Fault Tolerant Autonomic Computing in Web Services

Eric O'Laughlen
Professor Yashwant K. Malaiya, Department of Computer Science, Colorado State University
eric@olaughlen.net

***Abstract*—Web services enable networked systems, typically based upon the World Wide Web (WWW), and their related services, to interoperate. Moreover, Service-Oriented Architecture (SOA), Simple Object Access Protocol (SOAP), Extensible Markup Language, Remote Procedure Call (XML-RPC), and Representational State Transfer (REST) are architectural styles of networked systems, upon which Web Services are implemented. Given the growing dependence and criticality of these services for users, fault tolerance plays an increasingly important role in providing usable and reliable services. With the advent of autonomic computing, or self-managing computer systems, research groups have been pursuing methods of combining fault tolerant autonomic approaches with Web services. This paper reviews research and progress to date of those autonomic methods and researches Web services architectural style support of autonomic methods.**

I. Introduction

The World Wide Web (WWW or Web) commenced in the early 90's and today consists of numerous networked systems enabling interactive, shared communication [1]. As the public inter-network evolved, website applications began communicating with each other; this prompted the need for shared programmatic interfaces [2]. Today, these programmatic interfaces, or Web services, are at the epicenter of a shift of the Web from websites and published HTML-based pages to distributed services and applications. This shift, labeled Web 2.0 [3], encompasses a wide range of initiatives, which aim to deliver native, desktop application-like capabilities directly from websites via Web-based services.

However, as Web service technology has evolved, system complexity and cost has increased as well. As evidence of this trend, a Recovery Oriented Computing (ROC) group released a study on the total cost of ownership (TCO) of cluster-based services, which encompass Web services. The study estimated that the ratio between software and hardware purchases and other TCO factors (e.g., labor) was 3.6 to 18.5. Moreover, a survey, also associated with the study, showed that one third to one half of TCO was spent preparing or recovering from failures [4], [5]. Because of these TCO challenges, computing organizations have launched research initiatives to study and design self-managing computing mechanisms. The goal of self-managing systems is to reduce information technology (IT) management and administration costs.

One organization, dedicated to reducing IT and Web service complexity and costs like the ROC, is IBM. In 2001 the company published a manifesto and announced its autonomic computing vision [6]. The manifesto summarized the state and called for systems and services that would support self-correcting measures, including self-configuring, self-healing, self-optimizing, and self-protecting ones. According to the manifesto, self-configuring measures would address system setup and configuration tasks. Self-healing mechanisms would address fault tolerance and system redundancy and availability. To address challenges related to system operational efficiency and performance, self-optimizing methods would assist. In the realm of hardened security and system integrity, self-protecting methods would improve current approaches. IBM's vision for these measures, collectively called *self-CHOP* [7] (Fig. 1), would mask complexity and reduce cost for IT professionals with sophisticated, adaptive technologies.

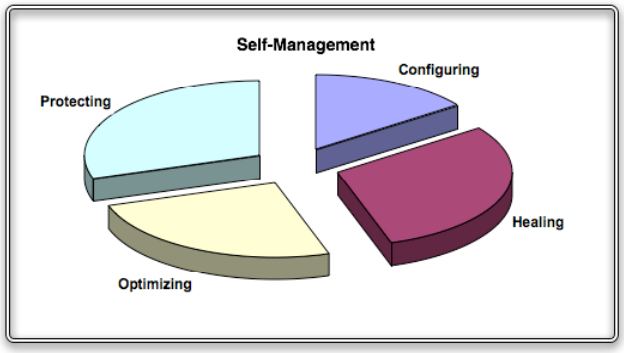

Fig. 1. Attributes of Autonomic Computing Initiative [8].

However, to obtain this vision systems would need to be more knowledgeable of their internal and external computing environment. With more awareness and knowledge, systems could then differentiate between healthy and unhealthy activity and take action, if necessary. However, before contemplating more knowledgeable systems, there are also stylistic considerations. Generally, Web services are based upon principles of loosely coupled interfaces and underlying technologies, commonly called a service-oriented architecture (SOA). However, there are varying philosophies of how to obtain loose coupling. For example, some development groups advocate the use of an extensible markup language (XML) protocol called SOAP (simple

©2007 Eric O'Laughlen. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

object access protocol), which decouples interface methods and other details from underlying protocols; however, others argue in favor of the existing representational state transfer (REST) style, which describes the architectural style of the Web [9].

The purpose of this paper is to review autonomic technologies and approaches, specifically self-healing ones, involving Web services. With the review the paper also examines the primary architectural styles of Web services and how those styles affect autonomic approaches. Moreover, the paper reviews industry progress with autonomic initiatives. The motivation for the paper is to provide informational aid to research groups and to contribute to ROC-type group initiatives that improve reliability, availability, and TCO in Web services. The organization of the paper includes the following sections:

II. Autonomic Computing

A. Background

An autonomic computing system, or a self-managing system, describes a computing system that is aware of itself and of its environment; a system that can dynamically adjust to its environment, if need be, like the human autonomic nervous system or other biological systems [10]. Just as biological systems obtain and maintain *homeostasis*, a phenomenon within biological systems that strives toward equilibrium and *health*, autonomic computing systems aim for internal equilibrium regardless of external system changes [11], [12]. Autonomic computing efforts work to embed complex, adaptive technologies into computing systems in order to reduce the manual management of that complexity. As mentioned, IBM's manifesto in 2001 prompted heightened awareness in the field by summarizing the complex and costly state of information technology and by calling for systems and services that would support self-correcting measures.

Although the manifesto and vision from IBM is relatively recent, autonomic computing has its foundation in years of science and technology evolution [13]; it drew, and continues to draw, from numerous areas within science and the industry. For example, hardware redundancy with disks, specifically RAID, is one area. RAID solutions (i.e., redundant array of inexpensive, or independent, disks) provide fault-tolerance and redundancy for disk drives and were available years previous [14]. Furthermore, software techniques, like virtualization and managed virtual machines, have been used for many years to improve reliability and availability of software services [15]. Although there are myriad examples, the initiative was unique in that it focused on interoperability of self-management technologies for enterprise Web services.

Within the *self-CHOP* framework autonomic approaches vary; however, Lapouchnian, Yu, Liaskos, and Mylopoulos outline three basic paradigms for creating autonomic systems [16]: isomorphic mapping of system behaviors to system configurations, system planning and augmentation, and evolutionary approaches like those found in biology. Isomorphic configuration and behavior modeling encompasses a requirements-driven approach to software behavior, which they advocate. Autonomic approaches, that utilize system planning and augmentation, delegate tasks to autonomic elements, also called external, intelligent agents. A biological approach incorporates theoretical approaches from machine learning, robustness principles, and statistical analysis to detect and determine appropriate actions and corrective measures [7]. This paper's research focus technologies and models that use the latter two approaches.

B. Self-Healing Terminology

When discussing self-healing systems and *self-CHOP*, many problems in each of the areas could be considered self-healing in nature [17]. For example, if a particular system is performing sub-optimal due to an incorrectly sized cache, and an autonomic system identifies and modifies the cache for better performance, the system *self-healed* the initial problem. However, for this purposes of this paper's research the hypothetical scenario will not be considered a *fault tolerant, self-healing system*, primarily because the system continued to operate with or without the cache size modification. The paper focuses upon self-healing models and technologies that detect and remedy defects in hardware, errors in software, and faults in operation in either hardware or software [18]. Furthermore, since the research addresses autonomic computing methods in Web services, there is an emphasis on interoperable, software-based protocols and techniques.

C. Autonomic Computing Architecture

Along with its autonomic initiatives, IBM published an autonomic computing architectural blueprint [19] and Autonomic Computing Toolkit [20], outlining a paradigm and framework for problem determination and *self-CHOP* capability. Although there may be other ways to enable autonomic systems, as alluded earlier [16], since the toolkit was the first of its kind [21], the research in this paper

focuses on its models and architecture. At the heart of the blueprint and toolkit is concept of an autonomic element (AE). An AE comprises of managed elements (or resources) and autonomic managers that operate as a part of a Monitor, Analyze, Plan, and Execute, control loop. The control loop is commonly referred to as the MAPE cycle, or MAPE-K if *knowledge* is emphasized (see Fig. 2) [8].

IBM envisioned that autonomic elements would be system *agents* and that autonomic systems would consist of multiple agents, or *multi-agent-based systems*, built upon Web services or OGSA (Open Grid Services Architecture [22]) infrastructure [7]. There is one important aspect of knowledge within the cycle: instead of using a definition that implies only logical inference and compilation of new knowledge from experience, IBM's autonomic computing defines knowledge as any structured data or information that is a part of the system, which would include information from system logs, from process state, and from scheduling [23].

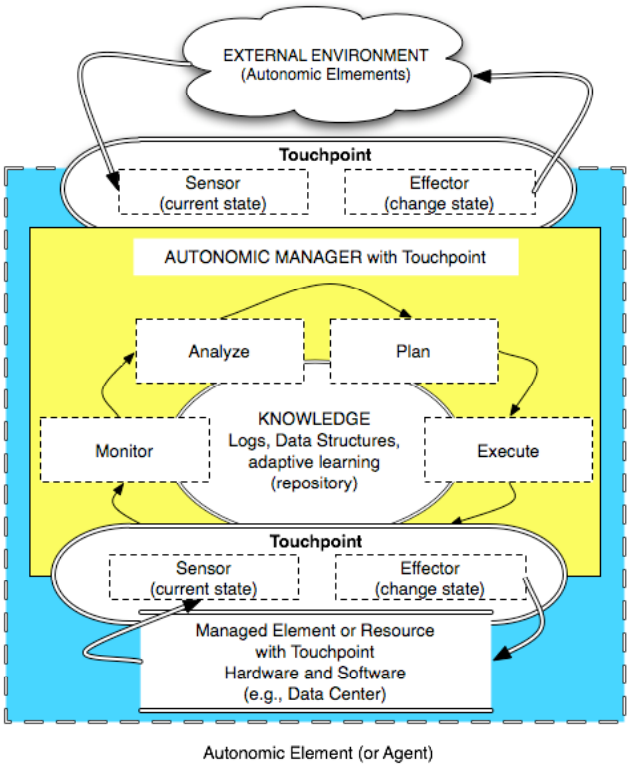

Fig. 2. Autonomic Element and MAPE model [19].

With this form of knowledge the control loop acquires information about the computing resources, and external environment, and then applies self-corrective algorithms as necessary. Managed elements contain sensors to determine states of the autonomic element and effect change in state, if necessary, with the use of effectors. Autonomic elements contain sensors and effectors as well in order to interact with its environment [19]. Using the blueprint, IBM showed the feasibility of the model by implementing a system for log analysis and remediation. Using autonomic elements and managers, and a common base event (CBE) format for aggregating log information, the system could: communicate and collect *knowledge*, characterize events, determine problems, compare events against accepted policies, and issue self-healing measures as necessary [8].

D. *Autonomic Computing Toolkit*

Using the MAPE-cycle, IBM released the *Autonomic Computing Toolkit* in 2004. It was the first autonomic toolkit released in the corporate world to introduce autonomic computing to IT infrastructures [21], so the paper provides a brief review of it. Before reviewing each component, it is important to understand that IBM assigns autonomic computing maturity levels to its solutions. There are five levels total and they progressively work toward full automation [20]:

- Level 1: Basic – requires staff for intervention
- Level 2: Managed – requires staff for analyzing and planning
- Level 3: Predictive – requires staff for selecting recommendations and implementation
- Level 4: Adaptive – requires staff for policies and generate plans automatically
- Level 5: Autonomic – no staff required; system-wide policy

The toolkit contains various components that operate between maturity levels 2 and 3: an Autonomic Management Engine (AME), the Generic Log Adapter (GLA), the Log and Trace Analyzer (LTA) tool, and an Integrated Solutions Console (ISC). There are other technologies; however, these provide the majority of capability.

The AME includes capabilities for the MAPE-model control loop and typically runs within a Java Virtual Machine (JVM) and Web server (i.e., on the IBM WebSphere platform). The GLA creates a touch-point for log-based data and translates log messages into CBE format. The GLA encompasses a client and server-side component that works within IBM WebSphere products and an integrated development environment (IDE) called Eclipse (see eclipse.org), a client based utility. The LTA provides certain MAPE functions by covering the analysis and monitoring aspects; it requires Eclipse as well. The ISC provides a mechanism to view events using IBM WebSphere Portal. Each aspect aligns with an autonomic element and each serves a purpose within the autonomic computing framework (see Fig. 3) [20].

As a hypothetical example, the toolkit user guide describes a situation where the toolkit and technology could be used to report upon a database failure. If the database reports a failure within a log (i.e., has *stopped*), then the autonomic manager, along with other facets of the framework, could issue a fault tolerant, self-healing action to restart the database and product. Other examples of use include solutions for automated installation and deployments, which both have guides for further information [20].

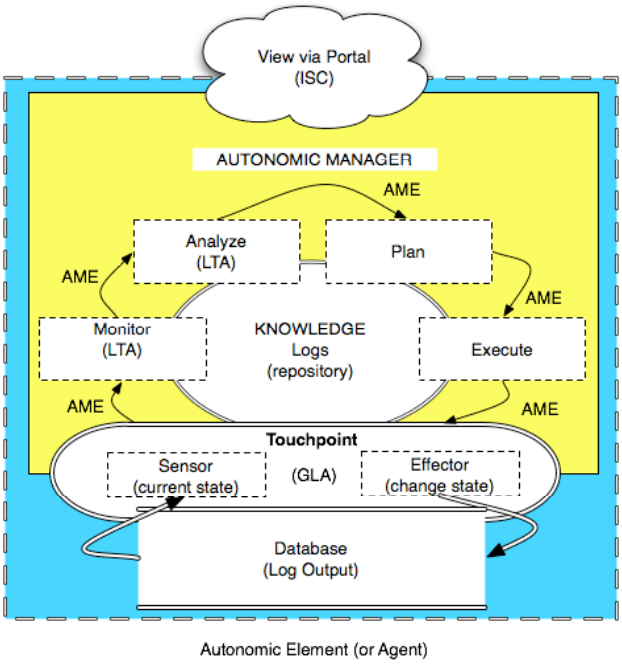

Fig. 3. Autonomic Computing Toolkit and Functions [19].

III. WEB SERVICES

A. Background

In order to review autonomic computing mechanisms in Web services and their effects, it is important to review the evolution of the protocols and technologies that have led to current Web service approaches. Web services evolved out of a desire of Web communities to share data and services between heterogeneous systems. With the advent of the WWW, the underlying architecture was designed to be open with publicly available specifications for implementation [24]. Although the open nature of the global network accelerated adoption and usage of the information system, it was not initially designed to accommodate shared, programmatic interfaces as imagined today. For example, initial specifications did not provide standards for remote service discovery, for certain service policies or contracts, and for programmatic object binding and encoding. Web service standards and technologies aim to address necessary extensions.

B. Service-Oriented Architecture

A network-based architectural style is a coordinated set of boundaries and relationships between elements. Network-based architectural styles are not specifications; they are constraints that classify how a networked architecture behaves: how it processes data; how it connects between data processors; how it manages state [25]. Similarly, a web service architectural style encompasses a particular network-based architecture while also determining boundaries concerning its shared programmatic interface: what the interface does and how the interface interacts. Web service architectures vary; however, they share goals for loosely coupled interfaces from implementations and for service-component reuse. Furthermore, Web services leverage open standards for network and data exchange for optimum interoperability. Systems that adopt open network-based architectural styles and embrace Web service goals are labeled service-oriented architecture (SOA) [26].

C. Web Services Styles

Within a SOA-based Web service, multiple complementary styles and specifications exist. However, as a fundamental tenant, they all use the HTTP protocol for transfers. Initially, XML provided a way to apply metadata to text documents via HTTP in the late 90's. Soon XML specifications and technology evolved into the development of XML-RPC (or XML remote procedure call); it enabled Web sites to execute methods and to transfer simple data objects. Eventually, the XML-based SOAP specification evolved out of XML-RPC specifications and efforts [27]. These developments collectively lead to the Web service specifications and styles that exist today: REST plus XML, XML-RPC, and SOAP. The following sections review briefly each approach. There are other derivative styles (e.g., the use of Asynchronous Javascript and XML (AJAX)), but these derivative styles still encompass fundamental approaches from the other styles [28].

D. Representational State Transfer (REST)

Fielding coined the term Representational State Transfer (REST) when he described the architectural style of the WWW. He described the WWW as a virtual state-machine of web pages in which users invoke (or *click* in browser parlance) hyper-links to invoke state transitions. The REST architectural style encompasses a number of constraints [25] (see Table I). An important facet of the REST virtual state machine involves the concept of a resource and its reference by a universal resource identifier (URI). For example, when one visits a hypothetical web page like `http://host/resource` inside a WWW browser, the resource retrieved represents a particular content type. The content type of the resource may be a text-based article, a raster graphic, a video encoded file, or some other digital content type. However, the resource could also represent varying content types within the same URI based upon request parameters or metadata. In addition to this multiplicity, each request and response invokes a stateless transfer of information, independent of other transactions. Although the REST style is not dependent upon a particular protocol like HTTP, it represents ideals encapsulated within protocol and the Web, of which Fielding was a contributor. An important differentiator between the REST style and other architectural styles relates to the constraints it defines: REST promotes uniformity by method and resource constraints while allowing multiple domain-based identifiers. These constraints optimize the network-based system, while also allowing extensibility of the underlying architecture.

TABLE I
REST CONSTRAINTS [25]

Term	Definition
Client-Server -	Separation of interface from data storage
Stateless -	Inclusion of all information necessary to understand a request
Cache -	Directives concerning stored copies of resources
Uniform Interface -	Software engineering generality and uniformity between components
Layered System -	Independence of hierarchy to improve scalability and to reduce complexity
Code-On-Demand -	Optional constraint that includes expanded client capability through downloadable code

E. REST Terminology And Web Services

Before comparing and contrasting REST to other Web-based architectural styles, it is important to note that not all Web service groups categorize REST-based systems the same. For example, a system that utilizes HTTP cookies,

```
[Newton@eolaughlen /]telnet api.flickr.com 80
Trying 68.142.214.24...
Connected to www.flickr.vip.mud.yahoo.com.
Escape character is '^]'.
GET /services/rest/?method=flickr.test.echo&api_key=XX HTTP/1.0
Host: api.flickr.com

HTTP/1.1 200 OK
Date: Fri, 04 May 2007 01:06:27 GMT
Server: Apache/2.0.52
Content-Length: 149
Connection: close
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
<method>flickr.test.echo</method>
<api_key>XX</api_key>
</rsp>
Connection closed by foreign host.
[newton@eolaughlen /]
```

Fig. 4. Example REST-based transaction using Flickr Services (api_key represented by **XX** instead of actual 32 hexadecimal-ASCII characters)

an HTTP header used to pass information about a client connection, to maintain state may is strictly not REST-based. Because a strict REST-based style requires statelessness. However, since uniformity of interfaces and of other REST-based constraints is followed, the system is classified as REST-based, or *RESTful*. In other words, the classification of REST may be based upon a bias and philosophy. Using a REST-based philosophical classification, many Web services offer REST-based programmatic interfaces using HTTP and URIs for *methods*, query strings for *method parameters*, and HTTP and XML (i.e., application specific) for *method return values*. An example of a REST-based Web service call, may be demonstrated using *telnet* and Flickr Services [29] and the flickr.test.echo method (see Fig. 4). Each API (application programming interface) on the service requires an authorization token for use, called the *api_key*, which is left out on purpose.

F. XML-Remote Procedure Call (XML-RPC)

XML-RPC is an XML-based protocol for executing procedures over a distributed network, which is conventionally used with the HTTP protocol and with Web-based technologies and networks [30]. Basically, its goal is to provide an easy protocol and mechanism to execute methods within a Web-service and return a limited number of data types: int, boolean, string, double, dateTime.iso8601, base64-encode binary data, struct, and array. A simple example of an XML-RPC message may be demonstrated using *telnet* and the Flickr Service [29] with the flickr.test.echo method (see Fig. 5).

```
[newton@eolaughlen /]telnet api.flickr.com 80
Trying 68.142.214.24...
Connected to www.flickr.vip.mud.yahoo.com.
Escape character is '^]'.
POST /services/xmlrpc HTTP/1.0
Host: api.flickr.com
Date: Fri, 04 May 2007 00:21:53 GMT
User-Agent: HelloWorld 1.0
Content-Type: text/xml; charset=utf-8
Content-Length: 350

<?xml version="1.0" encoding="utf-8" ?>
<methodCall>
<methodName>flickr.test.echo</methodName>
 <params>
<param>
<value>
        <struct>
        <member>
        <name>api_key</name>
        <value><string>XX </string></value>
        </member>
        </struct>
</value>
</param>
</params>
</methodCall>

HTTP/1.1 200 OK
Date: Fri, 04 May 2007 02:48:09 GMT
Server: Apache/2.0.52
Content-Length: 231
Connection: close
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8" ?>
<methodResponse>
        <params>
                <param>
                        <value>
                                <string>
&lt;api_key&gt;XX&lt;/api_key&gt;
                                </string>
                        </value>
                </param>
        </params>
</methodResponse>
Connection closed by foreign host.
[newton@eolaughlen /]
```

Fig. 5. Example XML-RPC transaction using Flickr Services (api_key value represented by **XX** instead of actual 32 hexadecimal-ASCII characters).

G. Simple Object Access Protocol (SOAP)

SOAP is an XML-based protocol to exchange structured data with stateless on-way messages over a network, typically with HTTP, but HTTP is not required for the underlying transport [31]. In some Web-services groups SOAP is synonymous with Web services [32]. Its origins have root in distributed application technologies like Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), and Java RMI (Remote Method Invocation). Although SOAP is a

message format for method invocation, that is more analogous to protocols like Internet Inter-ORB Protocol (IIOP) for CORBA [33], it was viewed as a mechanism to standardize and popularize distributed computing technologies over Web-based networks. A simple example of a SOAP message, using HTTP, may be demonstrated using *telnet* and the Flickr Services [30] and the flickr.test.echo method (see Fig. 6).

```
[newton@eolaughlen /]telnet api.flickr.com 80
Trying 68.142.214.24...
Connected to www.flickr.vip.mud.yahoo.com.
Escape character is '^]'.
POST /services/soap HTTP/1.0
Host: api.flickr.com
Content-Type: application/soap+xml
Content-Length: 400

<?xml version="1.0" encoding="UTF-8" ?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
            xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/1999/XMLSchema" >
<s:Body>
<x:FlickrRequest xmlns:x="urn:flickr">
<method>flickr.test.echo</method>
<api_key>XX</api_key>
</x:FlickrRequest>
</s:Body>
</s:Envelope>

HTTP/1.1 200 OK
Date: Fri, 04 May 2007 14:49:57 GMT
Server: Apache/2.0.52
Content-Length: 414
Connection: close
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope
        xmlns:s="http://www.w3.org/2003/05/soap-envelope"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema"
>
        <s:Body>
        <x:FlickrResponse xmlns:x="urn:flickr">
&lt;method&gt;flickr.test.echo&lt;/method&gt;
&lt;api_key&gt;XX&lt;/api_key&gt;
                </x:FlickrResponse>
        </s:Body>
</s:Envelope>
Connection closed by foreign host.
[newton@eolaughlen /]
```

Fig. 6. Example SOAP transaction using Flickr Services (api_key value represented by **XX** instead of actual 32 hexadecimal-ASCII characters).

H. Web Services: Predominant Style

Over the past few years, and even today, there is debate about which style is better than another. On the one hand the Web was modeled after the REST style, so there is little disputing its scalability and extensibility. As such, REST components point to its simplicity and to the Web's success in using it. Others point to the need for additional layers of abstraction in order to transfer data structures, and ease of implementation, like in XML-RPC. Large enterprise infrastructure corporations, like Microsoft and IBM, continue to encourage SOAP-based mechanisms as the appropriate technology for programmatic interface abstraction and for network transfer independence. However, what may be the most telling are what architectural styles Web service providers are providing in APIs (application binary interfaces) and what developers are supporting. As observed from a few researched, REST interfaces are supported by most; however, SOAP is the second most supported (see Table 2).

TABLE 2
WEB SERVICE ARCHITECTURAL STYLES

Web Service	REST	XML-RPC	SOAP	Other
Amazon [34]	X		X	
Yahoo! Flickr [35]	X	X	X	
Ebay [36]	X		X	
Google [37]	X (*)		X	(*) AJAX
YouTube [38]	X	X		

According to the information available, typically REST and SOAP interfaces are both supported by the same vendor. Which style and API a developer chooses, or should choose, will likely be based upon the technologies and tools used for integration (i.e., assuming there is choice of API). For example, if one were using Microsoft or IBM technologies, then SOAP may be a logical choice, given the tools and development environment. However, if one is using another development environment, and prefers an HTTP-based interface, then REST would likely be a logical choice. However, there is some evidence that developers prefer REST to other styles [39].

IV. WEB SERVICES: AUTONOMIC CHALLENGES

There are numerous challenges when considering autonomic computing and fault tolerance in Web services. If one takes the log-based model of IBM as an example, there is an old adage in IT circles: *You cannot manage what you cannot measure*. One of the first challenges involves the visibility of metrics and events within existing systems. With the author's experience within software engineering many developers use assertions and other debug capabilities to capture runtime errors. However, upon release of the software these assertions and debug capabilities are conventionally disabled, greatly reducing the amount of information that is useful in diagnosis and remediation. Retrofitting this information and autonomic capabilities into an existing, or legacy, system may be problematic and cost-inhibitive to do.

Furthermore, software error reporting is commonly either sparse or crafted within a particular domain, the domain of the engineer writing the code. If a reported problem is too granular then it may not have any understandable context outside of that domain. For example, a low-level domain exception, like a network data transfer error, may not yield anything useful within a global context base, for instance, of an unplugged cable [10]. Moreover, even descriptive messages of similar errors, within the same context, may look different. For example a hypothetical example could involve a programmers use of a system call to allocate memory. When the allocation routine fails it may return NULL and set an error code to NOMEM; however, the programmer may log a non-descriptive message in a system log like: *function()*

returned NULL. The disparate error codes and messages complicate visibility and interpretation.

In the scientific community there are various challenges as well. Along with the need for standards to collect and to aggregate data produced by a system, new models and algorithms may be necessary to mine, analyze, and determine appropriate action [7]. For example certain supervised, machine learning models and algorithms for text-based classification use iteration with trial and error in order to converge upon a solution. Although extensive training may be acceptable in some cases (e.g., where an eventual threshold is obtained), in a dynamic computing environment training and latency in obtaining a certain threshold may not be feasible. Furthermore, configurations and parameters change frequently, which may not be conducive to certain machine learning models and algorithms as well [10]. Like other text classification fields (e.g., mail and search), a combination of statistical and adaptive methods may prove most useful while research continues to surface new models and approaches.

Another challenge, involving the entire industry, is interoperability. Even if engineering and scientific groups are able to solve significant challenges within their respective fields, there is no guarantee that those solutions will be interoperable. There are numerous examples of this outcome throughout computing history including: network protocols, digital rights management, operating system frameworks, etc. Moreover, the interoperability challenges affect groups that may not be considered autonomic in nature (e.g., security and privacy) [10]. Solutions to interoperability will likely need to be open, publicly available to computing groups, and will likely need to be built collaboratively through consortiums. One of the more successful methods of achieving collaboration in recent years has been through the use of standards groups involving industry leaders.

V. Web Services: Autonomic Progress

Since IBM's manifesto and announcement, each major IT provider today has its own counterpart. For example, Microsoft Corporation supports autonomic capabilities with its Dynamic Systems Initiative (DSI) [40]. Hewlett Packard Development Company supports autonomic capability with its Adaptive Infrastructure (AI) efforts [41]. Sun Microsystems, Inc. supports self-management capabilities within technologies with its N1 initiatives [13]. These efforts show how the industry, and consumers, has embraced the notion of reduced complexity and self-management with autonomic computing concepts and technologies.

Not only has each forged autonomic initiatives, but also each has been collaborating on future standards. For example, a cross-company working group, that includes Microsoft, HP, Sun, IBM, and others, recently submitted a specification to the World Wide Web Consortium (W3C). The specification for modeling Web services, called the Service Modeling Language (SML) [42], proposes to make descriptions and management of network and computing assets easier. Its submission to the W3C aims to make the specification open for public implementation and to make it broadly adopted. The specification is based upon XML Schema and Schematron in order model complex IT system structures, constraints, and policies. Moreover, there are numerous other industry standards available (see Fig. 7).

Individually, within corporative initiatives, there has been progress as well. For one example, Microsoft Corporation recently released a whitepaper reviewing its SDI technologies. As a brief synopsis of their progress, Microsoft created technology strategies, delivered ITIL-based (Information Technology Infrastructure Library) guidance, and worked with industry leaders in order to enable dynamic systems. Moreover, they outlined dynamic systems features in Windows Vista, Virtual Server and Virtual PC, and in Visual Studio [40]. Microsoft is also incorporating SML into Windows Server (now called *Longhorn*) along with other dynamic systems technology slated for 2008.

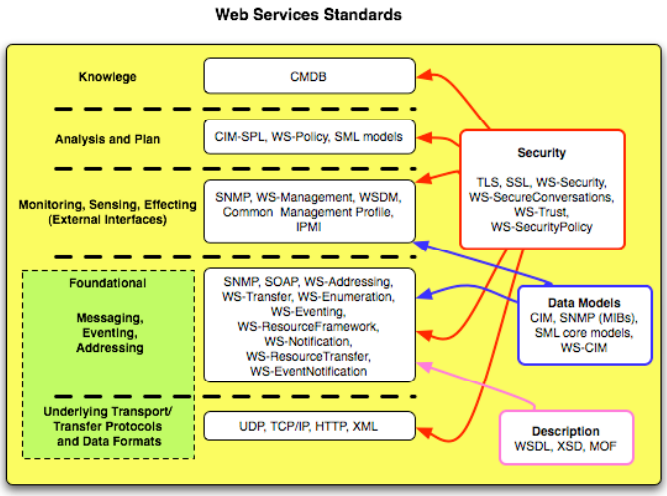

Fig. 7. Web Services Standard [43].

HP released a report [41] indicating their progress on their Adaptive Infrastructure initiatives while committing to bringing lower-cost, automated, pooled data center to customers. On the product side HP's BladeSystem c-Class for Microsoft Exchange Server improved reliability for e-mail applications with automated virtualization and provisioning technologies and tools. Moreover, with their OpenView Enterprise Manager Starter Kit, a technology organization may manage computing events for a holistic view of service health. Also, according to the report, HP also made progress with their Virtual Server Reference Architecture and the Shared Infrastructure Utility for Test and Development Offering.

As another example, IBM, since its manifesto, has engaged multiple hardware and software-related autonomic initiatives. On the hardware side, it created Active Memory as an example, that uses parity checking and error-correction code (ECC) algorithms to ascertain and fix problems. Furthermore, the corporation fabricated CPU error detection and correction

techniques, which were incorporated into its Blue Gene line of super computers [13]. On the software side, the eLiza project, now a part of IBM's Autonomic Computing initiative [44], supported, and continues to support, various mechanisms for fault tolerance and recovery with its IBM Global Services Enterprise Workload Manager and Electronic Server Agent software. These services and solutions utilized error messaging and Internet-based transfers to detect and remedy problems, using software and hardware fault tolerant mechanisms.

VI. Autonomic Computing: Future

IBM and other corporations and research groups are making good progress on the vision of autonomic systems. This is apparent in technical papers, toolkits, and proposed standards, many available on the Web. However, there is plenty of opportunity for further research, collaboration, and improvement. As mentioned in the challenges of autonomic computing, the same challenge surfaced during research for this paper: visibility into Web service APIs for errors and faults. When using the *telnet* program and the *echo* service to invoke Web services calls, error messages were not always helpful and descriptive. When encountering errors, error codes typically had to be cross-referenced with available documentation and forums, which was manually involved and a little time consuming. Although some of these errors were transient, and could be fixed permanently with testing (e.g., using an incorrect key or HTTP method), some problems would likely reoccur involving expired tokens or invalid HTTP headers, etc. As some evidence of reoccurring failures, a search for "SOAP and Flickr" via the Google.com search engine produces a SOAP error message from the *echo* service as its 7th highest result (see Fig. 8) [45].

```
<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
<s:Body>
<s:Fault>
<faultcode>flickr.error.0</faultcode>
   <faultstring>Invalid SOAP envelope.</faultstring>
<faultactor>http://www.flickr.com/services/soap/</faultactor>
<details>Please see http://www.flickr.com/services/api/ for more
details</details></s:Fault>
</s:Body>
</s:Envelope>
```

Fig. 8. SOAP error message from invalid SOAP method.

This one example, albeit small, seems to be a good example of how a few changes could potentially produce an autonomic type effect. Before delving into ideas, there are—of course—changes that could be implemented manually within current context. For example, in this simple *echo* service case, the web crawler could obviously be updated manually; however, recognizing proprietary error codes in a plethora of SOAP calls is probably not practical. Another manual option could encompass adding a robots.txt file to the echo services (see http://www.robotstxt.org/wc/robots.html) in order to dissuade crawlers from invoking the call or some other standard crawling method. However, these are both manually involved, when an automated one is preferred.

VII. Research: Observations

As with the *echo* example, problems surfaced concerning the context and interoperability of the fault codes. One of the goals for the exercise was to contemplate autonomic computing mechanisms that could be used to automatically handle Web service errors, along the lines of IBM's autonomic architecture, log adapters, and CBE format, etc. While using the Flickr *echo* SOAP service, the same error was received as the one supposedly received from the Google search crawler [45]. The cause was eventually discovered as an errant HTTP method, but the problem was not apparent from the fault code or from the message within the SOAP envelope. Although a simple example, and not entirely indicative of complex, real-world scenarios, it is likely that any log analyzer and monitor would require better error information from the response before remediation.

Moreover, as each Web service provider was researched, it was interesting to discover how each API supports a different set of fault codes and descriptions for errors (i.e., outside of the HTTP and SOAP defaults [46]). Although application specific errors may be necessary, it seems advantageous for Web service groups to define a common set of error codes and messages across providers, much like errno and errno.h provide context on Unix and Unix-variant operating systems. Common definitions for errors and descriptions could aid developers and assist groups in autonomic computing efforts to handle certain error cases. In order to achieve an interoperable effect, an XML namespace could be introduced to cover a standard set of fault codes, fault types, and descriptions (see Fig. 9). Once supported, certain conditions could be detected and handle automatically.

```
<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope xmlns:s=http://www.w3.org/2003/05/soap-envelope
            xmlns:err=http://host/2007/05/standard-errors
>
<s:Body>
<s:Fault>
<faultcode>err: Client.http.method.get</faultcode>
   <faultstring> Invalid Method </faultstring>
<faultactor>My specific detail </details></s:Fault>
</s:Body>
</s:Envelope>
```

Fig. 9. SOAP pseudo-code error message.

Another interesting facet surfaced when examining SOAP errors within the *echo* service: out-of-sync return codes and duplicity of errors. Although HTTP defines error codes for methods not allowed and other server error messages, the service would respond with an *Invalid SOAP envelope,* due to a GET versus a POST HTTP method. Again, although a simple example, and easily rectified, there is an additional layer of abstraction and complexity added with SOAP tunneled over HTTP. Given my experience with HTTP, errors may be ambiguous, and any additional ambiguity with tunneled protocols would seem to further complicate efforts of interoperability and of autonomic initiatives. For this reason it may be better to focus some autonomic research on REST-based styles first before delving into SOAP-based

mechanisms. At the least, it seems worthwhile review implementations of HTTP and SOAP return codes and messages before applying autonomic algorithms. Along these lines, the author suspects that if the error code inside of the SOAP envelope for the Flickr *echo* service were in synch with the HTTP error response code (i.e., instead of using 200-success for HTTP and a faultcode with SOAP), then the response would not be indexed or cached by the Google search engine, thus enabling an *autonomic*-type capability.

However, regardless of these observations, out of the three architectural styles researched, there were two that surfaced as the most likely candidates for additional fault tolerant autonomic research: REST and SOAP. All the major Web service providers studied provide REST, or REST like interfaces, in their APIs. Furthermore, the second most supported style was SOAP. Many Web service infrastructure vendors provide support for SOAP; many Web services API providers are supporting SOAP as well. Clearly, both styles are important when concerning autonomic models and technologies in Web services.

VIII. Conclusion

The paper reviewed autonomic computing initiatives, technologies, and trends. Moreover, it studied Web service architectural styles in order to gauge impacts on autonomic computing research efforts. Overall, the industry is making good progress on autonomic computing technologies, like with IBM's Autonomic Computing Toolkit, and with open standard efforts, like the Service Modeling Language, which involved multiple industry vendors. Furthermore, those trends will have to continue to incorporate REST and SOAP-based architectural styles. Perhaps SOAP will be the preferred method for Web services for all APIs in the future, but REST styles currently claim ubiquity across all publicly available Web service providers researched. When researching publicly available Web service APIs and architectural styles, the author discovered an apparent need for better standard error reporting within and across Web API providers and for better synchronized error reporting between protocol layers, which could aid in interoperability and in autonomic approaches. Perhaps, in the future more research can be spent in this area. Hopefully, the research contributed to autonomic computing initiatives and aids those researching the topic.

IX. References

[1] T. B. Lee (1996, August). *The World Wide Web: Past, Present, and Future*, par. 1 [Online]. Available: http://www.w3.org/People/Berners-Lee/1996/ppf.html

[2] W3C Working Group Note 11 (2004, February). Web Services Architecture, *section 1.4*. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis.

[3] T. O'Reilly (2005, September). *What Is Web 2.0? Web As Platform* [Online]. Available: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[4] M. Salenhie, L. Tahvildari. *Autonomic Computing: Emerging Trends and Open Problems,* DEAS 2005, St. Louis, MO. par. 2.

[5] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, et al. (2002, March 15). Recovery Oriented Computing (ROC): *Motivation, Definition, Techniques, and Case Studies*. Computer Science Technical Report UCB/CSD-02-1175, U.C. Berkley, pp. 1-2. [Online]. Available: http://roc.cs.berkeley.edu/papers/ROC_TR02-1175.pdf

[6] P. Horn (2003, January). *Autonomic Computing: IBM's Perspective on the State of Information Technology* [Online], p. 1, p. 17. Available: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[7] J. O. Kephart, D. M. Chess (2001, October) [Online]. *The Vision of Autonomic Computing*, pp. 2-3, p. 5, p. 45, pp. 48-50. Available: http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf

[8] E. Mancel, M. J. Nielsen, A. Salahshour, S. Sampath K. V. L, S. Sudarshanan (2005). Problem Determination Using Self-Managing Autonomic Technology, p.5, pp. 195-198, pp. 306-309, p. 14. IBM Redbooks [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg246665.pdf

[9] E. Roch. (May 10, 2006) *SOA versus REST Debate.* [Online]. Available: http://blogs.ittoolbox.com/eai/business/archives/soa-versus-rest-debate-9225

[10] J. O. Kephart, (May 2005) *Research Challenges of Autonomic Computing*, pp. 15-16, p. 20-21. [ACM]. *ICSE'05,* St. Louis, Missouri, USA.

[11] M. Shaw. *Self-healing: Softening Precision to Avoid Brittleness*. Position paper for SIGSOFT WOSS '02: Workshop on Self-Healing Systems, pp. 111-113.

[12] M. Parashar, S. Hariri Autonomic Computing: An Overview. p. 248. The Applied Software Systems Laboratory, Rutgers University, High Performance Distributed Computing Laboratory, University of Arizona. Available: http://www.caip.rutgers.edu/TASSL/Papers/automate-upp-overview-05.pdf

[13] L. D. Paulson (2002, August 2002). Computer system, heal thyself, pp. 20, par. 4. Computer (Volume 35, Issue 8) [IEEE Xplore].

[14] F. Hayes (2003, November 17). *The Story So Far: The History of RAID: Redundant Arrays of Inexpensive Disks turned out to be expensive—but dependable*. Computer World [Online]. Available: http://www.computerworld.com/hardwaretopics/storage/story/0,10801,87093,00.html

[15] D. Menasce (2005). Virtualization: Concepts, Applications, and Performance Modeling. The Volgenau School of Information Technology and Engineering [Department of Computer Science, George Mason University], pp. 10-11. Available: http://cs.gmu.edu/~menasce/papers/menasce-cmg05-virt-slides.pdf

[16] A. Lapouchnian, Y. Yu, S. Liaskos, J. Mylopoulos. *Requirements*-Driven Design of Autonomic Application Software. Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research (CASCON '06). Department of Computer Science, University of Toronto, p.1 2006.

[17] D. Garlan. (2003). Self-healing systems. CMU CS 17-811. Available: http://www.cs.cmu.edu/~garlan/17811/Lectures/01-Course-Intro.pdf

[18] Y. K. Malaiya (2007). Fault Tolerant Computing (Colorado State University, CS 530 DL). Introduction: Lecture 1 Notes. Slide 9.

[19] IBM Corporation (2003). *An architectural blueprint for autonomic computing*, pp. 6, p. 10, pp. 10-11 [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/ACwpFinal.pdf

[20] IBM (2005, September). Autonomic Computing Toolkit: User's Guide. Available (3rd ed.), p. 2, p. 7, pp. 11-19. [Online]. Available: ftp://www6.software.ibm.com/software/developer/library/autonomic/books/fpu3mst.pdf

[21] C. Preimesberger (2004, February 16). IBM releases first 'autonomic' SDK. NewsForge, Application Development Available: http://www.newsforge.com/article.pl?sid=04/02/15/2354219

[22] The Globus Alliance (2007, May 6). Open Grid Services Architecture. Available: http://www.globus.org/ogsa/

[23] B. Miller (2005, September 13). The autonomic computing edge: The role of knowledge in autonomic system. Available: http://www-128.ibm.com/developerworks/autonomic/library/ac-edge6/

[24] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee (June 1999). *Introduction. RFC 2616: Hypertext Transfer Protocol*. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.txt

[25] R. Fielding (2000). *Architectural Styles and the Design of Network-based Software Architectures*, *1.5 Styles*, *Chapter 5: REST*. Ph.D. dissertation, Dept. Information and Computer Science, UC, Irvine, 2000. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/software_arch.htm#sec_1_5

[26] H. Hao. *What Is Service-Oriented Architecture?* [Online]. Available: http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[27] D. Winer (1999, September 25). Dave's History of SOAP, [Online]. Available: http://www.xmlrpc.com/stories/storyReader$555

[28] G. Murray (2006, October 2nd ed.). Asynchronous JavaScript and XML [Online]. Available: http://java.sun.com/developer/technicalArticles/J2EE/AJAX/

[29] Flickr Services (2007). Available: http://www.flickr.com/services/api/

[30] D. Winer (1999, July 15). XML-RPC Specification. Available: http://www.xmlrpc.com/spec

[31] W3C XML Protocol Working Group. (2003, June 24). *SOAP Version 1.2 Part 0: Primer*, [Online]. Available: http://www.w3.org/TR/soap12-part0/#L1149

[32] F. Sommers (2003, March 17) *Why Use SOAP? Choosing Between SOAP and Application-Specific XML for Your Web Services*. Available: http://www.artima.com/webservices/articles/whysoap.html

[33] I. De Jong (2002, April 27). Web Services/SOAP and CORBA. Available: http://www.xs4all.nl/~irmen/comp/CORBA_vs_SOAP.html#2

[34] Amazon, Inc. Web Services (AWS) (2007, April 4). E-Commerce Service Developer Guide [Online]. Available: http://docs.amazonwebservices.com/AWSECommerceService/2007-04-04/DG/

[35] Flickr Web Services API (unit of Yahoo! Incorporated) (2007, May 7). Request Formats [Online]. Available: http://www.flickr.com/services/api/

[36] Ebay, Inc. Developers Program (2007, May 7). Available: http://developer.ebay.com/support/docs/

[37] Google Inc. Developer Network [Online]. Available: http://code.google.com/

[38] YouTube. API Documentation [Online]. Available: http://youtube.com/dev_docs

[39] T. O'Reilly (2003, April 3). REST vs. SOAP at Amazon [Online]. Avaialble: http://www.oreillynet.com/pub/wlg/3005

[40] Microsoft Corporation (2007). *Dynamic Systems 2007: Get Started With Dynamic systems Technology Today*, pp. 3-6 [Online]. Available: http://download.microsoft.com/download/C/3/C/C3CE985F-7C01-4DB3-81EA-EE4A00E06B49/DSI_Overview.doc

[41] M. J. Turner (2006, September). HP Inc. and Summit Strategies, Inc. *HP's Adaptive Infrastructure Initiative Powers Enterprise Business Priorities*, pp. 8-13. http://h71028.www7.hp.com/ERC/downloads/Summit_HP-Adaptive-Infrastructure_Custom_09-21-2006.pdf

[42] D. Orchard, K. Wilson, K. Sankar, W. Adams, J. Bell, S. Holbrook, et al., (2007, February 28). Service Modeling Language (SML): request to W3C [Online]. Available: http://www.w3.org/Submission/2007/01/

[43] V. Tewari, M. Milenkovic (2006). *Standard for Autonomic Computing*. Intel Technology Journal, Volume 10(4), p. 1. Available: ftp://download.intel.com/technology/itj/2006/v10i4/v10-i4-art03.pdf

[44] IBM (2007, May 7). ELiza. Available: http://www-03.ibm.com/servers/autonomic/

[45] "Flickr and SOAP" (2007, May 7). Google search query [Online]. Available: http://www.google.com/search?client=safari&rls=en&q=Flickr+and+SOAP&ie=UTF-8&oe=UTF-8

[46] R. Monson-Haefel (2004, February 6). *SOAP Faults*. Using SOAP with J2EE [Online]. Available: http://www.awprofessional.com/articles/article.asp?p=169106&seqNum=6&rl=1