discover, diagnose, and react to unexpected events that would lead to malfunctions. To achieve this target, different situations should be tracked and analyzed so that abnormal conditions can be identified. Moreover, in the case of failure, some actions should be initiated in order to recover from the failure while meeting the system requirements.

3.1. Problem Determination

A *Problem Determination Mechanism (PDM)* is used to automate analysis and correlation of logs data to help in solving problems. A typical PDM should be able to discover the root cause of problems; and then, recommends some actions that are initiated to correct those problems. Therefore, A PDM should adhere to the following steps while dealing with any problem: (1) identifying, (2) classifying, (3) diagnosing, (4) requesting some changes and (5) correcting [6]. However, the fact that each computing application reports logs representing different events and errors in its own way and format contributes to complicating the task of correlating reported logs. Therefore, in order to have a common PDM, logs should be standardized both syntactically and semantically.

3.2. Common Base Event (CBE)

Common Base Event (CBE) inherits the power of XML to unify both the syntax and semantic of the product-specific logs through converting events into XML-based messages. The following 3-tuple information is captured per each event: (1) the reporting component, (2) the affected component, and (3) the situation. Often, the reporting component is the component being affected by the situation [7].

After analyzing logs from different products, CBE concluded that different events that probably occur in computing systems can be categorized into eleven predefined situations; and one user-defined situation [7]. The set of predefined situations includes: *Start Stop*, *Connect*, *Request*, *Configure*, *Available*, *Report*, *Create*, *Destroy*, *Feature*, and *Dependency* situations.

For example, *FeatureSituation* denotes that some feature has become either available or unavailable on some component. Each of those situations has some parameters, such as *reasoningScope* that denotes whether the impact of the situation is internal to the affected component or it propagates to other components [8].

3.3. MAPE Cycle

The MAPE-cycle of the self-healing Web Service uses a *Diagnosis Engine* as its analyzing Web Service to recognize patterns exiting in the logs in order to realize that a specific problem has occurred [7]. The engine uses a *Symptom Database*, which is an XML file containing symptoms and recovery actions. By looking up the symptom database, one or more directives can be found such that following those directives should eventually lead to recovering from the current situation.

Once the diagnosis engine has found some recovery actions, a *Rule Engine* is used as the MAPE-cycle's planning Web Service to determine which actions can be taken according to the system policies [7]. The rule engine uses a *Policy Database*, which maintains high-level policies, to match the recommended actions by the diagnosis engine against policies in order to ensure that no action will have a negative impact on critical processes.

A typical record in the Symptom Database looks like the following:

```
<symptomId>
            <eventMessage>
                        <explanation>
                                    <directives>
                                    .
                                    .
                                    <\directives>
                        <\explanation>
            <\eventMessage>
<\symptomId >
```

Figure 5. Structure of the Symptom Database

The figure below shows the operation of the MAPE-cycle used by the self-healing Web Service.

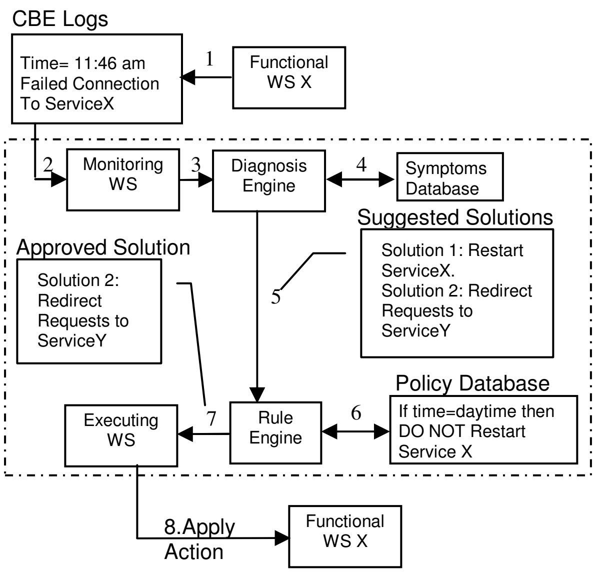

Figure 6. Functionality of Self-Healing MAPE-Cycle

3.4. Achieving Synchronization

Self-healing Web Service can achieve synchronization among the Web Services of its MAPE-cycle by implementing a *Notification Web Service* that adheres to the event-based architecture. Each Web Service in the self-healing MAPE cycle that is concerned with the occurrence of some events subscribes with the notification Web Service to be notified every time those events occur. Whenever an event of significance occur, it is published so that interested Web services in the MAPE-cycle are notified.

In the figure below, the monitoring Web Service publishes events as they occur, the diagnosis engine publishes suggested corrective actions, and the rule engine publishes the corrective action to be taken. Moreover, the diagnosis engine subscribes to be notified about specific events, the rule engine subscribes to be notified about suggested corrective actions, and the executing Web Service subscribes to be notified about specific actions to be applied.

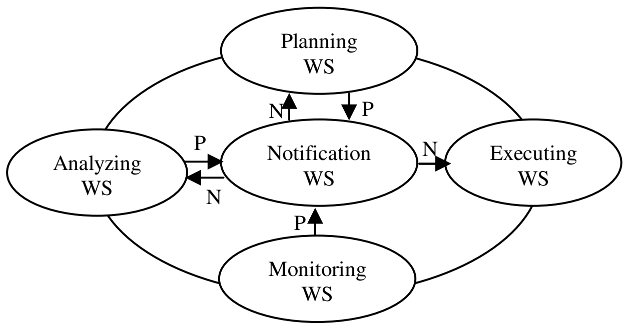

Figure 7. Functionality of Notification Web Service
P =Publish – N = Notify

3.5. Using Self-Healing Web Service

Suppose that WS-X is a self-healing Web Service that has been developed. WS-X should be published into an XML-based registry by registering the interfaces of the MAPE-cycle Web Services composing WS-X. In addition, suppose that WS-Y is a functional Web Service that does not implement the self-healing attribute. The developer of WS-Y considers the situation, in which, the logs file reads abnormal measures such that WS-Y locates a Web Service like WS-X to analyze the situation and provide some recommendations. Once WS-X has been located, WS-Y is bound to it and invokes the analyzing Web Service of WS-X. WS-Y enables WS-X to access its logs file as well as its policy database if there is any. WS-X uses this information to provide an action, which is applied to WS-Y through the executing Web Service of WS-X.

3.6. Developing and Testing Environment

IBM provides the *Emerging Technologies Toolkit (ETTK)* that can be used in building a proof-of-concept prototype. The toolkit contains a logs adapter called *Generic Log Adapter for Autonomic Computing (GLA)* that translates legacy logs into CBE format. Moreover, *Log and Trace Analyzer (LTA)* can be used to implement the diagnosis engine as it provides the means for performing a root-cause analysis by correlating events and showing the interactions among the logs that contributed to the problem. In addition to the autonomic computing tools, IBM also provides the tools required for building Web Services. In our research, we use WebSphere Application Developer (WSAD) for developing Web Services and WebSphere Application Server to deploy the developed Web Services so that they can be exposed to public usage. Also, we use some Web Services testing tools to simulate a real operational environment by applying some stress on Web Services and to measure some parameters of Web Services, such as response time. In addition to developing the self-healing Web Service, a simple functional Web Service is being developed to serve as a case study for testing. That Web Service will be forced to behave badly with the aid of the stress tools; and then, inspecting how the self-healing Web Service would help that Web Service to recover from such behavior. Therefore, we can test the concept of Autonomic Web Services.

4. POSSIBLE APPLICATIONS: EVOLVING LEGACY SYSTEMS

70-80% of the data used in business still reside on legacy repositories, which means that almost the same percentage of the business logic also resides on legacy systems [9]. Therefore, redeveloping legacy applications is, obviously, not the good choice in order to apply new technologies to your business. Consequently, enabling potential legacy systems to adopt both the Autonomic Computing and Web Services paradigms without the need for redeveloping them is a critical consideration. A legacy system can be wrapped by WS wrappers so that the wrapped system will be enabled to be discovered and accessed from any location connected to the Internet using Web protocols. Moreover, legacy logs can be converted into CBE logs by using a *logs-adapter*. Eventually wrapped legacy systems can be enabled to behave autonomically by discovering and using the autonomic Web Services over the Internet.

5. CONTRIBUTION

Autonomic computing attributes can be implemented as Web Services in order to overcome the problem of heterogeneous computing systems. By adhering to the Web Services triangle, autonomic Web Services can be published into a public registry of Web Services. Therefore, other Web Services or wrapped legacy systems that lack embedded autonomic behaviors can discover, bind-to, and use those autonomic Web Services in order to borrow the autonomic behavior. Consequently, Web Services will be able to survive in different working conditions without depending on human intervention that hinders achieving the expected benefits of Web Services.

6. RELATED WORK

Many research efforts contribute to the topic considered by our research. In the context of achieving self-healing management among heterogeneous applications, Gary Dudley et al proposed a hyper solution that provides self-healing to an environment of applications and network devices [10]. In the context of evolving legacy systems using WS, Al Williams proposed an approach that depends on adapting legacy messages into SOAP messages [11]. Moreover, Shalil Majithia et al proposed a framework for automating the process of collaboration among services in an SOA environment [12].

7. FUTURE WORK

Only the self-healing autonomic attribute is being considered by our research in this stage; implementing the proposed MAPE cycle of the self-healing Web Service as well as the test functional Web Service is currently conducted. Afterwards, the remaining autonomic attributes will be considered for implementation respectively to be provided in the same way as self-healing attribute is.

8. REFERENCES

[1] Grid Technology Partners, "Autonomic Computing: Characteristics of self-management IT systems", 2003.

[2] A.G. Ganek and T.A. Corbi, "The dawning of the autonomic computing era", IBM Systems Journal, VOL 42, NO 1, 2003.

[3] IBM web site, http://www.ibm.com/developerworks/.

[4] Jeffery O.Kephart and David M.Chess, "The vision of autonomic computing", IEEE, January 2003.

[5] Chris Nelson and Willy Farrell, "Transforming legacy applications with Web Services", http://ibm.com/developerworks/, May 2002.

[6] IBM, "Understand problem determination for autonomic computing", http://ibm.com/developerworks/.

[7] IBM Autonomic Computing, "Automating problem determination: A first step toward self-healing computing systems", October 2003.

[8] David Bridgewater, "Standardize messages with the Common Base Event model", http:/ibm.com/developerworks/, Feb. 2004.

[9] "Legacy Transaction Integration in a Service-Oriented Architecture", Red Oak Software, November 2003.

[10] Gary Dudley et al, "Autonomic Self-Healing Systems in a Cross-Product IT Environment", proceedings of the international conference on autonomic computing (ICAC'04), IEEE, 2004.

[11] Al Williams, "Bridging Legacy Systems and the Web", http://www.newarchitectmag.com/documents/, July 2002.

[12] Shalil Majithia et al, "A framework for Automated Service Composition in Service-Oriented Architectures", 2004.