SECURED REMOTE TRACKING OF CRITICAL AUTONOMIC COMPUTING APPLICATIONS

Paritosh Kumar Srivastava, Sandeep Sahu
Dept. of IT, ABV-Indian Institute of Information Technology & Mgmt. Gwalior, 474003, India
paritosh_iiitm@yahoo.co.in, sahusandeep@yahoo.com

Abstract

Autonomic computing is an entirely new philosophy for the development of computing systems .It is emerging as a significant new approach to the design of computing systems. Its goal is the development of systems that are self-configuring, self-healing, self-protecting and self-optimizing. This makes autonomic computing best candidate for designing critical applications, thus minimizing human intervention. But these applications need to be monitored, even if total maintenance is done by the system itself. Remotely tracking the performance of system is necessary in case of critical autonomic computing applications to spot any extraordinary behavior which may arise due to some error .But the tracking system needs to be a part of the whole system and needs to be secure as it is vulnerable to all kinds of security threats. This paper proposes to add security at the application layer to track remotely these critical applications .These security mechanisms secures the communication channel between tracking system and application thus making it safe from all kinds of security threats. An example of a critical autonomic computing application, e-medicine is also discussed showing how the remote tracking system can help in monitoring the application guaranteeing total security.

Keywords: *Autonomic Computing, Security, Remote tracking, Web Services*

1. INTRODUCTION

The difficulty of managing today's computing systems is not only because of the administration of individual software environments, but also because of the need to integrate multiple heterogeneous environments, and to extend beyond company boundaries into the Internet [1]. All these factors contribute to increased levels of complexity in computing systems. Installing, configuring, and maintaining such large systems is becoming an increased challenge even for experts. A possible solution to this problem is to embed the complexity in the system infrastructure itself (both hardware and software), then automating its management. This is in a way similar to the human system, with its autonomic nervous system, which provides automatic, involuntary regulation of the major physiological functions.

The essence of autonomic computing systems is self-management, the intent of which is to free system administrators from the details of system operation and maintenance. The power of autonomic computing makes them best candidate for designing critical applications like the ones used in space research programmes, bio-technological equipments, etc. But it is very important to ensure high reliability in these systems because any minute error can create a big havoc.

So it is proposed in this paper to monitor these critical systems using a remote tracking system. But though autonomic computing systems have security built-in, connecting them to the remote tracking system through an ordinary communication channel opens the system to various security threats. To guard against such threats, it is proposed in this paper to implement security at the application layer for the autonomic computing application and its tracking system. An example of an autonomic computing application, E-Medicine is also discussed and a remote tracking system is also used which is secured making use of WS-Sec standards.

2. AUTONOMIC COMPUTING

The need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond company boundaries into the Internet, introduces new levels of complexity. Computing systems' complexity appears to be approaching the limits of human capability, yet the march toward increased interconnectivity and integration rushes ahead unabated. As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime. Soon systems will become too massive and complex for even the most skilled system integrators to install, configure, optimize, maintain, and merge. And there will be no way to make timely, decisive responses to the rapid stream of changing and conflicting demands [2].

The difficulty of managing today's computing systems goes well beyond the administration of individual software environments. The need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond company boundaries into the Internet, introduces new levels of complexity. Computing systems' complexity appears to be approaching the limits of human capability, yet the march toward increased interconnectivity and integration rushes ahead unabated.

As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime. Soon systems will become too massive and complex for even the most skilled system integrators to install, con-figure, optimize, maintain, and merge. And there will be no way to make timely, decisive responses to the rapid stream of changing and conflicting demands.

Autonomic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost.

2.1 Self-management

The essence of autonomic computing systems is self-management, the intent of which is to free system administrators from the details of system operation and maintenance and to provide users with a machine that runs at peak performance 24/7 in face of changing components, workloads, demands, and external conditions and in the face of hardware or software failures, both innocent and malicious. The autonomic system might continually monitor its own use, and check for component upgrades, for example. If it deems the advertised features of the upgrades worthwhile, the system will install them, reconfigure itself as necessary, and run a regression test to make sure all is well. When it detects errors, the system will revert to the older version while its automatic problem-determination algorithms try to isolate the source of the error.

2.2 Self-configuration

Installing, configuring, and integrating large, complex systems is challenging, time-consuming, and error-prone even for experts. Most large Web sites and corporate data centers are haphazard accretions of servers, routers, databases, and other technologies on different platforms from different vendors. It can take teams of expert programmers' months to merge two systems or to install a major e-commerce application such as SAP.

Autonomic systems will configure themselves automatically in accordance with high-level policies—representing business-level objectives, for example—that specify what is desired, not how it is to be accomplished. When a component is introduced, it will incorporate itself seamlessly, and the rest of the system will adapt to its presence

2.3 Self-optimization

Complex middleware, such as WebSphere, or database systems, such as Oracle or DB2, may have hundreds of tunable parameters that must be set correctly for the system to perform optimally, yet few people know how to tune them. Such systems are often integrated with other, equally complex systems. Consequently, performance-tuning one large subsystem can have unanticipated effects on the entire system.

2.4 Self-healing

IT vendors have large departments devoted to identifying, tracing, and determining the root cause of failures in complex computing systems. Serious customer problems can take teams of programmers several weeks to diagnose and fix, and sometimes the problem disappears mysteriously without any satisfactory diagnosis. Autonomic computing systems will detect, diagnose, and repair localized problems resulting from bugs or failures in software and hardware, perhaps through a regression tester,

2.5 Self-protection

Despite the existence of firewalls and intrusion detection tools, humans must at present decide how to protect systems from malicious attacks and inadvertent cascading failures. Autonomic systems will be self-protecting in two senses. They will defend the system as a whole against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures. They also will anticipate problems based on early reports from sensors and take steps to avoid or mitigate them.

Once a computing system has the inherent properties of self-mechanism and complexity hiding, it will be able to exhibit a wide range of capabilities and attributes, such as autonomy in the control and management of internal resources and external service provisions, trustworthiness, robustness to un-modeled properties of the external environment, adaptability to external environment, fault tolerance/resumption, resilience to environment, protectiveness against attacks, etc. Major challenges of autonomic computing will be the design of the self-mechanism and the complexity hiding in addition to the conventional design of the system, more exactly, how to explicitly design the self-mechanism and the complexity hiding. Some other issues will be that autonomic systems are inevitably knowledge intensive systems, and the architecture of autonomic computing is absolutely paramount.

3. CHALLENGES BEFORE AUTONOMIC COMPUTING APPLICATIONS

Critical autonomic computing applications are those which perform very important tasks such as real-time autonomic systems or others where fault tolerance limit is almost equal to zero. So, such applications need to be remotely monitored to keep track of important system performance evaluation parameters. Even though autonomic computing applications are self-managed, for critical systems a separate monitoring system is imperative to check and keep track of any extraordinary behavior shown by the system. So, a remote tracking system should be used to monitor the autonomic computing application.

But using such a system opens the autonomic computing system to new threats as autonomic computing applications work on the platform of web services, and the remote tracking system will also be communicating with the application using these services only. But there is no application level security in web services implemented which can check the security threats to which this communication channel is vulnerable. Hence, if any intruder breaks into the communication channel and plays with the system performance's information passed from the application to the tracking system, there are chances the critical application may start behaving erroneously. In such a case, even if the autonomic computing application is behaving erroneously, tracking system is not able to detect it because the intruder hides the correct information from him.

Thus to remotely monitor the critical autonomic computing application, and to make sure that communication channel is totally secure against all kinds of attacks, it is proposed to add security at the application layer. Using WS-Sec, this task can be done through XML encryption and XML signature. Thus using WS-Sec, authentication, authorization can be done as well as the confidentiality of messages can be ensured.

2. WEB SERVICES SECURITY

This emerging model lets Web applications call independently published Web based software components to conduct business transactions. A typical Web services architecture consists of three entities:

1. Service providers who create Web services and publish them to the outside world by registering the services with service brokers.
2. Service brokers who maintain a registry of published services; and service requesters who find required services by searching the service broker's registry.
3. Service requesters then bind their applications to the service provider to use particular services.

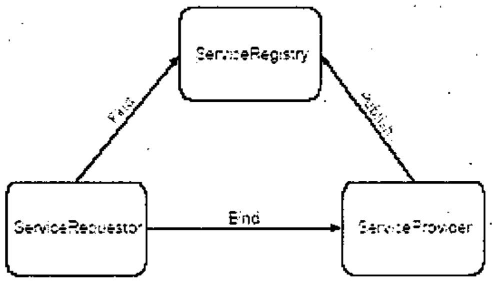


Fig. 1 Architecture of web services

Figure 1 shows the interaction between service providers, service brokers, and service requesters in the publication, discovery, and consumption of Web services.

Web services are essentially founded upon three major technologies: Web Services Description Language (WSDL); Universal Description, Discovery and Integration (UDDI); and the Simple Object Access Protocol (SOAP) [3]. These three technologies form the core Web services technologies.

WSDL is an XML language for describing the programmatic interfaces Web services .Conceptually it is similar to the interface definition language (IDL) used by the Common Object Request Broker Architecture (CORBA). The description includes details like data type definitions, the operations supported by the service, input/output message formats, network address, protocol bindings, and so on.

UDDI lets Web services register their characteristics with a registry so that other applications can look them up. The UDDI specification provides a mechanism to register and locate Web services. It defines an electronic business registry where businesses can describe their business and register their Web services as well as discover and integrate with other businesses that offer Web services. UDDI is itself a Web service that is based on XML and SOAP. Interaction with UDDI is accomplished via a set of pre-defined SOAP interfaces.

SOAP provides a simple and lightweight protocol for exchanging XML (Extensible Markup Language) data over the Web. Client applications typically call Web services over Service providers the Web using SOAP mechanisms.

Despite the promise, Web services present network administrators with a thorny problem: As network security becomes an increasing concern, Web services open up networks by letting outside users access databases, applications, and internal users.

Traditional security techniques—such as virtual private networks or secure sockets layer (SSL) technology— cannot secure the large number of transactions that Web services can perform in a short time.

3.1 Security issues

Web services security requires authentication (establishing identity), authorization (establishing what a user is allowed to do), confidentiality (ensuring that only the intended recipient can read the message, accomplished with encryption), and integrity (ensuring the message hasn't been tampered with, generally accomplished with digital signatures).

Browsers support SSL and TLS, but the protocols don't scale well to complex, high-volume transactions, like those in Web services. This is because SSL and TLS systems must decrypt data every time it arrives at a new Web server and then encrypt the data for transmission to the next server.

SAML defines a vendor-neutral way to express security information in an XML format. It defines the schemas for the structure of documents that include information related to user identity and access or authorization rights. By defining how this information is exchanged, SAML lets companies with different internal security architectures communicate. Assertions provide proof of identity— via SAML subjects, which contain identity-related information—for users and computers. In addition, assertions list transaction-related user information (such as credit limits for e-commerce) and activities users are authorized to perform (such as executing permissions to access and work with files). SAML can also indicate the authentication method that must be used with a message, such as a password, Kerberos authentication ticket, hardware token, or X.509 digital certificate.

SAML would facilitate single-sign on for Web users by, for example, letting them log on to one site and have their security credentials transferred automatically to partner sites for authentication.

3.2 WS-Sec

WS-Sec, under consideration as a standard by Oasis, lets applications attach security data to the headers of SOAP messages. This can include security metadata like that found in the XML Encryption and XML Signature specifications. WS-Sec lets companies send messages with digital signatures that tell recipients whether hackers have altered documents during transmission and whether the documents are actually from the person named as the sender.

The XML Encryption and XML Signature specifications are central to WS-Sec. The two approaches provide ways to include both encrypted data and digital signatures in XML documents. They also include XML elements that identify the encryption.

1. **XML Encryption:** XML Encryption describes the process for encrypting and representing encrypted data in XML documents. The specification supports common encryption algorithms and techniques. The standard provides ways to encrypt all or just parts of the XML in the message. Proponents say this approach is more efficient because information that isn't confidential can be sent unencrypted. Selective encryption and signing also let senders add different signatures and keys to parts of a single document that are designated for different recipients.
2. **XML Signature:** XML Signature defines syntax and processing rules for representing digital signatures. Digital signatures on one computer can be read by another because the machines work with the same encrypted digest—a cryptographic hash that represents the signed material—for the same section of XML code

4. SECURED REMOTE TRACKING

An autonomic system is an autonomous computing environment that completely hides its complexity. Complexity hiding from users/services means that autonomic computing will provide users with a computing environment that allows them to concentrate on what they want to do without worrying about how it has to be done. This particular property is highly advantageous for designing critical applications, minimizing human intervention and adding more accuracy to the task accomplished by the application.

To discuss the benefits of autonomic computing, an example of a critical autonomic computing application, e-medicine is taken and discussed how it is based on the autonomic computing architecture. Then to monitor such a system a remote tracking system is proposed. Also this remote tracking system is made secure to safeguard against all security threats.

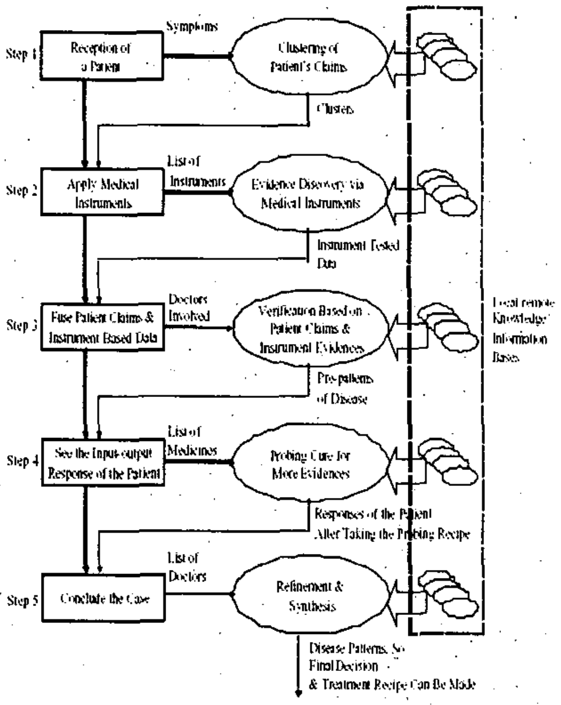
4.1 E-medicine

Extensive research has been carried out in health informatics, e.g., the object management group's specifications for healthcare domain [4]. Computer supported medical diagnosis is an important area of health informatics. Cooperative medical diagnosis is a complicated process of evolving, decision-making, reasoning, therapy planning, and/or guessing. It crucially relies upon actors' knowledge and the utilization of their knowledge. Physicians/surgeons/clinicians from a variety of medical specialties co-operate toward reasonable and feasible diagnosis. Such co-operation is through the interoperability facilities (platforms, architectures, communication languages, system management mechanisms (e.g., forgetting, version, backtracking, etc.)), and is represented in the dynamic growth of the procedural knowledge and descriptive knowledge. Different types of actors are involved in cooperative medical diagnosis process, including human physicians/surgeons/clinicians, information repositories, intelligent instruments, and software packages. Actors are packaged into agents, i.e. agentized, so that services are made accessible and sharable in a unified and integrated computing model. The agents represent all types of scientific, instrumental and historical evidences, human doctors and the patients, which mostly are heterogeneous in formats and contents, public and proprietary in accessibility, and temporally distributed over a long history and geographically dispersed all over the globe. Medical diagnosis is a decision making process of repeat, detour, verification, refinement, self-exploring, converging, synthesis, etc. At a stage of this process, not all agents are actually drawn into involvement. Actors play different major and minor roles at different stages. Procedure of cooperative medical diagnosis can typically be divided into five decision steps, i.e., clustering of patient's claims, evidences discovery by use of medical instruments, verification based on patient's claims and instrument-inspected evidences, probing cures for more evidences, refinement and synthesis, as depicted in Figure 2.

4.2. Multi-agent autonomic architecture for e-medicine

Interoperability between distributed and heterogeneous medical/health knowledge/information bases is essential for cooperative medical diagnosis. The major challenges of cooperative medical diagnosis are the legacy and diversity, and the distributing and heterogeneity of various kinds of medical/health knowledge/information bases. The multi-agent autonomic architecture (MA3) [5] is applied to evidence-based collaborative medical diagnosis. At the higher level of MA3 lies the medical diagnosis decision support, and at its lower level are various kinds of knowledge/information bases. Multi-agent systems will perform interaction, interoperation, evolution, interfacing browsing, searching, archival, filtering, knowledge mining, automatic document analysis, virtual localization of information sources, etc.

This is made possible by utilizing and applying autonomic computing architecture to the E-Medicine application. As a result, following an autonomic computing architecture, it will be able to realize the collaborative diagnosis transcending geographical/organizational barriers and derive optimal and alternative diagnostic conclusions and treatment/healthcare plans [6].



Step 1
Reception of a Patient
Symptoms
Clustering of Patient's Claims
Clusters
Step 2
Apply Medical Instruments
List of Instruments
Evidence Discovery via Medical Instruments
Instrument Tested Data
Step 3
Fuse Patient Claims & Instrument Based Data
Doctors Involved
Verification Based on Patient Claims & Instrument Evidences
Pre-patterns of Disease
Local remote Knowledge Information Bases
Step 4
See the Input-output Response of the Patient
List of Medicines
Probing Cure for More Evidences
Responses of the Patient After Taking the Probing Recipe
Step 5
Conclude the Case
List of Doctors
Refinement & Synthesis
Disease Patterns, So Final Decision & Treatment Recipe Can Be Made

Figure 2. E-Medicine

Now, this application, e-medicine is a highly critical one. The reason for being it so is that the application is a real-time one and all the different agents involved in the architecture depend on each other. So, there is an acute and pressing need to monitor and track this system to protect the system against any extraordinary or unforeseen error.

Hence in Fig. 3, a remote tracking system is shown which monitors the system and alarms in case of any problem. Also using WS-Sec, with the application of XML signature and XML encryption, this communication channel is made secure. The process of initiating and terminating communication between application and tracking system is as follows:

1. The tracking system initiates the communication channel by requesting for connection.
2. Using digital signature, the tracking system and application authenticates each other.
3. Now, the communication is started. This communication is nothing but transfer of the application's performance variables which are monitored by the remote system. The data transferred is also made secured by XML encryption.
4. After monitoring is over, tracking system terminates the connection.

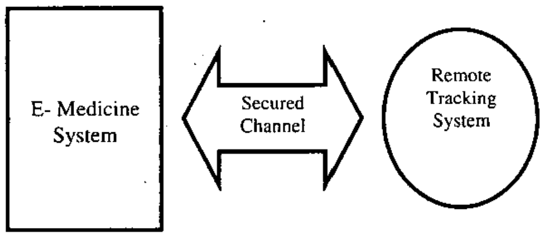


Fig. 3 Secured Remote Tracking system

7. CONCLUSIONS

Autonomic computing is an emerging holistic approach to computer system development that aims to bring a new level of automation to systems through self-healing, self-optimizing, self-configuring and self-protection functions. Thus once a computing system has these properties, it will be able to exhibit a wide range of capabilities and attributes, such as autonomy in the control and management of internal resources and external service provisions, trustworthiness, robustness to un-modeled properties of the external environment, etc.

Use of autonomic computing in designing and maintenance of critical systems is going to be the next big wave in information technology industry. But remote tracking of these systems is also required to meet the reliability requirements. Securing the communication channel between the tracking system and application system guarantees protection from all kinds of security threats.

Acknowledgements

The authors would like to thank Dr. Rajendra Sahu, Asstt. Professor, IT Department, ABV-IIITM Gwalior, India for his encouragement and guidance in carrying out the work.

References

[1] Zoran Constantinescu, "Towards an Autonomic Distributed Computing System," *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA'03)*, 2003

[2] Jeffrey O. Kephart and David M. Chess, "The Vision of Autonomic computing," *Computer*, IEEE Publishing Society, January 2003.

[3] David Geer, "Taking Steps to Secure Web Services," *Computer*, IEEE Publishing Society, January 2003.

[4] OMG (object management group), http://www.omg.org

[5] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era.," *IBM Systems Journal*, Vol. 42, No.1, pp. 5-18, 2003

[6] Huaglory Tianfield, "Multi-Agent Autonomic Architecture and Its Application in E- Medicine," *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology(IAT'03)*, 2003.