

American University in Cairo

AUC Knowledge Fountain

Archived Theses and Dissertations

6-1-2006

Web services - based autonomic computing framework (WSAC)

Sherif Atef Gurguis

The American University in Cairo AUC

Follow this and additional works at: https://fount.aucegypt.edu/retro_etds



Part of the [Digital Communications and Networking Commons](#)

Recommended Citation

APA Citation

Gurguis, S. A. (2006). *Web services - based autonomic computing framework (WSAC)* [Thesis, the American University in Cairo]. AUC Knowledge Fountain.

https://fount.aucegypt.edu/retro_etds/2042

MLA Citation

Gurguis, Sherif Atef. *Web services - based autonomic computing framework (WSAC)*. 2006. American University in Cairo, Thesis. *AUC Knowledge Fountain*.

https://fount.aucegypt.edu/retro_etds/2042

This Thesis is brought to you for free and open access by AUC Knowledge Fountain. It has been accepted for inclusion in Archived Theses and Dissertations by an authorized administrator of AUC Knowledge Fountain. For more information, please contact fountadmin@aucegypt.edu.

**WEB SERVICES .
BASED AUTONOMIC
COMPUTING
FRAMEWORK**

**SHERIF ATSE
GURGIS**

2006



The American University in Cairo
School of Sciences and Engineering



**WEB SERVICES - BASED AUTONOMIC COMPUTING
FRAMEWORK
(WSAC)**



A Thesis Submitted to

Department of Computer Science
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
The degree of Master of Science

By
Sherif Atef Gurguis

B.Sc. in Electrical Engineering, Electronics and Telecommunication
Zagazig University, August 1998

Under the supervision of
Dr. Amir Zeid
Dr. Sherif El Kassas

May 2006

The American University in Cairo

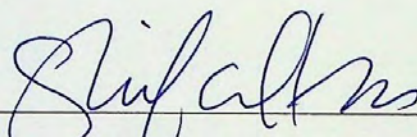
**WEB SERVICES - BASED AUTONOMIC COMPUTING
FRAMEWORK
(WSAC)**

A Thesis Submitted by Sherif Atef Gurguis
To The Department of Computer Science
May 2006

In partial fulfillment of the requirements for
The Degree of Master of Science
has been approved by

Dr. Sherif El-Kassas

Thesis Committee Chair / Adviser

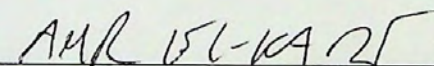


Affiliation

Department of Computer Science, The American University in Cairo

Dr. Amr El-Kadi

Thesis Committee Reader / Examiner

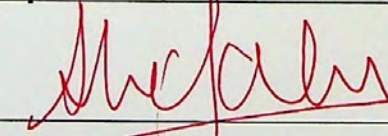


Affiliation

Department of Computer Science, The American University in Cairo

Dr. Sherif Gamaleldin Aly

Thesis Committee Reader / Examiner

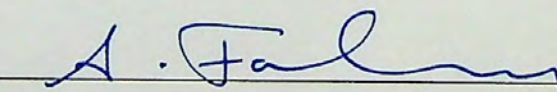


Affiliation

Department of Computer Science, The American University in Cairo

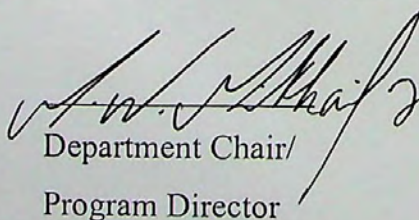
Dr. Aly Aly Fahmy

Thesis Committee Reader / Examiner

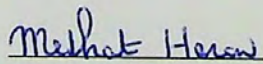


Affiliation

Dean of Faculty of Computers and Information, Cairo University

 June 6, 06
Date

Department Chair/
Program Director


Date

Dean

6 June 2006
Date

DEDICATION

To the soul of my mother, my lovely wife, and all my family and friends

ACKNOWLEDGEMENTS

The first thank you must go to my research supervisors; I would like to thank Dr. Amir Zeid who started this research with me and inspired me with the Autonomic Computing topic to be the original topic of this research and Dr. Sherif El Kassas who accepted to supervise the remaining work in this research despite of his tight schedule, and without his support and active participation in every step of the process, this thesis may never have been completed. I would like to acknowledge all the readers of this thesis who have given some of their precious time in reading and assessing the thesis. I would also like to express my sincere gratitude to my friend Rafik Amir and the IBM alphaworks technical support staff, especially Daniel Jemiolo, who helped me to solve some of the problems I faced during the implementation of the prototype. Lastly, a word of thanks and appreciation must go to my lovely wife for revising the entire document and for her continuous support and to both of my father and my father-in-law for their permanent encouragement.

ABSTRACT

The complexity associated with the advancement in computing systems, has raised a critical challenge, which is that the traditional approaches that have been used for building and managing hardware and software computing systems are no longer sufficient to achieve the much needed advances in our ability to manage and operate such systems. Therefore, without creating alternative approaches, the expected benefits of new technologies would not be achieved.

In the context of architecting computing systems, Service-Oriented Architecture (SOA) was introduced in order to fulfill new business requirements that entail dynamic and seamless relationships, which could not be accommodated by traditional development paradigms, such as the Object-Oriented one. By applying the concepts introduced by SOA into industrial computing systems, the notion of Web Services (WS) has emerged. Although, the expected business benefits from WS are enormous, managing the complexity that is introduced in different WS scenarios was recognized as a crucial challenge that threatens accomplishing those expected benefits. By working around that challenge, it was concluded that managing WS-based applications cannot be based on the human-intervention management model; and therefore, there has been a need for working out an approach that can be applied in the context of automating WS.

Additionally, by contrasting the problem of managing complicated computing systems with the human biology domain, especially the human autonomic nervous system, the approach of Autonomic Computing (AC) was introduced by IBM as the approach of permitting computing systems to be self-manageable. Therefore, the notion of AC can be used in the context of managing modern IT infrastructures.

However, the fact that such infrastructures are usually composed of a set of heterogeneous computing resources that implement different protocols and standards and also run over different underlying platforms makes achieving the goal of adopting the notion of AC a hard task. Clearly, building AC systems cannot be based on preparatory standards, but rather, they should be built with the aid of open standards that permit achieving the goal of self-management regardless of the implementation details of the computing resources being managed.

By considering the two highlighted challenges of both WS and AC in the previous two paragraphs, the problem that is considered in this thesis can be defined as a two-directional problem with one direction focusing on working around the challenge of automating the management of WS and the other direction focusing on dealing with the challenge of heterogeneity in the context of AC. The promises of both AC and WS paradigms motivated us in this research to employ of the notion of AC to automate the management of WS-based applications and to exploit the technologies of WS to overcome the obstacle of heterogeneity that hinders achieving the expected benefits of AC. In order to achieve the goal of our research, which is solving the identified two-directional problem, we propose in this thesis the Web Services - Based Autonomic Computing Framework (WSAC), which is composed of an architectural model as well as a processing model. To conclude the research efforts, a proof-of-concept prototype, which focuses on the self-healing aspect of AC, was built in order to prove the effectiveness of the proposed framework in achieving the goal of our research.

TABLE OF CONTENTS

| | |
|--|-----|
| LIST OF TABLES | VII |
| LIST OF FIGURES | VII |
| LISTINGS..... | VII |
| CHAPTER 1: INTRODUCTION..... | 1 |
| 1.1 CHAPTER OVERVIEW | 1 |
| 1.2 RESEARCH TOPIC | 1 |
| 1.3 RESEARCH MOTIVATION..... | 3 |
| 1.4 PROBLEM DEFINITION..... | 4 |
| 1.5 THESIS OBJECTIVE | 5 |
| 1.6 THESIS STATEMENT | 7 |
| 1.7 THESIS OUTLINE | 8 |
| CHAPTER 2: COMPUTING SYSTEMS BETWEEN TRADITIONAL MANAGEMENT AND SELF-MANAGEMENT | 11 |
| 2.1. CHAPTER OVERVIEW | 11 |
| 2.2. TRADITIONAL MANAGEMENT OF COMPUTING SYSTEMS..... | 11 |
| 2.2.1. <i>Computing Systems Management Overview</i> | 11 |
| 2.2.2. <i>Framework of Computing Systems Management Systems</i> | 12 |
| 2.2.3. <i>Computing Systems Management vs. Computing Systems Manageability</i> 13 | |
| 2.2.4. <i>Managing Application Lifecycles</i> | 13 |
| 2.2.5. <i>Computing Systems Instrumentation</i> | 14 |
| 2.2.6. <i>Traditional Management Systems</i> | 15 |
| 2.2.7. <i>Computing Systems Management Challenges and Open Issues</i> | 16 |
| 2.3. COMPUTING SYSTEMS SELF-MANAGEMENT: AUTONOMIC COMPUTING | 17 |
| 2.3.1. <i>The Notion of Autonomic Computing</i> | 17 |
| 2.3.2. <i>Origins of Autonomic Computing</i> | 18 |
| 2.3.3. <i>Autonomic Computing Vision</i> | 19 |
| 2.3.4. <i>Autonomic Computing Characteristics</i> | 20 |
| 2.3.5. <i>Autonomic Computing System Architecture</i> | 23 |

| | | |
|---------|---|----|
| 2.3.6. | <i>Life Cycle of Autonomic Computing Systems Components</i> | 27 |
| 2.3.7. | <i>Relationships among Autonomic Computing Systems Components</i> | 29 |
| 2.3.8. | <i>Paradigms Significant to Autonomic Computing</i> | 30 |
| 2.3.9. | <i>Levels of Autonomic Maturity</i> | 33 |
| 2.3.10. | <i>Autonomic Computing Challenges and Open Issues</i> | 34 |
| 2.4. | CHAPTER SUMMARY | 37 |

CHAPTER 3: THE WEB SERVICES EDGE – THE INTEGRATION

BETWEEN BUSINESS AND MANAGEMENT 38

| | | |
|--------|--|----|
| 3.1 | CHAPTER OVERVIEW | 38 |
| 3.2 | WEB SERVICES | 38 |
| 3.2.1 | <i>The origin of Web Services: Service-Oriented Architecture</i> | 38 |
| 3.2.2 | <i>The Notion of Web Services</i> | 40 |
| 3.2.3 | <i>Web Services Characteristics</i> | 40 |
| 3.2.4 | <i>Web Services Architecture</i> | 41 |
| 3.3 | MANAGING WEB SERVICES | 43 |
| 3.3.1 | <i>Managing Web Services Using Traditional Management Systems</i> | 44 |
| 3.3.2 | <i>Tasks Involved in Managing Web Services</i> | 44 |
| 3.3.3 | <i>Web Services Management Framework</i> | 46 |
| 3.3.4 | <i>Extending Web Services Architecture to Support Management</i> | 47 |
| 3.3.5 | <i>Web Service Management Approaches</i> | 48 |
| 3.3.6 | <i>Manageability Interfaces in The Context of Web Services</i> | 49 |
| 3.3.7 | <i>Web Services Managing Example: AccountInfo Web Service Example</i> .. | 52 |
| 3.3.8 | <i>Events-Based Management</i> | 55 |
| 3.3.9 | <i>Web Services Management Patterns</i> | 56 |
| 3.3.10 | <i>Web Services Management Challenges</i> | 57 |
| 3.4 | USING WEB SERVICES IN MANAGEMENT | 58 |
| 3.4.1 | <i>Significance of Web Services to the Context of Management</i> | 58 |
| 3.4.2 | <i>The Emergence of Web Services-Based Management Systems</i> | 58 |
| 3.4.3 | <i>Composition of Web Services-Based Management Systems</i> | 59 |
| 3.5 | WEB SERVICES MANAGEABILITY | 60 |
| 3.5.1 | <i>Manageability Model</i> | 60 |
| 3.5.2 | <i>Manageability Aspects</i> | 61 |

| | | |
|--|---|-----------|
| 3.5.3 | <i>Manageability Capabilities</i> | 62 |
| 3.5.4 | <i>Web Services Manageability Framework</i> | 63 |
| 3.6 | CHAPTER SUMMARY | 63 |
| CHAPTER 4: RELATED WORK | | 65 |
| 4.1. | CHAPTER OVERVIEW | 65 |
| 4.2. | RELATED WORK | 65 |
| 4.2.1 | <i>Management Standards</i> | 65 |
| 4.2.2 | <i>Autonomic Web Processes</i> | 70 |
| 4.2.3 | <i>Adding High Availability and Autonomic Behavior to Web Services</i> | 71 |
| 4.2.4 | <i>Adapt: Towards Autonomic Web Services</i> | 72 |
| 4.2.5 | <i>Architecture for an Autonomic Web Services Environment</i> | 73 |
| CHAPTER 5: PROPOSED SOLUTION: WEB SERVICES – BASED AUTONOMIC COMPUTING FRAMEWORK..... | | 75 |
| 5.1 | CHAPTER OVERVIEW | 75 |
| 5.2 | SIGNIFICANCE OF THE RELATED WORK TO OUR RESEARCH..... | 75 |
| 5.3 | THE PROPOSED SOLUTION FOR ACHIEVING THE THESIS OBJECTIVE..... | 76 |
| 5.4 | WSAC FRAMEWORK ARCHITECTURAL MODEL..... | 77 |
| 5.4.1 | <i>Autonomic Computing Web Services</i> | 77 |
| 5.4.2 | <i>MAPE Web Services</i> | 77 |
| 5.4.3 | <i>Manageable Resources</i> | 82 |
| 5.5 | WSAC PROCESSING MODEL..... | 83 |
| 5.5.1 | <i>Autonomic Computing Requests</i> | 83 |
| 5.5.2 | <i>Autonomic Computing Infinite Loops</i> | 84 |
| 5.5.3 | <i>Autonomic Computing Rounds</i> | 84 |
| 5.5.4 | <i>Processing between Manageable Resources and Autonomic Computing Web Services</i> | 84 |
| 5.5.5 | <i>Processing between Autonomic Computing Web Services and MAPE Web Services</i> | 85 |
| 5.6 | WSAC METAMODEL | 90 |
| 5.7 | SCOPE OF WORK | 91 |
| 5.8 | CHAPTER SUMMARY | 92 |

CHAPTER 6: WSAC FRAMEWORK VALIDATION.....93

6.1 CHAPTER OVERVIEW93

6.2 PROOF-OF-CONCEPT PROTOTYPE.....93

 6.2.1 *Prototype Management Scope*93

 6.2.2 *Prototype Architecture*.....93

 6.2.3 *Prototype Communication Patterns*.....100

 6.2.4 *Revisiting the Prototype Management Scope*101

6.3 THE PROTOTYPE IMPLEMENTATION102

 6.3.1 *Prototype Implementation Assumptions*102

 6.3.2 *Prototype Implementation Tools*.....103

 6.3.3 *Prototype Developing Technique*.....104

6.4 PROTOTYPE TESTING104

 6.4.1 *Prototype Testing Setups*.....105

 6.4.2 *Prototype Test Cases*.....105

 6.4.3 *Prototype Testing Screen Shots*108

6.5 CHAPTER SUMMARY112

CHAPTER 7: CONCLUSION AND FUTURE WORK114

7.1 CHAPTER OVERVIEW114

7.2 RESEARCH SUMMARY114

7.3 CONTRIBUTIONS117

 1. *Loosely-Coupled Autonomic Computing*.....117

 2. *Autonomic Computing Web Services*.....117

 3. *MAPE Web Services*117

 4. *Hybrid Instrumentation*.....118

 5. *Upgrading Analytical Skills*.....118

 6. *Evolving Legacy Computing Resources*.....118

 7. *Splitted Channels*118

7.4 CHALLENGES AND NON-RESOLVED CONFLICTS119

7.5 FUTURE WORK120

 1. *Enhancing the Prototype*.....120

 2. *Expanding the Management Scope of the Prototype*.....121

 3. *Improving the Capabilities of MAPE-WS*.....121

| | | |
|---|---|------------|
| 4. | <i>Adding New Features to the Prototype</i> | 123 |
| 5. | <i>Accommodating Other Autonomic Computing Characteristics</i> | 123 |
| 6. | <i>Security Considerations</i> | 124 |
| LIST OF REFERENCES | | 125 |
| LIST OF PUBLICATIONS | | 130 |
| APPENDIX A: ACRONYMS AND GLOSSARY | | 131 |
| 1. | ACRONYMS | 131 |
| 2. | GLOSSARY | 132 |
| APPENDIX B: OVERVIEW OF RELEVANT STANDARDS | | 138 |
| 1. | WEB SERVICES STACK..... | 138 |
| 1.1 | <i>SOAP</i> | 138 |
| 1.2 | <i>WSDL</i> | 139 |
| 1.3 | <i>UDDI</i> | 142 |
| 1.4 | <i>WS-Resource</i> | 142 |
| 2. | LOGS REPRESENTATION..... | 143 |
| 2.1 | <i>Common Base Events</i> | 143 |
| 2.2 | <i>WSDM Event Format (WEF)</i> | 145 |
| 3. | WEB SERVICES MANAGEMENT | 147 |
| 3.1. | <i>Using Web Services Platform Notions in the context of management</i> ... | 147 |
| 3.2. | <i>WSDM List of Manageability Capabilities</i> | 150 |
| 3.3. | <i>WSDM List of Manageability Capabilities Elements</i> | 151 |
| 4. | AUTONOMIC COMPUTING MANAGEABILITY CAPABILITIES..... | 155 |
| 5. | SUMMARY OF NAMESPACES AND STANDARDS..... | 157 |
| APPENDIX C: OVERVIEW OF RELEVANT TOOLS | | 159 |
| 1. | IBM AUTONOMIC COMPUTING TOOLKIT | 159 |
| 2. | ECLIPSE SDK..... | 160 |
| 3. | ECLIPSE WTP PROJECT..... | 161 |
| 4. | APACHE TOMCAT APPLICATION SERVER | 161 |
| 5. | JAVA API FOR XML-BASED RPC (JAX-RPC)..... | 162 |
| 6. | NSCLIENT..... | 162 |

| | | |
|----|--|-----|
| 7. | NSCLIENT4J | 162 |
| 8. | COMMAND LINE PROCESS VIEWER/KILLER/SUSPENDER FOR WINDOWS NT/2000/XP | 162 |

APPENDIX D: WSAC PROOF-OF-CONCEPT PROTOTYPE

IMPLEMENTATION AND USE CASES 163

| | | |
|-----|-------------------------------------|-----|
| 1. | THE PROTOTYPE IMPLEMENTATION | 163 |
| 1.1 | <i>Sensor</i> | 163 |
| 1.2 | <i>Healing Web Service</i> | 165 |
| 1.3 | <i>Monitoring Web Service</i> | 167 |
| 1.4 | <i>Analyzing Web Service</i> | 167 |
| 1.5 | <i>Planning Web Service</i> | 169 |
| 1.6 | <i>Executing Web Service</i> | 170 |
| 1.7 | <i>Effector</i> | 170 |
| 2. | THE PROTOTYPE USE CASES..... | 171 |

APPENDIX E: WSAC PROOF-OF-CONCEPT PROTOTYPE J2EE CLASSES

SOURCE CODE 176

| | | |
|----|--------------------------------------|-----|
| 1. | SENSOR WEB SERVICE J2EE CLASS | 176 |
| 2. | HWS J2EE CLASS | 182 |
| 3. | MWS J2EE CLASS..... | 199 |
| 4. | AWS J2EE CLASS | 201 |
| 5. | PWS J2EE CLASS | 217 |
| 6. | EWS J2EE CLASS..... | 220 |
| 7. | EFFECTOR WEB SERVICE J2EE CLASS..... | 222 |

LIST OF TABLES

| | |
|--|-----|
| TABLE 1 LIST OF EXTERNAL TOOLS USED IN IMPLEMENTING THE POC PROTOTYPE | 95 |
| TABLE 2 LIST OF TOOLS USED IN IMPLEMENTING THE PROTOTYPE | 104 |
| TABLE 3 LIST OF ACRONYMS | 131 |
| TABLE 4 WSDM MANAGEABILITY CAPABILITIES URIS..... | 150 |
| TABLE 5 WSDM MANAGEABILITY ELEMENTS | 151 |
| TABLE 6 AUTONOMIC COMPUTING MANAGEABILITY CAPABILITIES URIS | 156 |
| TABLE 7 AUTONOMIC COMPUTING MANAGEABILITY ELEMENTS | 157 |
| TABLE 8 LIST OF NAMESPACES | 157 |
| TABLE 9 LIST OF STANDARDS | 158 |
| TABLE 10 LIST OF SENSOR WS METHODS | 163 |
| TABLE 11 LIST OF HWS METHODS | 165 |
| TABLE 12 LIST OF MWS METHODS | 167 |
| TABLE 13 LIST OF AWS METHODS..... | 167 |
| TABLE 14 LIST OF PWS METHODS | 169 |
| TABLE 15 LIST OF EWS METHODS | 170 |
| TABLE 16 LIST OF EFFECTOR WS METHODS | 170 |

LIST OF FIGURES

| | |
|---|----|
| FIGURE 1 AUTONOMIC COMPUTING FLOW CHART [2] | 20 |
| FIGURE 2 RELATIONSHIP BETWEEN MAJOR AND MINOR AUTONOMIC COMPUTING CHARACTERISTICS [17]..... | 22 |
| FIGURE 3 FUNCTIONALITY OF AUTONOMIC COMPUTING CONTROL LOOP | 23 |
| FIGURE 4 REFERENCE ARCHITECTURE OF AUTONOMIC COMPUTING | 23 |
| FIGURE 5 DETAILED COMPOSITION OF A TYPICAL AUTONOMIC MANAGER | 24 |
| FIGURE 6 AUTONOMIC TOUCHPOINT COMPOSITION [19] | 26 |
| FIGURE 7 MAPE CYCLES AS THE BUILDING BLOCKS OF AUTONOMIC COMPUTING SYSTEMS [20] | 27 |
| FIGURE 8 AUTONOMIC COMPUTING MATURITY LEVELS..... | 34 |
| FIGURE 9 MIGRATION FROM MONOLITHIC OBJECTS TOWARDS REUSABLE SERVICES [5] | 39 |
| FIGURE 10 WEB SERVICES TRIANGLE: ROLES AND OPERATIONS [23] | 42 |
| FIGURE 11 CREATING, PUBLISHING, AND INVOKING WEB SERVICES OVER THE INTERNET [5]..... | 43 |
| FIGURE 12 WEB SERVICES MANAGEMENT GENERAL FRAMEWORK [7] | 46 |
| FIGURE 13 REVISITED WEB SERVICES STACK..... | 47 |
| FIGURE 14 DIFFERENT OPTIONS OF IMPLEMENTING THE ROLE OF MANAGEABILITY CAPABILITIES PROVIDER [32] | 51 |
| FIGURE 15 IN-BAND MANAGEABILITY | 51 |
| FIGURE 16 OUT-OF-BAND MANAGEABILITY | 52 |
| FIGURE 17 COMPOSITION OF WEB SERVICES-BASED MANAGEMENT SYSTEMS | 60 |
| FIGURE 18 MANAGEABILITY MODEL [33] | 61 |
| FIGURE 19 WEB SERVICES MANAGEABILITY META-MODEL [33] | 63 |
| FIGURE 20 WEB SERVICES MANAGEMENT FRAMEWORK ARCHITECTURE [27]..... | 66 |
| FIGURE 21 WEB SERVICES DISTRIBUTED MANAGEMENT GENERAL ARCHITECTURE.... | 67 |
| FIGURE 22 PRINTING WEB SERVICE MANAGEABILITY STACK [32]..... | 68 |
| FIGURE 23 X SPECIFIC MANAGEABILITY CAPABILITY UML CLASS MODEL | 68 |
| FIGURE 24 PROPOSED AUTONOMIC WEB PROCESSES FRAMEWORK | 71 |
| FIGURE 25 COMPONENTS OF THE PROPOSED EXTENSION TO THE WEB SERVICES ARCHITECTURE [38] | 72 |

| | |
|---|-----|
| FIGURE 26 AUTONOMIC WEB SERVICES ENVIRONMENT [40] | 74 |
| FIGURE 27 ARCHITECTURAL MODEL OF WSAC FRAMEWORK | 83 |
| FIGURE 28 WSAC FRAMEWORK PROCESSING MODEL | 87 |
| FIGURE 29 MR, ACWS AND MWS | 88 |
| FIGURE 30 MR, ACWS AND AWS | 89 |
| FIGURE 31 MR, ACWS, AWS, PWS AND EWS | 89 |
| FIGURE 32 WSAC FRAMEWORK METAMODEL | 90 |
| FIGURE 33 ARCHITECTURE OF THE PROTOTYPE | 100 |
| FIGURE 34 SENSOR WEB SERVICE WEB INTERFACE | 109 |
| FIGURE 35 NO PROBLEM FOUND AT THE END OF AN AUTONOMIC COMPUTING ROUND | 109 |
| FIGURE 36 A PROBLEM WAS FOUND & SOLVED AT THE END OF AN AUTONOMIC COMPUTING ROUND: SCENARIO 1..... | 110 |
| FIGURE 37 A PROBLEM WAS FOUND & SOLVED AT THE END OF AN AUTONOMIC COMPUTING ROUND: SCENARIO 2..... | 111 |
| FIGURE 38 STOPPING THE NSCLIENT SERVICE | 111 |
| FIGURE 39 DETECTING THAT NSCLIENT SERVICE WAS STOPPED, AND STARTING IT AUTOMATICALLY | 112 |
| FIGURE 40 NSCLIENT HAS BEEN AUTOMATICALLY STARTED | 112 |
| FIGURE 41 SPLITTED BUSINESS AND MANAGEMENT CHANNELS | 119 |
| FIGURE 42 ENHANCED WSAC ARCHITECTURAL MODEL | 121 |
| FIGURE 43 SENSOR WS CLASS STRUCTURE..... | 164 |
| FIGURE 44 HWS CLASS STRUCTURE | 166 |
| FIGURE 45 MWS CLASS STRUCTURE..... | 167 |
| FIGURE 46AWS CLASS STRUCTURE | 169 |
| FIGURE 47 PWS CLASS STRUCTURE..... | 169 |
| FIGURE 48 EWS CLASS STRUCTURE..... | 170 |
| FIGURE 49EFFECTOR WS CLASS STRUCTURE..... | 171 |
| FIGURE 50 PUBLISHING WS USE CASE | 171 |
| FIGURE 51 STARTING AC INFINITE LOOP USE CASE | 171 |
| FIGURE 52 LOCATING HWS USE CASE..... | 172 |
| FIGURE 53 ESTABLISHING MANAGEMENT RELATIONSHIP BETWEEN THE OS AND HWS USE CASE | 172 |

| | |
|--|-----|
| FIGURE 54 GETTING AVAILABLE METRICS USE CASE..... | 173 |
| FIGURE 55 COLLECTING METRICS USE CASE..... | 173 |
| FIGURE 56 LOCATING MAPE-WS USE CASE | 173 |
| FIGURE 57 INVOKING MWS USE CASE..... | 174 |
| FIGURE 58 INVOKING AWS USE CASE | 174 |
| FIGURE 59 CHECKING ACTIVE PROCESSES USE CASE..... | 174 |
| FIGURE 60 CHECKING RUNNING SERVICES USE CASE | 174 |
| FIGURE 61 CHECKING SYSTEM LOGS USE CASE | 175 |
| FIGURE 62 INVOKING PWS USE CASE | 175 |
| FIGURE 63 INVOKING EWS USE CASE..... | 175 |
| FIGURE 64 EXECUTING HEALING COMMAND USE CASE..... | 175 |

LISTINGS

| | |
|--|-----|
| LISTING 1 SKELETON MANAGEABILITY OPERATION OF A GENERIC MANAGEABILITY INTERFACE [7] | 50 |
| LISTING 2 ACCOUNTINFO SERVICE WSDL DOCUMENT [7] | 53 |
| LISTING 3 ADDING MANAGEMENT [7] | 53 |
| LISTING 4 CUSTOMIZED MI [7] | 54 |
| LISTING 5 GENERIC MI [7] | 55 |
| LISTING 6 SAMPLE ECWS INTERFACE [7] | 56 |
| LISTING 7 SENSOR WEB SERVICE J2EE CLASS | 182 |
| LISTING 8 HWS J2EE CLASS | 199 |
| LISTING 9 MWS J2EE CLASS | 201 |
| LISTING 10 AWS J2EE CLASS | 217 |
| LISTING 11 PWS J2EE CLASS | 219 |
| LISTING 12 EWS J2EE CLASS | 222 |
| LISTING 13 EFFECTOR WS J2EE CLASS | 224 |

Chapter 1: Introduction

Chapter 1: Introduction

1.1 Chapter Overview

This introductory chapter introduces the research topic that highlights the area of our research, the motivation behind conducting the research, the problem definition, and the objective of our thesis; then, the thesis statement will be declared. At the end of the chapter the organization of the document will be introduced.

1.2 Research Topic

Although development paradigms and languages have been contributing to advancing *Computing Systems (CS)*, they have also been extending the complexity of such systems [1]. For example, by taking a look on a typical IT infrastructure of an enterprise, we find that it includes a wide variety of applications, computers, storage devices, and networking equipment. This has been resulting in increasing the complexity of CS to the extent that it is approaching the limits of human capabilities [1]. Since traditional *Management Systems (MS)* cannot manage the increasing level of complexity as well as dynamic relationships that are established at run time between different applications, there has been a real challenge regarding the management of CS. Therefore, without working out a solution to the problem that was raised by that management challenge, the implementation of new technologies, such as *Ubiquitous Computing*, where a huge number of different computing resources are connected over the Internet to serve different types of service requests would not be possible [1].

In addition to the concern of managing modern CS that was highlighted in the last paragraph, there is another concern regarding the development paradigms that are used for architecting and developing modern CS. By using traditional development paradigms, such as Object-Oriented and Component-Oriented paradigms, applications have been developed as a set of objects or components that are modeled in the form of tightly-coupled client-server architectures. Tightly-coupled applications are difficult to adapt to different dynamic changes; in addition, modifying such applications is a time consuming and a costly process. Therefore, there is a need for evolving a new development paradigm that is capable of accommodating new business requirements that are characterized with dynamic and seamless relationships between applications.

As a result of the dilemma of managing modern CS, the notion of *Self-Management* has become a hot topic of research for reputable vendors, such as IBM, HP, and SUN [2]. IBM took the initiative to introduce the new computing paradigm to the computing world, which is *Autonomic Computing (AC)* and it provides a set of commercial products that help developing *Autonomic Computing Systems (ACS)* [2]. Generally speaking, the term "*autonomic*" is derived from the human biology, where the autonomic nervous system continuously monitors all the body's vital functionalities without any conscious effort exerted by the human. For example, the nervous system regulates the number of heartbeats if they were changed by any external or internal condition [3]. Moreover, based on the reference architecture of the paradigm of AC that was introduced by [3], an ACS is composed of a set of computing resources that manage themselves together with a set of *Autonomic Managers (AM)* that is responsible for managing the whole environment that hosts those computing resources. Therefore, AM acts as a MS that provides autonomic behavior to the environment, in which, those AM are running; therefore, AM can be referred to as *Autonomic Management Systems (AMS)*.

As a result of the new business requirements, there was an urgent requirement for developing applications to be more scalable than tightly-coupled applications; and therefore, *Service-Oriented Architecture (SOA)* was introduced. In SOA, applications are developed as a set of services that can be dynamically integrated with each other through well-defined interfaces to form and reform applications as the needs of the business change [4]. *Web Services (WS)* can be considered as the most commonly realized application of SOA. A WS is an application that can be accessed using standard Web communication protocols, such as HTTP; moreover, WS can be easily reused without knowing about the actual implementation. In addition, a WS can act the role of both providers and consumers to both receive and send requests; therefore, the architecture of WS is *peer-to-peer*. As a result, WS can be adopted in implementing loosely-coupled applications that are capable of establishing dynamic relationships with each other without requiring any information about the implementation or requiring specific communication protocols to be used. The major characteristic of WS is the interoperability, where different platforms and standards can communicate. Therefore, WS are adopted in the context of integrating heterogeneous CS [5].

1.3 Research Motivation

After studying the paradigms of both AC and WS, which will be discussed in details in chapters two and three, the following conclusions were realized together to constitute the motivation behind conducting this research.

1.3.1. Autonomic Computing Conclusion

The notion of AC can be used to solve the problem of the increasing complexity that directly affects the task of effectively managing CS. However, the expected benefits from AC would be limited because of the inherited challenges in today's CS; one of those challenges is the heterogeneity that keeps arising because modern CS are usually composed of a variety of computing resources in terms of vendors as well as platforms. Therefore, heterogeneity will likely limit the management scope of AMS as that scope will be based on the type of resources being managed. Consequently, it was recognized that AC cannot rely on being a proprietary offering, but rather ACS must be based on open standards [6]. Fortunately, the Internet technologies provide the required networking infrastructure for connecting different AMS to heterogeneous computing resources; however, by themselves those technologies are not sufficient for addressing heterogeneity problem.

Although the problem of heterogeneity can be partially solved by using platform-independent development environments, such as *Java*, design approaches provided by such environments are unable to anticipate the interaction among different components and consider the runtime issues. Therefore, depending only on further advances in programming approaches is not a solution for the complexity problem that threatens the future of CS [1]. Therefore, the need for enabling AMS to communicate with CS for the purpose of providing AC behavior regardless of the implementation details of those CS has been comprehended. Consequently, there has been an imperative need for some integration layer that permits seamless-kind of communication between AM and MR; and therefore, AM can manage MR regardless of platforms and/or languages.

1.3.2. Web Services Conclusion

In some WS scenario, a dynamic relationship might be established among multiple entities (e.g. providers and/or consumers) in order to fulfill a single request submitted by one of those entities. Therefore, a failure of one entity might have the impact to

bring the whole relationship down; or in other words, make the processing of the submitted request fail, which might have some critical business implications. Therefore, the complexity that is expected to be associated with different WS scenarios will raise some challenges in the context of management. For example, since WS span multiple environments, managing WS cannot be based on using particular management technology; as an example, *Java Management eXtension (JMX)* is suitable for WS implemented in Java; however, it is not for those implemented using .Net technology [7].

Since the human-intervention management model cannot keep track of dynamic relationships that are established during different scenarios in WS environments, it was realized that such model is not suitable for managing WS. In addition, it was stated in [8] that any SOA infrastructure should be able to dynamically adjust itself to provide the expected service levels and business goals, and this adjusting should occur at runtime in a time-sensitive way. Therefore, managing WS should be automated in order to catch and recover from failures that occur at runtime to guarantee that *Business Operations (BO)* are performed properly. Therefore, there has been a crucial need for adopting some concepts that automate the management, such that management actions are initiated by WS not by WS administrators.

1.4 Problem Definition

According to the motivations of our research that have been discussed in the last section, the problem that is considered by our research can be defined as a two-directional problem with the first direction addresses the heterogeneity challenge in the context of AC, and the second direction addresses the management challenge in the context of WS. The two directions of the problem considered by our research can be stated as follows:

1. The inherited heterogeneity in the modern computing systems limits the management capabilities of AM while adopting the notion of AC; and therefore, the expected outcome from AC would be limited.
2. The human-intervention management model is not suitable for managing complex WS scenarios.

1.5 Thesis Objective

Based on the problem that is considered by our research, which has been identified in the previous section, our thesis has a two-directional objective. The first direction of the objective is concerned with suggesting an approach for permitting seamless management relationships between AMS and CS regardless of the underlying details of any of them, and the other direction focuses on adopting the proper technology for automating management of WS environments. Consequently, in our thesis, we suggest a solution in order to achieve the thesis objective. The motivations behind the suggested solution can be discussed as follows:

1.5.1. Permitting Seamless Management Relationships between AMS and CS

The vision of AC that was introduced by IBM can be further extended by decoupling AC capabilities from the underlying computing resources. This can be achieved by elevating that AC vision from the resource level to the process level; in that level, autonomic behavior can be achieved through a set of *Management Operations (MO)* in the form of processes that are dispatched between AM and MR regardless of the infrastructural details of MR. The WS paradigm provides the foundation for permitting the ability of composing different aspects independently to enable flexible integration of processes and applications; and therefore, the overall adaptability to changes that appear overtime is permitted. Consequently, WS can be used for evolving aspects that used to be ranked as secondary aspects, such as manageability, to primary aspects that can be designed, implemented, composed, and reused across different business services independently [9]. As a result, the required elevation of AC can be realized with the aid of WS.

Researchers in [10] stated that WS can be employed in the context of managing both software applications and hardware devices; this statement can be extended for the sake of solving the first direction of the problem considered by our research. In addition, it was stated in [3] that the significance of WS in the context of AC can be highlighted by referring to the following quote: "This architecture does not prescribe a particular management protocol or instrumentation technology because the architecture needs to work with the various computing technologies and standards that exist in the industry today -- SNMP, JMX, Distributed Management Task Force, Inc. (DMTF) -- as well as future technologies. Given the diversity of these management

technologies that already exist in the IT industry, this architecture endorses Web services techniques for sensors and effectors. These techniques encourage implementers to leverage existing approaches and support multiple binding and marshalling techniques.”

As a result, WS technologies can be adopted in building ACS to enable seamless, dynamic management relationships, which implement different AC capabilities, between AM and computing resources regardless of platforms and languages. For example, different computing resources can have their MC described by standardized management interfaces; consequently, AM can use those management descriptors to understand, monitor, and interact with those resources [6]. Similarly, as BO can be outsourced through WS, management capabilities can be also extended beyond enterprise boundaries in order to either provide those capabilities to computing resources or such management capabilities are dynamically integrated with each other in complex management scenarios [6].

1.5.2. Adopting the Proper Technology for Automating the Management of WS

Regarding the second direction of the problem that is considered by our research, which is concerned with automating the management of WS, the inherited characteristics of WS encourage the implementation of WS interfaces that can handle significant error conditions by automatically sending messages to MS; and then, the interfaces receive actions from MS to fix errors. Moreover, Managing WS can be made proactive, such that through WS interfaces MS can check the status of WS without waiting for the occurrence of failures [11].

By implementing AMS in the form of WS, AMS can receive messages in the form of management-related information from WS, or notifications when failures occur; and then, AMS can also apply management actions to WS in the form of either RPC-centric or document-centric messages that can be understood by WS; and therefore, the management actions can be executed on WS. Consequently, the reference architecture of AC that was introduced by IBM can be employed in managing WS by using WS interfaces for the purpose of management; and therefore, managing WS can be automated by adopting the notion of AC.

Therefore; in this thesis, we suggest a framework that is called *Web Services - Based Autonomic Computing (WSAC)*, which is based on both AC and WS in order to suggest a solution for each direction of the identified problem. In WSAC, AMS are

implemented by using WS technologies so that AMS act as Managing WS, and computing resources, such as hardware devices and software applications are instrumented by WS management wrapping interfaces. Through WS management wrapping interfaces, computing resources can discover and dynamically establish management relationships with Managing WS at the runtime in order to outsource the required management aspects from those Managing WS. Moreover, a proof-of-concept prototype was built in order to prove the effectiveness of the proposed framework in achieving the goal of this research; or in other words, to prove the concepts introduced by WSAC. An interesting finding is that solving the AC's heterogeneity direction of the problem that is considered by our research automatically leads to solving the WS' management direction of the problem. This is because when AMS are architected as WS, it will be a trivial task to enable WS to dynamically discover those AMS to start a management relationship, through which, AMS automatically monitor and manage WS.

1.6 Thesis Statement

Based on the discussion of this chapter, the statement of our thesis can be declared as follows:

1. By using WS technologies in architecting and developing both computing resources and AM, management relationships among both entities will be permitted to be established seamlessly and dynamically regardless of the underlying details of any of them. Consequently, AC aspects can be provided by AM to computing resources without being affected by the underlying implementation details.
2. The notion of AC can be adopted in the context of automating the management of WS in order to enable WS to be both automatically and dynamically monitored and managed with minimal human-intervention that is required in defining policies and information that are needed for permitting and regulating management. Consequently, runtime failures that occur in complex WS scenarios can be automatically caught and recovered.

An interesting finding is that accomplishing the first direction of the research's goal automatically leads to accomplishing the second one. This is because when AM are

architected as WS in the form of ManagingWS, it will be a trivial task to enable WS to dynamically discover those ManagingWS to start management relationships, through which, ManagingWS can automatically monitor and manage WS.

1.7 Thesis Outline

In addition to the preface, this document is composed of six chapters, which are organized as follows:

1. Chapter 2: Computing Systems between Traditional Management and Self-Management

In the first part of this chapter, the fundamentals of managing CS are reviewed, and the challenges accompanying traditional MS are highlighted. The second part of the chapter introduces the concept of AC, which can be used in managing complex IT infrastructures. Moreover, the challenges that hinder building ACS are discussed, while the focus of this research will be on the heterogeneity challenge in the context of AC. Therefore, by the end of the chapter, it should be clear that there is a need for adopting some techniques to work around the challenge of heterogeneity.

2. Chapter 3: The Web Services Edge: Integration between Business and Management

This chapter starts with introducing the SOA; and then, discusses different aspects of WS as the paradigm of WS is considered the most common industrial application of SOA. The second part of the chapter discusses different aspects related to the management of WS environments and highlights the challenges that threaten the goal of implementing management solutions that can fulfill management requirements of WS to provide the required level of reliability. Additionally, the chapter talks about using WS in the context of managing computing systems. After finishing this chapter, it should be clear to the reader that there is a need for adopting some technique to automate the management of WS in order to achieve the expected benefits of WS.

3. Chapter 4: Related Work

This chapter starts by reviewing some standards that are concerned with management in the context of WS. Following to that, the chapter will discuss some of the related work to the contexts of both AC and WS.

4. Chapter 5: Research Topic: Web Services – Based Autonomic Computing Framework

Based on what it was concluded from chapters 2 and 3, this chapter begins with defining the motivations behind this research, the topic of the research and the thesis statement. In order to achieve the goal of our research, the Web Services – Based Autonomic Computing Framework is proposed in this chapter. In the rest of the chapter, the theoretical aspects of the framework are discussed. At the end of the chapter, the scope of work will be identified in order to establish an awareness of the expected outcome from the research.

5. Chapter 6: WSAC Framework Validation

In this chapter, the practical aspects of the proposed framework by our research are discussed by introducing a prototype that implements both the architectural and processing models of the framework. The brief of different aspects related to the implementation of the prototype is introduced in this chapter; however, the implementation details together with a set of use cases that illustrate different scenarios in the prototype are presented in Appendix D. At the end of the chapter, the implemented prototype is validated with the aid of two test cases to ensure that the prototype proved the practicality of our research.

6. Chapter 7: Conclusion and Future Work

In this chapter, our research is summarized by highlighting the research objective and the milestones that were achieved while accomplishing that objective. In addition, the contributions of our research are discussed in order to emphasize the potential of this thesis. The challenges and the conflicts that were not resolved are also reviewed in order to identify the expected enhancements. Moreover, some of possible future work, such as enhancing the implemented prototype, adding new features, and accommodating new AC characteristics is suggested at the end of the chapter.

7. Appendix

The appendix of this document is divided into five parts; in appendix A, the acronyms included in the text are listed; moreover, the appendix includes a list of definitions for different terms used in the text. In appendix B, standards that are significant to our research are briefly discussed. In appendix C, the tools that were used in implementing the prototype are described in brief. The implementation and use cases

of the proof-of-concept prototype are discussed in Appendix D; and finally, the source code of the J2EE classes of the prototype is attached in appendix E.

Chapter 2: Computing Systems between Traditional Management and Self-Management

Chapter 2: Computing Systems between Traditional Management and Self-Management

2.1. Chapter Overview

This chapter is divided into two parts with the first part reviewing the fundamentals of managing CS as well as the advantages and disadvantages of traditional management techniques, and the second part introducing the concept of AC, which focuses on enabling CS to be self-managed in terms of configuration, healing, security, and optimization. At the end of the chapter, the significance of AC to achieve effective management in modern CS will be highlighted. We aim to show, by the end of this chapter, that AC can be considered the evolutionary step of evolving traditional management techniques to meet new management requirements that have been arising in modern CS.

2.2. Traditional Management of Computing Systems

CS are currently built by different vendors that use different technologies in developing, deploying, configuring and running resources and components composing those systems. Therefore, the issue of managing CS has been gaining the attention of several vendors [12]. This part reviews the fundamentals of managing CS as well as the challenges that might come to the surface in this respect.

2.2.1. Computing Systems Management Overview

One of the main objectives of managing CS is to guarantee a maximum availability of computing services to provide the expected benefits to the users of such systems [13]. The activities that are involved in the context of management should cover the entire lifecycle of each computing component, such as installation and configuration; however, the cornerstone of managing a CS is monitoring the performance of each individual component by collecting information in order to allow the analysis of that information to properly manage the entire system [7]; for example, metrics are collected through monitoring. Metrics of significance are determined from the scope of management; however, there is a set of common metrics that are important for different management contexts, such as *response-time* and *throughput*. Moreover, there are some other metrics that are not significant, but help in achieving better

management, such as *versioning* that is used for identifying upgrades and *accounting* that is used for either billing or auditing [13]. Security should also be managed in order to provide confidentiality, integrity, authorization, authentication, and privacy of both information and business processes involved in CS [13].

2.2.2. Framework of Computing Systems Management Systems

In [12], the following framework was introduced in the context of describing the general architecture of MS that can be adopted by different management approaches as well as it can be applied to different management scenarios.

I. Computing Resources

Computing resources include both hardware devices, such as networking devices and servers and software applications, such as DB applications. The providers of some computing resource have the role of virtually equipping that resource with two interfaces *Business Interface (BI)*, which is responsible for providing the resource's functionalities and *Manageability Interface (MI)*, which is responsible for enabling the resource to be managed by MS. In order to support different management technologies, MI should implement those technologies. Consequently, the complexity of MI has been increasing as new management technologies are standardized. Involving both the production and management islands in the issue of management can help both parties grow more collaboratively. A computing resource that is equipped with one or more MI is called a *Manageable Resource (MR)*; an MR can be mapped to both physical and virtual resources, such as virtual memory. However, computing resources that do not have integrated MI can be still managed as discussed below.

II. Management Providers

Management providers are the providers of management solutions, such as *configuration managers*, *performance monitors*, etc. Instead of supporting a particular set of management techniques by an MI, an *adapter* that is usually provided by the provider of the management technology is used, such that the MI communicates indirectly with MS through that adapter. Although the use of management adapters reduces the cost of development, it increases the cost of deployment and operation.

2.2.3. Computing Systems Management vs. Computing Systems Manageability

Management Capabilities are the capabilities that are provided by MS to manage CS. For example, the ability of monitoring is implemented by MS to monitor CS. On the other hand, *Manageability Capabilities (MCs)* are the capabilities that are provided by MRs to enable management systems to manage them. For example, the ability of being monitored is implemented by CS to enable MS to monitor and manage them. Consequently, the notion of management capabilities complements the notion of MC because the former is achieved by consuming the latter. Therefore, CS that expose MC can be called *Manageability Providers*, and MS can be called *Manageability Consumers*.

2.2.4. Managing Application Lifecycles

The lifecycle of a typical software application can be summarized in the following points: (1) *install* or *deploy*, (2) *start* (making the application available), (3) *execute* (receiving requests, processing received requests and generating a result to be delivered to requesters), (4) *update*, (5) *stop* (making the application unavailable), and (6) *uninstall* or *undeploy*. To manage the lifecycle of an application, the application should expose a set of *Application Programming Interfaces (APIs)* to provide a set of *Manageability Operations (MO)* to be performed (methods to be invoked) by MS, such as *start* and *stop*. MO can be customized to meet specific business needs; for example, *DataBaseBackup* is an MO that is specific for the domain of managing databases. An MO is only performed in the case of failure or abnormal behavior; therefore, after performing any MO, the application should behave normally. In addition to APIs, the application should expose a set of information related to its lifecycle. In [7], the following set of information was mentioned as the information that is related to application lifecycles and can be used in the context of applications management:

I. Identification

It includes description, version, a *Universal Resource Identifier (URI)*, which should be unique per domain, etc.

II. Availability

It represents the availability status of the application to determine whether the application is available and can receive requests, available and cannot receive requests (busy) or unavailable and cannot receive requests.

III. Metrics

They represent numeric and non-numeric information that reflects the health and performance of the application. The collected metrics are usually subjected to threshold analysis for deciding whether an application is performing normally or not.

IV. Configuration

It represents the information that, for example, includes parameters controlling the operational behaviour of an application or specifies how an application interacts with its neighbors (providers or customers). Configuration can be either static, which cannot be changed while the application is running, or dynamic, which can be changed at runtime without causing any disruption.

V. Events

They represent messages from an application to some MS to indicate different situations, such as configuration changes, failures, warnings, etc. Events can be emitted either periodically (heartbeats that indicate the health of application), or instantly whenever an interesting situation occurs. Based on the received events, MS react in order to recover applications from any discrepancy. Therefore, events assure synchronization between applications and MS.

2.2.5. Computing Systems Instrumentation

The target of instrumenting applications is to dynamically collect data that are significant to MS. Both MO and management-related information can be provided by either external or internal instrumentation [7]. In the case of external instrumentation, a set of files (*instrumentation files*) describing the application and its MC, log files and some utilities (e.g. scripts, command lines, etc) are defined and made available to be used by MS. There are several options for providing instrumentation files, such as: (1) provided with the application, (2) created during install and/or update, (3) provided by the application's operational environment, such as application servers, on behalf of the hosted application or (4) provided by MS. Such files aim at guiding MS

in managing applications. MS diagnoses status by analyzing the log files. Moreover, utilities automate the way, by which, MS manage applications.

Although, the external instrumentation option reduces both the development and the maintenance cost from the side of coding, the approach lacks customizability. Therefore, internal instrumentation might be needed to provide detailed information that is related to specific management domain. Internal instrumentation is provided as an embedded layer within the application itself. In this case, a management object that complies with some MI standard or conforms to specific management needs is created and published. Java Management Extensions (JMX) Management Bean (MBean) provides an example of internal instrumentation. An MBean is an object that is accessible via standard JMX to enable developers to expose application-specific MI. MBeans run in the local application environment, such as the application server (e.g. MBean Server), which provides adapters that bridge the application with the MS.

2.2.6. Traditional Management Systems

Traditional MS provide MO through third-party management applications that reside in the application's runtime environment. With the aid of such third-party applications, MS monitor applications in order to detect any violations; for example, measured values are compared against the thresholds that are determined by system analysts to satisfy the business requirements of the application [13]. In addition, in [7], it was stated that the architecture of most traditional MS is centralized. An *Agent-Based Management System (AMS)* is as an example of MS that has been used for many years in different management scenarios. In a typical AMS, a group of software agents is used to collect information related to management form applications and report this information back to one or more software managers, which run on one or more servers (e.g. cluster). A software agent usually runs in the same host of the application to establish a communication channel between the application and the manager(s). Therefore, a software agent polls the required information from the application as a response to the manager's commands. Originally, agents were meant to be as simple as possible, with most of the processing and control done at the manager-side [14]. However, management agents might be equipped with some intelligence so that it can recognize internal errors and correlate events before communicating any information to the managers [7].

2.2.7. Computing Systems Management Challenges and Open Issues

After discussing the fundamentals of managing CS, the major challenges facing management approaches are highlighted in this subsection as it was mentioned in [12]:

I. Realizing Business Objectives

In a typical organization, the impact of business decisions on MS used in that organization should be studied, such that the management objectives are not contradicting with the business objectives. Therefore, MS require that business processes are linked to separate computing resources, such that the performance of composite resources satisfies the overall business objectives. Therefore, the availability of computing resources must be realized by MS because that availability has a direct impact on achieving business objectives.

II. Dealing with the heterogeneity

Using different technologies in developing CS has been resulting in a complex collection of management technologies and techniques that are based on a variety of management standards and protocols. For example, many networking devices use *Simple Network Management Protocol (SNMP)*, JAVA applications use JMX, servers use *Common Information Model (CIM)*, etc. In addition, CS are composed of a set of both heterogeneous physical and logical computing resources.

Therefore, MS should manage CS regardless to operational environments, and management should be performed dynamically without degrading the operational performance. As a result of that conclusion, *Enterprise Management Solutions (EMS)* have been emerging with the main goal is reducing the complexity of using multiple, different management technologies in a unified manner that provides a single point of management for different kinds of computing resources within the boundaries of some enterprise. However, when it comes to spanning different administrative domains, where goals and requirements differ from one domain to another, EMS do not represent the suitable solution.

Consequently, the need of having *Inter-systems Management Solutions (IMS)* has become a critical issue, especially with the proliferation of distributed systems; IMS are concerned with integrating a variety of standards that are adopted by different CS. Therefore, the need for integration standards that accommodate various management standards has been created; such integration standards should keep evolving to

simplify the task of managing heterogeneous CS by including mechanisms for both retrieving management-related information and applying management-related actions.

III. Providing adaptability, scalability and reliability

MS should enable CS to easily adapt to changing requirements and environments by dynamically allocating resources as changes occur. Moreover, the distributed identity of modern CS hinders the goal of maintaining scalable and reliable management; therefore, MS can be migrated towards adopting the emerging distributed computing paradigms, such as WS. However, building purely decentralized MS may not be a good choice in the environments that have limited networking resources. Therefore, a hybrid management approach that merges between centralized and decentralized management can be implemented to provide the advantages of each approach. For example, MO, such as monitoring can be shifted closer to the application being managed in order to improve the reliability of management by reducing the impact of network availability. In addition, centralized managers can be used to provide a broad view of the environment in order to resolve any conflict between management and business.

IV. Coupling Management to Business

There should be some kind of coupling between business and management processes (BO and MO). For example, performance metrics can be retrieved from some resource by some MO and passed to some BO to decide whether the resource is suitable for that BO or not. By accommodating such aspects MO become an integrated part in the IT BO; and therefore, MO can be performed by using the same technologies that are used by BO.

2.3. Computing Systems Self-Management: Autonomic Computing

After reviewing the fundamentals of managing CS in the previous part, this part will discuss the details of the emerging self-management notion, which can be considered as an evolutionary extension to traditional management approaches discussed in the previous part.

2.3.1. The Notion of Autonomic Computing

In order to cope with the complexity of managing IT infrastructures in a reasonable time and/or cost, human intervention involved in the management tasks should be

minimized, or in other words, CS should be enabled to be self-managed. As a result, IBM introduced the concept of AC in order to facilitate the design, development, and deployment of self-managed CS [15]. The term AC comes from the autonomic nervous system of the human body, which is the system that regulates the body's basic functions without any conscious awareness of humans. Similarly, ACS handle the execution of management-related tasks on their own, without the need for human intervention. However, the human intervention will still be needed in high-level tasks that require specific kinds of experiences, such as specifying policies for dealing with different situations in order to meet business objectives.

Therefore, AC paradigm is conceived as an approach that helps reducing both cost and complexity accompanied with operating IT infrastructures. In an ACS, computing components – hardware (e.g. networking devices) and software (e.g. databases) monitor themselves and adapt to dynamic changes in either the operational environment or in business requirements. Such computing components can be referred to as *Autonomic Computing Systems Components (ACSC)*. However, dynamic adaptation should be driven by policies that capture business requirements. Moreover, applications should adhere to some quality factors (non-functional requirements), such as reliability and maintainability; therefore, AC aims at achieving the required quality factors.

2.3.2.Origins of Autonomic Computing

For addressing the issues of reducing complexity and building failure-resistant CS, many approaches have been emerging; some of the notable and efficient approaches in those respects are: *Object-Oriented Programming* and *Fault-Tolerant Computing*. As it will be discussed in the following two points, AC can be considered as a composite extension of those two approaches

I.Object-Oriented Programming

The concept of *abstraction* was introduced by Object-Oriented Paradigm in order to separate the implementation details of each software component in some application from other components. This separation is achieved by interfaces, through which, the components communicate with each other in the same application or in other applications. A software component in Object-Oriented Programming is an object, and by applying the concept of abstraction, an object can be implemented regardless

of the implementation of other objects. This provides an important feature that an object does not need to know or deal with the complexity of the internal details of other objects, on which, the former depends [16]. Abstraction can be used to decouple management from functionalities of objects therefore adding or removing MO does not affect BO, and similarly, adding or removing BO does not affect MO.

Moreover, the concept of *polymorphism* is one of the key features of Object-Oriented Programming, which can be used in having multiple implementations of the same object; and based on the class of inputs, the proper implementation is invoked. This can be used in the context of dynamically optimizing the performance of the whole CS for a specific class of inputs, which can be used as a primitive concept of reacting to the inputs, which can be further refined in ACS [16].

II. Fault-Tolerant Computing

The concept of detecting and correcting any fault out of a pre-defined set of faults was introduced by Fault-Tolerance designs, which require CS to include enough redundancy to make recovery from possible faults. This notion is further refined in ACS, such that input(s) and output(s) of each component are continuously monitored to detect any deviation; as a reaction to this deviation, the infected component either switches to another component to perform the task it is currently executing, or it adjusts itself in a certain way [16].

2.3.3. Autonomic Computing Vision

ACS will maintain and adjust themselves by continuously monitoring their behaviour and automatically adjusting their performance as a response to changes in ACSC, workloads, and hardware and software failures [3]. Although IBM is the founder of autonomic computing vision, this vision is based on Microsoft's Joseph Barrera III paper, Self-Tuning Systems Software [2]. In this paper, Barrera outlined the activities that are necessary to construct self-tuning systems as (1) an administrator of such systems defines the expectations of the system's performance in the form of high-level policies that will be translated in the runtime into some low-level actions, (2) the actual performance of the system should be measured by some means, (3) the measured performance is analyzed and compared with the expectations and (4) depending on the result of comparison, some actions are performed. Based on

Barrera's discussion, the activities performed in ACS can be defined with the aid of the figure [2]:

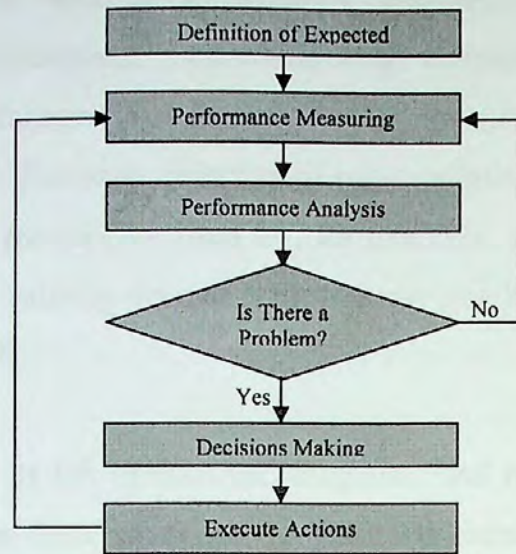


Figure 1 Autonomic Computing Flow Chart [2]

2.3.4. Autonomic Computing Characteristics

In order to guarantee autonomic behavior, an ACS is characterized with four major characteristics namely: *self-configuring*, *self-healing*, *self-optimizing*, and *self-protecting*, which are often referred to as *self-CHOP characteristic* [17]. In addition to those four major characteristics, ACS are characterized with additional four minor characteristics that are *self-awareness*, *openness*, *context-awareness*, and *anticipation* [18]. The following two subsections discuss both major and minor AC characteristics:

I. Major Autonomic Computing Characteristics

The major self-managing characteristics represent the core of an ACS as they define the tasks involved in configuring, healing, optimizing, and protecting the CS. Those tasks are automatically initiated and performed based on detected situations to enhance the ability of CS to dynamically react to different changes. For instance, in an ACS, a new resource is simply deployed and then optimization occurs; this is a notable shift from traditional implementations, in which a significant amount of analysis is required before deployment, to ensure that the resource runs effectively. Self-CHOP characteristic is discussed in more details in the following points:

1. Self-Configuring

It is the ability of CS to automatically adapt to dynamically changing operational environments as well as to dynamic changes occurring within the same operational environment. Self-configuring characteristic involves two main aspects: (1) installation and reconfiguration and (2) architecture adaptability on the ACSC level. The main objective of self-configuring is to enable the whole CS to dynamically reconfigure itself by, for example, deploying new ACSC and/or removing some existing ones to achieve some quality factors, such as improving performance [17].

2. Self-Healing

It is the ability of CS to discover, diagnose, and react to disruptions. A failed ACSC should be detected; and if the failure is simple enough, the failure can be fixed while the ACSC is online; otherwise, the ACSC should be taken off-line in order to be fixed; and during the fixing procedure, the failed ACSC can be replaced by another healthy one if available. This entire scenario should be conducted without interrupting the CS in order to maximize availability and resiliency. In addition, if the CS is enabled to predict, actions will be taken in a proactive manner to prevent failure from occurring.

3. Self-Optimizing

It is the ability of CS to effectively maximize the allocation as well as the utilization of the available resources to satisfy different requirements of different users. In addition, workload management can be employed to balance the load among all the ACSC; logical partitioning and clustering are examples of workload management techniques that manage the process of assigning tasks to the CS resources.

4. Self-Protecting

It is the ability of CS to consistently enforce security and privacy policies by anticipating, detecting, identifying, and protecting against attacks. First, the system should be defended against correlated problems resulting from malicious attacks, such as unauthorized access, virus infection and denial-of-service attacks or cascaded failures that remain uncorrected by self-healing measures. Second, early reports from sensors should be analyzed so that problems can be anticipated; and then, corrective actions can be taken to eliminate or reduce vulnerability.

II. Minor Autonomic Computing Characteristics

As it was mentioned by [17], minor autonomic characteristics aim at adding proactiveness to ACS to achieve intelligent decisions in the context of management. The following points discuss in brief the meaning of each minor autonomic characteristic:

1. Self-Awareness

An ACS should be aware of its state as well as its internal and external behaviors.

2. Openness

An ACS may be composed of heterogeneous ACSC as well as it may be operating in heterogeneous environments; therefore, it should be interoperable and portable across multiple administrative domains.

3. Context-Awareness

An ACS should be aware of the execution environment in order to react to changes.

4. Anticipation

An ACS should be able to anticipate occurrence of situations and prepare the required plans to improve the resiliency as actions should be performed immediately after the occurrence of such situations to minimize the outage time.

Both major and minor autonomic characteristics complement each other in order to provide the required quality factors. Figure 2 shows the relationship between the two categories of characteristics represented by the impact of each characteristic on the quality factors.

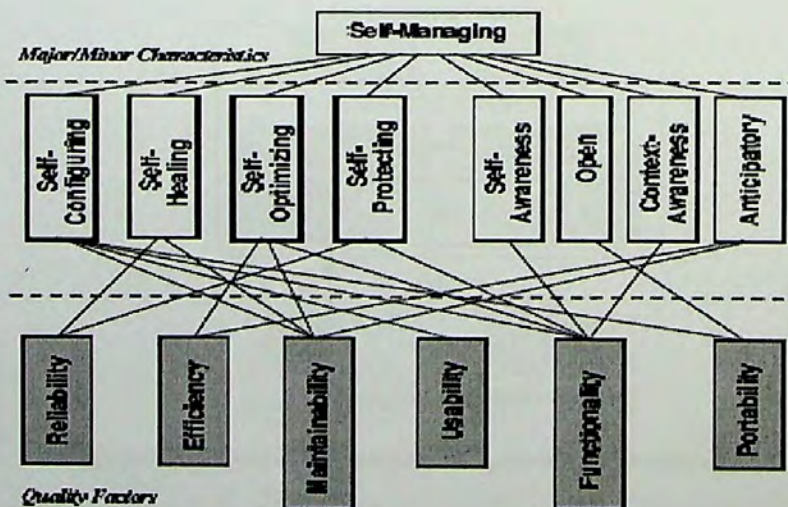


Figure 2 Relationship between Major and Minor Autonomic Computing Characteristics [17]

2.3.5. Autonomic Computing System Architecture

An ACS is composed of a set of integrated ACSC; moreover, and an ACS may collaborate with other ACS. Therefore, there are two levels of relationships in ACS: *intra-ACS* and *inter-ACS* relationships [17]. Within an ACS, the notion of self-management is implemented by embedding an intelligent *control loop* that helps achieving the self-CHOP characteristics described before in the ACS [1]. This loop collects information from the ACSC, analyzes the collected information, makes decisions based on the result of analysis; and then, adjusts the ACS as required. The functionality of AC control loop can be emphasized with the aid of the following figure:

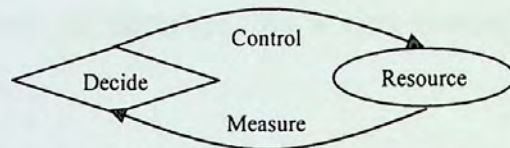


Figure 3 Functionality of Autonomic Computing Control Loop

In an ACS, each ACSC has a dual identity. One identity implements the computing functionalities of the ACSC (BO), and the other identity introduces some aspects of self-management (MO) [17]. Therefore, in the context of ACS, each ACSC is considered as a composite component composed of two subcomponents: *MR* and *AM*. Figure 4 shows the relationship between the two subcomponents, which illustrates the reference architecture of AC; following to the figure, the details of the architecture will be discussed.

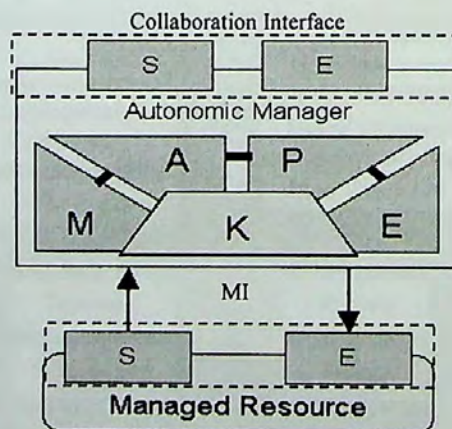


Figure 4 Reference Architecture of Autonomic Computing

I. Autonomic Manager

An AM implements a particular control loop that is composed of the following four integrated phases, which collaborate together to provide the control loop functionality:

1. Monitor

It provides mechanisms to collect, aggregate, filter, and report details (e.g. metrics) collected from an MR.

2. Analyzer

It provides mechanisms to correlate and model complex situations that allow the AM to learn about the IT environment and help predicting future situations.

3. Planner

It provides mechanisms to structure the action needed to achieve goals and objectives.

4. Executive

It provides mechanisms that control the execution of a plan.

The four phases consume and generate knowledge based on known information about the ACS, and this knowledge grows as the AM learns more about the characteristics of the MR. By having the knowledge shared among the four phases, decisions being made by the phases will be consistent. Based on the last discussion about AM, figure 6 can be revisited to illustrate the exact functionalities of each phase of the AM are highlighted as follows:

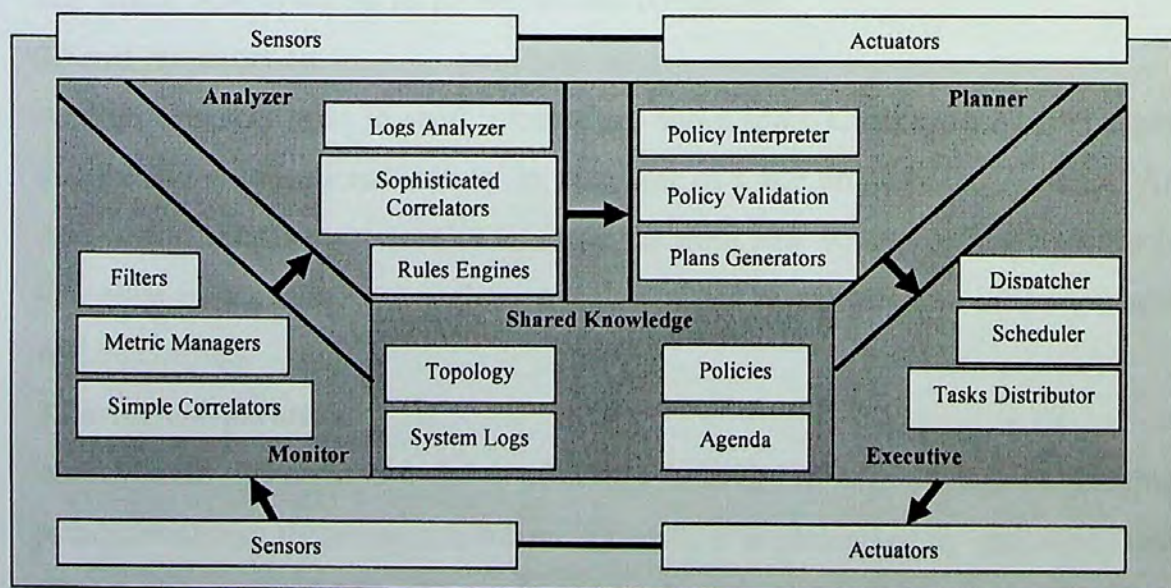


Figure 5 Detailed Composition of a Typical Autonomic Manager

II. Manageable Resource

An MR is managed by one or more AM, where each AM is responsible for some management capability. Moreover, an MR may represent a single resource, such as a server or router, or a collection of resources, such as a server farm. As shown in figure 5, an AM communicates with a MR through an MI.

III. Manageability Interface

An MI is also called an *Autonomic Touchpoint (ATP)*, and each ATP implements a set of *sensor* and *effector* operations (e.g. *get* and *set*) for one or more MR. Sensor operations are typically used to transmit events and information from MR to AM, whereas effector operations are typically used to cause some sort of changes in MR, such as altering state or setting property values. Moreover, the sensor and the effector operations should be linked together; for example, a configuration change that occurs through an effector should be reflected as a configuration change notification through the sensor.

IV. Manageability Interface Interaction Styles

The *sensors* and *effectors* operations captured by an ATP define the interaction styles that define how an AM and its managed resources interact with each other. As stated by [19], there are four unique interaction styles as follows:

1. Request-Response

An AM uses this interaction style to poll a MR in order to retrieve information about the current conditions. Each MR should implement the proper operations that enable AM to access its properties and operations.

2. Send-Notification

An MR uses this style to send information about some changes to AM; changes may be in the operational state or in the value of some property (e.g. metric). An AM should subscribe to receive notification from MR whenever the events it is interested in occur so that as those events emitted by an MR, the interested AM will be notified.

3. Perform-Operation

An AM uses this style in order to affect the behavior of MR through performing some operations; therefore, this interaction style is implemented by operations that change the state of MR. The values of only properties that are described as modifiable can be changed through this interaction style.

4.Solicit-Response

An MR uses this style to ask some third party for details or some service it may need.

The interaction styles as well as the function of a typical ATP are illustrated in the figure below:

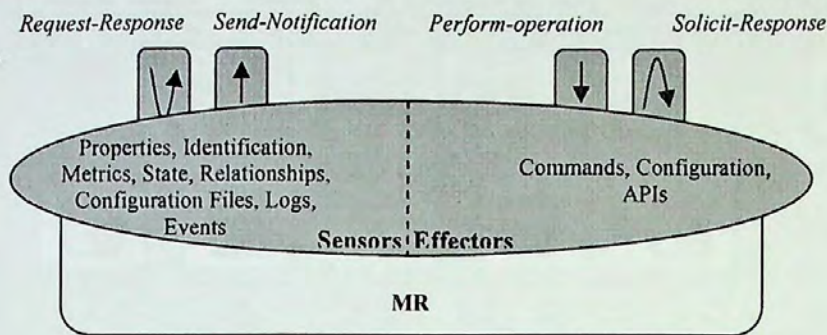


Figure 6 Autonomic Touchpoint Composition [19]

As mentioned by [20], interaction styles can be divided into two groups with the first representing interactions that return data (bi-directional interaction), which includes both Request-Response and Solicit-Response interaction styles, and the second representing data flows in only one direction (unidirectional interaction), which includes Send-Notification and Perform Operation interaction styles. Although the second group interactions do not return, they may return some information indication the success of failure of the interaction.

V.Collaboration Interface

An ACSC can communicate with other ACSC through its *collaboration interface*. Like an MI, the collaboration interface is equipped with sensors and effectors to facilitate the ACSC interaction with other ACSC and the interaction can be either in a peer-to-peer or a hierarchical manner [3].

In order to provide a system-wide view of ACS, figure 7 shows how ACS can be architected based on the reference architecture of ACS that was discussed in this subsection.

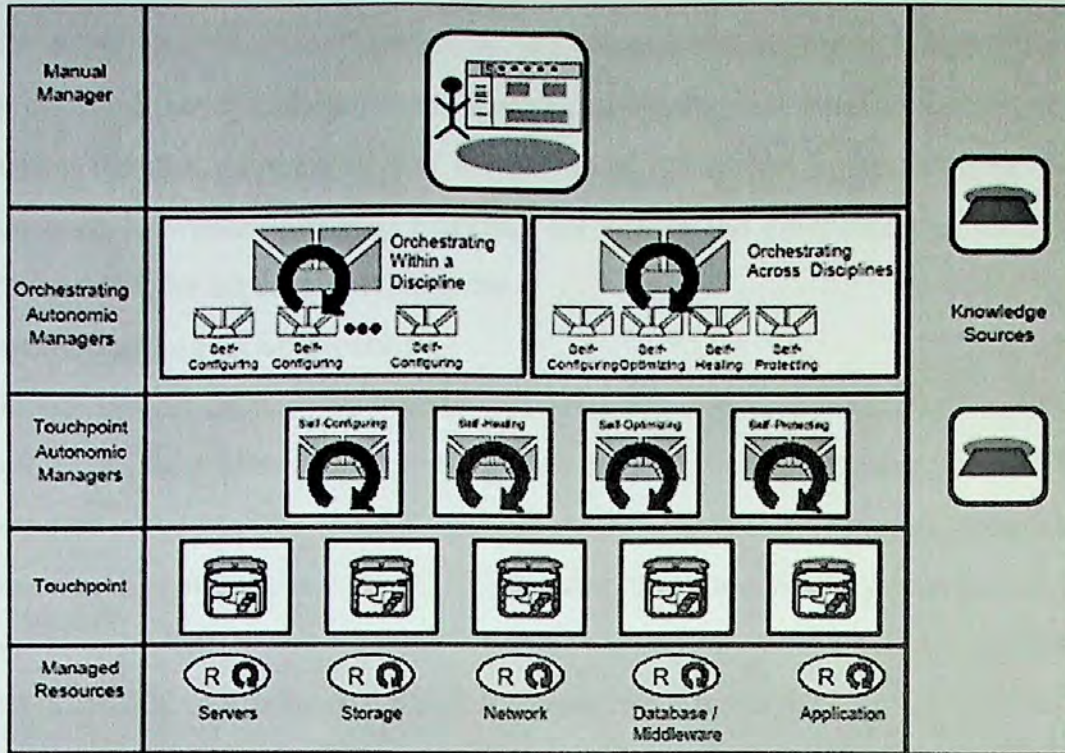


Figure 7 MAPE Cycles as the Building Blocks of Autonomic Computing Systems [20]

2.3.6. Life Cycle of Autonomic Computing Systems Components

The behavior of an ACSC will be continually carried on several threads of different activities. For examples, the ACSC will be continually sensing itself and responding to requests from outside or changes in the environment; the ACSC may be participating in one or more negotiation processes at various phases of completion; etc. Consequently, it is obvious that ACSC will be likely having complex lifecycles; the phases of that lifecycle are discussed in the following points as they were mentioned in [3]:

I. Design and Implementation

Programming an ACSC is obviously different from traditional programming, such as an object in an Object-Oriented System. For example, implementing an ACSC can be achieved by using tools that provide techniques for managing dynamic relationships among ACSC. *Utility functions* can be also used to capture the system goals and constraints and represent them in terms of high-level policies and rules. Moreover, there are some other programming requirements that should be considered by a typical autonomic programming tool, such as negotiating an SLA, establishing relationships, monitoring the relationship, and enforcing the agreement.

II. Testing and Verification

Since ACSC are likely to operate in unexpected environments, especially when multiple administrative domains are spanned, preparing test cases to test an ACS, by capturing the size, complexity, and workloads of the system in the run time, is not a trivial task. However, attaching a testing method to the description of each ACSC might be a helpful approach in this sense.

III. Installation and Configuration

Installing and configuring an ACSC will properly require a *bootstrapping process*, which begins when the component registers itself in some directory service. In the registration process, the ACSC publishes its capabilities, output, and contact information. Moreover, an ACSC might also use the directory service to locate suppliers or brokers to complete the initial configuration. An ACSC can also delegate a broker the task of finding customers for the services it provides.

IV. Monitoring and Problem Determination

AC characteristics cannot be supported without enabling each ACSC to be continuously self-monitored. During monitoring, information is logged about each ACSC; this logged information forms the basis for later adaptation. Not only an ACSC monitors itself, but also it monitors its relationships with other ACSC (suppliers or consumers) to ensure that it provides or receives the agreed-on level of service. When monitoring is coupled with events correlation and other forms of events analysis, this will help determining problems; and then, supporting recovery from failures. However, applying monitoring, auditing, and verification testing will be likely exhausting the ACS resources by either excessive traffic on networks or processing on nodes; therefore, sample-based statistical analysis might prove useful in providing a cost-effective, but yet efficient, monitoring solutions.

V. Upgrading

An ACS needs to upgrade itself as time runs; one approach to achieve this is that ACSC might subscribe to a third-party service, which is responsible for notifying subscribed components whenever new relevant updates are available. Another approach is that a fresh version of the ACSC is created and replaces the outdated version of the ACSC; however, the old ACSC would not taken offline unless the new one proved that it works properly.

2.3.7. Relationships among Autonomic Computing Systems Components

As each ACSC has its own life cycle, relationships among ACSC also have lifecycles, such that each lifecycle is composed of the following phases, which are listed in [3]:

I. Specification

Each ACSC is a provider of some services and a consumer of some other services. Therefore, ACSC specify their *input services*, which are services needed from other ACSC, and their *output services*, which are the services provided to other ACSC. As mentioned in installation and configuration phase of the lifecycle, an ACSC publishes its output services in some public directory service. The services registered in directory must be expressed in format that is understandable by other ACSC. The description of a service includes the service's capabilities and contact information, such as protocols the other ACSC can use to communicate with the service providing ACSC; researches in the area of service ontologies may be helpful for automating the process of service discovery and binding.

II. Location

An ACSC also uses the directory service to locate the input services it needs; consequently, ACSC can be located by looking them up either by the name or function. The Locating process might be a search process that involves reasoning about service ontologies. Once an ACSC providing the needed input service has been located, the requester ACSC uses the contact information to directly communicate with the providing ACSC. An important issue to be considered is judging the trustworthiness of the discovered ACSC via the directory service.

III. Negotiation

Once a potential ACSC providing some input service that is needed by another ACSC has been located, the requesting ACSC starts to negotiate the located ACSC. There are many forms of negotiation, such as *demand-for-service*, in which, the providing ACSC must process the request unless it does not have sufficient resources to do this and *posted-price*, in which, the providing ACSC sets a price for each of its service in some electronic currency, and the requesting ACSC has the choice to take it if it owns the price or leave it if it does not. Negotiation can be multilateral, in which a third-party facility can run an auction for the providing ACSC or a bid for the requesting ACSC. In addition, negotiation can be performed on multiple attributes, such as service level and price. The negotiation attributes should be expressed in an

understandable format to ACSC. Proposed negotiation protocols should establish the rules of negotiation and manage the flow of messages during negotiation.

IV.Provisioning

Once an agreement has been signed between the requesting and providing ACSC, each party must provision the required internal resources. Provisioning may include a simple access list that gives each party the permission to access specific resources on the other. Provisioning may also cause other relations to be established with other ACSC.

V.Operation

Operation is monitored by the two parties to enforce the level of service either provided or consumed. Monitoring is achieved by the AM of ACSC. If the agreement has been violated, one or both of the ACSC would perform some actions, such as the requesting ACSC might start looking up another providing ACSC via the directory service or a penalty might be assessed on the violating component.

VI.Termination

Each service agreement has a lease that specifies the duration of relationship. As the relation expires, the requesting ACSC might negotiate the providing ACSC to renew the lease, or the two parties might agree on terminating the relationship. By terminating the relationship, each party releases the reserved resources for the relationship.

2.3.8.Paradigms Significant to Autonomic Computing

Supporting AC capabilities would require collaboration from different computing paradigms. The following two subsections discuss two of those paradigms:

I.Tracing and Logging

Current IT systems are likely to include a mixture of different type of components, such as Web servers, databases, storage devices, etc. In addition, there might be different products from different vendors, where each product uses its own logging scheme and format and defines and uses specific events. Practically, a failure on one component in the CS might be the result of a failure that was occurred somewhere in the system. Thus, determining the root cause of errors in such systems is obviously a great challenge. There are many researches that are being done to work around that challenge; in [21], a solution that focuses on creating a common format for reporting

errors in the form of messages was proposed. In that proposed solution, each message adheres to the following content: (1) *Observing Component*, which is the component observing a problem (2) *Impacted Component*, which is the component that currently experiences the problem; therefore, it is very important to have each component uniquely identified, especially, in highly-populated environments like ACS and (3) *Situation Details*, which explain the observed situation using common terms both syntactically and semantically.

Reporting different situations should be consistent across the whole system; in this respect, there have been some trials in order to achieve a consistent way for reporting different situations. One of those trails was initiated by IBM in order to create a set of canonical situations; this was achieved by looking at plenty of log files to establish a small set of reported situations. The surprising result was that: although there were a few events that could not be categorized, from the tested numerous number of log files, there were less than two dozen categories of logged events, within each, there were many different ways of expressing the same situations [21]. For example, a set of situations including *start*, *stop request*, *configure*, and *connect* can be expressed by plethora of different words. Therefore, the challenge is how to unify the way, by which, situations can be expressed, especially, in heterogeneous environments that include a mixture of vendors, platforms, and technologies. Moreover, it is very difficult to convince customers to update their systems in order to adhere to some formatting approach.

One feasible solution is to install an *adapter* in each system to translate legacy log information into some common situation format.

Translated information can be used for sophisticated analysis that is used to determine the root cause of discovered errors; for example, the Analyzer of each ACSC can work on the log files to make its decisions; after analysis, the Analyzer may add the result of analyzing log files in the form of *symptoms* to the shared knowledge base, and those symptoms can help the Analyzer in its future tasks to fine tune its decision making process [21]. Moreover, traveling information might overload the CS by exhausting the system's resources; therefore, most of the analysis should be kept as close to each component as possible; however, if there is a real need to exchange information through the CS, that information should be aggregated and filtered [21].

II. Policy-Based Computing

There are many different definitions of what is meant by a policy; for examples, the *Internet Engineering Task Force (IETF)* defines a policy as a set of guiding actions; the *British-based Ponder* views a policy as a rule defining a choice that determines how the system behaves. There are many vendors that are concerned with policy-based computing, such as IBM. For IBM, policies can be considered as a set of guidelines that are determined, by human administrators of CS, to guide the process of automatic decision making; therefore, policies are the guidelines for the CS behavior. For example, optimizing policies that adjust the configuration of resources eliminate the need to configure each individual resource when the load on the resource changes [21].

By abstracting the functionalities, provided by the CS, in the form of service-level objectives, those objectives can be mapped to policies. Then, for example, an SLA can be monitored against those policies to determine whether the objectives of the CS are met or not. In the control loop of each ACSC, policies are stored in the shared knowledge base and are interpreted by the *Planner*, which makes sure that the CS objectives are met; a policy can be designed to provide different classes of users with different classes of services. For example, customers of some WS are divided into three classes: gold, silver, and bronze, where gold customers are guaranteed the shortest response time, and the response time experienced by bronze customers is determined in a best-effort manner. Therefore, the response time can be monitored for each class of customers in order to provide each class of customers with the expected level of service [21].

As it was mentioned in [21], IBM organizes the composition of policies as that each policy is made up of the following items:

1. Scope

It is the scope of applying certain policy to resources, or in other words, the scope of a policy determines, to which resources in the CS the policy should be, and, to which resources the policy should not be applied. For example, a *Backup Policy* may be applicable to databases but not to Web servers. However, the policy might be *capability-based*; for example, the capability of the Backup Policy is the backup process; therefore, the Backup Policy can be applied to every resource that can be backed up, regardless of the resource type.

2.Preconditions

The preconditions of a policy enable the CS to decide which policies are relevant to which situations; or in other words, the preconditions of some policy specify the situations, in which, that policy is applicable. For example, there might be two policies, one is applied in working hours, and the other is applied in the week ends.

3.Intent

The intent of a policy captures the intended result of applying a particular policy, such as "perform a backup" or "provide an x-second response time".

Supporting policy-based computing requires a set of tools that are able to capture, define, validate, enforce, and store policies. Defining a policy should be validated to ensure that the policy does not contradict with the CS objectives. In addition, it is possible that a set of policies could be destructive if they are not applied all together; therefore, there should be some means to enforce this requirement by firing all the policies that should be fired together. Moreover, policies might conflict with each other in the run time; therefore, policies should be interpreted correctly, and if there is any conflict between two policies, this conflict must be resolved by validating the defined policies. If policies aren't well validated to ensure their absolute correctness, they will be like a back door for different attacks [21].

2.3.9.Levels of Autonomic Maturity

Incorporating AC capabilities into CS is an evolutionary process that should be incrementally enabled by technology. Regarding the maturity of the way, by which, CS can be managed until they evolve to purely AC behaviour, the following five levels of maturity are listed by [3]; through those levels, CS can gradually accommodate different AC aspects and technologies:

I.Level 1 (Basic)

At the basic level, all functions are manually performed by IT staff, such that IT professionals manage each infrastructure component independently by setting it up, monitoring it, and eventually replacing it when it is needed.

II.Level 2 (Managed)

At the managed level, MS can be used to collect information from disparate CS onto fewer consoles.

III. Level 3 (Predictive)

At the predictive level, analysis capabilities are introduced in the CS to monitor situations that arise in the environment, and analyze them to provide suggestions of actions. The IT professionals decide on what course of actions to take.

IV. Level 4 (Adaptive)

At the adaptive level, the IT environment can automatically take actions based on the available information and knowledge of what is happening in the environment. CS can progress to the adaptive level as analysis and algorithmic technologies improve.

V. Level 5 (Autonomic)

At the autonomic level, business policies and objectives govern the operation of IT infrastructures. IT professionals interact with the autonomic technology tools to monitor business processes, alter the objectives, or both.

Figure 8 summarizes the maturity levels of AC that have been discussed above:

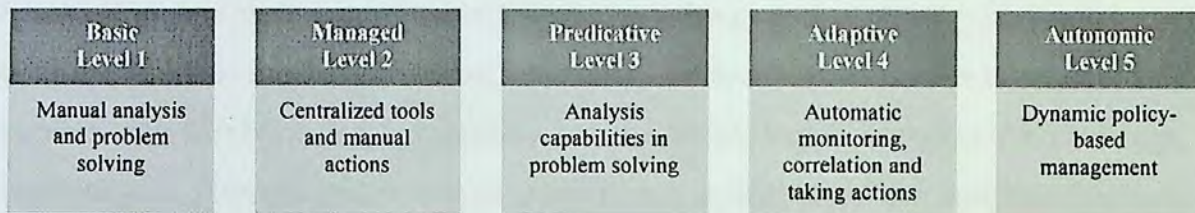


Figure 8 Autonomic Computing Maturity Levels

2.3.10. Autonomic Computing Challenges and Open Issues

Although there have been many academical and industrial efforts and projects in the area of AC, there are still some open issues. The following subsections review some of those issues:

I. Choosing Suitable Paradigms

There is a wide range of techniques that can be used in designing, developing, testing, and maintaining ACS. Each technique has its own merits and weaknesses; hence, identifying which ones are most appropriate for which activity and how the different techniques can be combined together is a significant issue [17]. Moreover, in [1], it was stated that the objectives of AC can be achieved by using some well-known paradigms, such as *Agent-Oriented Architecture*, *Component-Oriented Architecture*, *Control Theory*, *Grid Computing*, and *Software Engineering Methodologies*. The following two points highlight how the first two paradigms might contribute to building ACS:

1. Agent-Oriented Architecture

As it can be deduced, software agents are inherently autonomous, goal-oriented, and proactive. Therefore, there is a clear correspondence between the characteristics of the agent-oriented architecture and the objectives of AC; consequently, ACSC can be considered as agents and ACS can be considered as multi-agent systems [1].

2. Component-Oriented Architecture

Component-Oriented Architecture is inspired from the concepts of encapsulation and self-sufficiency. Interfaces of ACSC can be enhanced to encapsulate functional, operational and control aspects; this enables ACSC to export information regarding behavior, resource requirements, and interaction to other ACSC [17].

II. Applying Autonomic Computing Capabilities to Legacy Systems

Legacy Systems are the production CS that have been up and running for long period of time. Some of those systems are invaluable, such that it could not be replaced by modern CS. Evolving legacy systems to new technologies introduces a challenge, especially in systems where source code is not available or the coupling between components is very high; therefore, in the context of AC, it is virtually impossible to include monitoring and actuating functions to legacy systems. Working around that challenge was done by researchers in [17], which asked a new question that: is it possible to develop AC functionalities that are separated from application functionality by using, for example, *Aspect-Oriented Programming (AOP)* technology? The answer to that question would provide a very good solution for adopting legacy systems to accommodate AC notion.

III. Achieving Interoperability in Autonomic Computing Systems

Migration towards e-business and future CS is motivated by some key factors, one of which is interoperability. Lacking open standards that address the issue of interoperability introduces a challenge for getting ACS succeed in IT industry [17]. Moreover, after describing the reference architecture of ACS, a real ACS can be thought of as a CS of CS; therefore, the expected degree of complexity is very high. Consequently, interoperability should be considered to exist among both ACSC and ACS while the latter are being built [22].

As it was highlighted by some recent studies that new CS that are designed to interoperate with either legacy or new CS while adhering to common standards fail to interoperate properly [22]; therefore, achieving interoperability is not an easy task. Although the exact reasons are scenario-dependent, the failure reasons were concluded by [22] as follows: (1) interoperability between new and legacy systems is often rewound back to maintain compatibility with legacy systems, which cannot be upgraded unless a tremendous work is done, (2) specifications of standards are strict enough to allow flexible levels of interoperability and (3) current business policies are enough tough to achieve inter-domain interoperability.

Moreover, the following points that detail the problems threatening achieving the required level of interoperability were highlighted in [22]:

1. Individual Systems vs. Composite Systems

The majority of modern CS are constructed of integrated components; therefore, such CS are actually composite systems. The integration process might not have the proper control over components from different domains. Moreover, to cope with both operational and business changes, the capabilities of CS should be constantly upgraded. In the case of CS of CS, adaptation affects the whole CS as well individual CS. However, there might be the case that the changed demands placed on an individual CS are not the same, or even incompatible, with demands planned for the composite system; therefore, maintaining interoperability in this case is an ongoing problem. In order to maintain interoperability, new approaches are needed to analyze the impact of change on both individual systems as well as composite ones, structuring individual systems to compose the composite ones while meeting the new requirements, and verifying interoperability expectations when individual systems are deployed.

2. Domain Independency

Current versions of interoperable CS are designed to interact with some particular CS not with any CS. In order to support seamless relationships among CS, the design of future CS should emphasize on dynamic reconfiguration to support operating in different, unexpected domains.

2.4. Chapter Summary

The expected output from any CS cannot be guaranteed unless such systems are properly managed. Management tasks should be orthogonally performed with business tasks in order to ensure that the latter are properly performed. Although there have been many approaches introduced for implementing MS, such as centralized and agent-based solutions, in all approaches, the management-related actions are initiated and performed by human-administrators. With the advance in computation standards and technologies, the complexity of CS has been increasing; and consequently, having modern CS properly managed has become a crucial issue as the traditional human-intervention management model is no longer suitable for managing the reached level of complexity.

As a result, IBM introduced the notion of AC was to the computation world as the paradigm that allows CS to seamlessly manage themselves, or in other words, the paradigm of self-management. The AC characteristics cover different management scopes, such as configuration, healing, optimization, and protection. In addition, some other characteristics are considered to add intelligence, such as prediction. The core of building ACS is the AM that is responsible for enabling different ACSC to behave autonomically without the need for human intervention; however, human-expertise is still needed in high-level tasks, such as setting the CS-wide policies and goals. Moreover, the core of AM is the autonomic control cycles that provide measuring and deciding capabilities to enable AM to control the behavior of MR.

This research addresses one of the significant open issues of AC, which is concerned with openness, or interoperability. Specifically, our research is concerned with allowing seamless communication between MR and AM to enable the latter to manage the former regardless of platforms and/or languages.

Chapter 3: The Web Services Edge: The Integration between Business and Management

Chapter 3: The Web Services Edge – The Integration between Business and Management

3.1 Chapter Overview

This chapter highlights the significance of WS to modern CS by discussing different aspects of the WS paradigm. Generally, the chapter is divided into two major parts with the first part focusing on reviewing the basics of WS, and the second part is concerned with discussing the notion of management in the context of WS. Therefore, the first part of the chapter starts with briefly defining the SOA as the emerging paradigm for architecting software applications; and then, the fundamentals of WS are introduced to establish a proper awareness of the significance of WS in building modern CS. The second part of the chapter is further divided into four sections as follows: The first section discusses how WS can be managed, the second section discusses how WS can be used to provide management capabilities, and the third section reviews the notion of manageability in the context of WS. At the end of the chapter, the importance of WS in building both modern business and management CS will be highlighted; moreover, the management requirements of WS will be identified. Therefore, by the end of this chapter a relationship between this chapter and the previous one should be established and made clear to the reader because of the importance of that relationship to the rest of the document.

3.2 Web Services

WS can be considered as the most commonly realized application of SOA; therefore, this part starts with briefly discussing SOA as the origin of WS; and then, different aspects of WS will be discussed to emphasize the significance of WS to modern CS.

3.2.1 The origin of Web Services: Service-Oriented Architecture

In traditional programming paradigms, enterprise applications have been modeled as monolithic computing components; in addition, such components have been habitually architected in the form of tightly-coupled client-server architectures. Such applications have clear boundaries that prevent the reusability of business information as well as processes. Moreover, within the same enterprise, tightly-coupled applications are difficult to adapt to dynamic changes as well as modifying, upgrading

and enhancing such applications are time consuming, costly and risky. Since business requirements are captured by enterprise applications, organization's IT infrastructure became the bottleneck while attempting to adapt the business to changing market conditions [4].

In order to create applications that are more cost-effective, scalable, reliable, available, and flexible than monolithic applications, SOA was introduced. In SOA, applications are developed as a set of integrated components that interact with each other through well-defined interfaces; such integrated components are referred to as services [4]. Therefore, SOA breaks monolithic applications into distributed, smaller, more focused computing components, where each component can be owned, maintained or reused independently [13]; moreover, components are glued together to form and reform applications as the needs of the business change [4]. From this discussion, the vision of SOA may be abstracted as it is shown in figure 9.

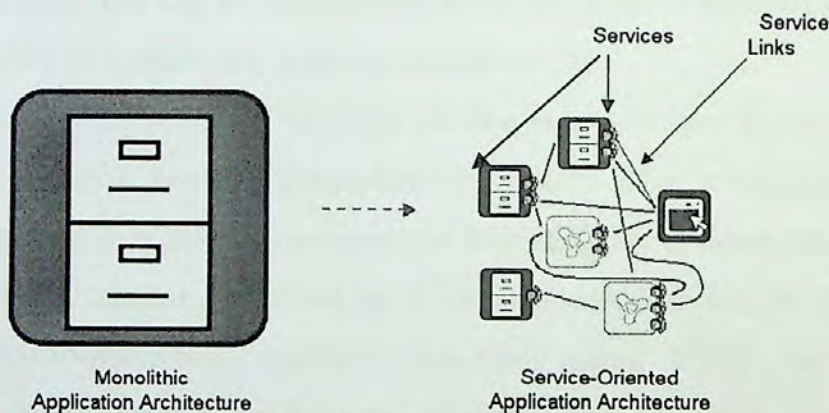


Figure 9 Migration from Monolithic Objects towards Reusable Services [5]

Since existing services, within the enterprise or across the enterprise-boundaries, can be further integrated in order to create new services, both time and cost of developing enterprise applications are reduced dramatically [4]. In addition, the modularity characterizing service-oriented applications enhances the process of troubleshooting in the case of failure. Consequently, in [4], it was concluded that: in SOA, a change is easy and affordable to some reasonable extent.

In the beginning of SOA, clients were accessing services through distributed-computing approaches, such as *CORBA*, *RMI*, *DCOM*, etc. Although such protocols proved efficiency, they are dependent on implementations, platforms, languages, or data encoding schemes [5]. Therefore, the interoperability of services was limited in a

way that prevented services from being accessible over the Internet. By taking all the best features of SOA and combine them with the Web technologies, the concept of WS was emerged to support web-based access and easy integration [5]; WS will be discussed in details in the rest of this part.

3.2.2 The Notion of Web Services

A WS is a discrete unit of business, an application or system functionality that can be accessed using standard Web communication protocols, such as HTTP and data formats, such as XML over any Web environment, such as Internet, intranet, or extranet; moreover, WS can be easily reused without knowing about the actual implementation. Because of the distinct features of WS, they were adopted for integrating heterogeneous IT systems in a way that enables enterprises to make use of their legacy systems. A WS customer can be either an end-user or another WS; therefore, multiple WS can be orchestrated to deliver a more sophisticated WS or to create a new WS to support new business objectives [5].

W3C defined WS as follows:” A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. Other systems interact with the Web Service in a manner prescribed by its description using Simple Object Access Protocol (SOAP)-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards”. Therefore, WS enabled machines to communicate with each other by some standardized messages in a loosely-coupled manner regardless of the hardware or the software the machines use [14]. Consequently, with the aid of WS, heterogeneous environments can be easily integrated in merging scenarios.

3.2.3 Web Services Characteristics

As it was mentioned by [12], the most compelling characteristics of WS can be summarized in the following points:

I. Platform Independency

WS provide access to services through standard transport protocols to maximize the opportunity of accessing services.

II. Legacy Systems Reusability

Existing software resources can be exposed as services using WS technologies so that those resources can be easily extended or adapted. Therefore, the need of replacing legacy CS will be eventually eliminated.

III. Loose-Coupling

By reducing the information that should be shared between WS, which is only the WS interfaces, the relationships that are established among WS are loosely-coupled. For example, an already running WS can be either replaced or evolved, or a new WS can be started without disrupting the whole WS-based CS. Consequently, through isolating the effect of changes, the overall development and maintenance cost are reduced.

IV. Distributed Computing Standardization

WS technologies have become the common point of researches concerned with distributed computing.

V. Industry Support

Major software vendors support the core technologies of WS, such as SOAP, WSDL, XML, UDDI and other. This makes WS the first choice of integration middleware as well as building SOA systems.

VI. Composability

Composition is one of the concepts considered by WS, in which, capabilities are allowed to be mixed and matched to fulfill dynamic requirements. For examples, security can be enforced on the message-level, delivery of messages can be made reliable, etc. This allows WS technologies to be applied in different scenarios.

3.2.4 Web Services Architecture

Like any application, the architecture of WS includes a set of roles and operations, which can be summarized in the following points as it was mentioned in [23]:

I. Publish

A WS publishes its capabilities together with its contact information (interface) in some service registry, such as UDDI, which acts like a broker for the registered WS.

II. Locate

The service broker is known to service requesters as well as to service providers; therefore, a service requester uses the broker to discover a service provider capable of providing the required service.

III. Bind

After locating the target service provider, the requester knows the information needed for establishing a relationship with the provider; and then, the requester is bound to the provider.

Both operations and roles involved in the architecture of WS can be described with a triangle that represented in figure 10, such that roles in the architecture are represented by the vertices of the triangle, and the operations are represented by the triangle's sides:

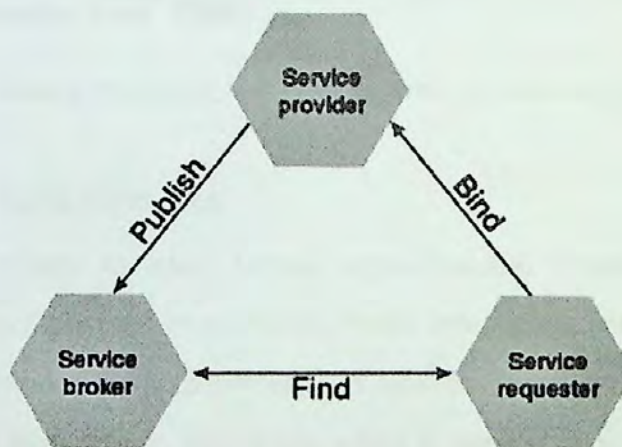


Figure 10 Web Services Triangle: Roles and Operations [23]

In order to make a service available to customers, the service provider describes that service by an interface using *Web Service Description Language (WSDL)*; and then, the interface is registered in a *Universal Description, Discovery, and Integration (UDDI)* registry. A Client queries the UDDI registry to find the services that match its requirements, and for each service that made a match, an access point to that service is obtained from the registry. The service description is used to create a proxy for the service, through which, the client can communicate with the service directly. SOAP, WSDL and UDDI are detailed in appendix B. Figure 11 summarizes the process of creating, publishing, and accessing WS over the Internet.

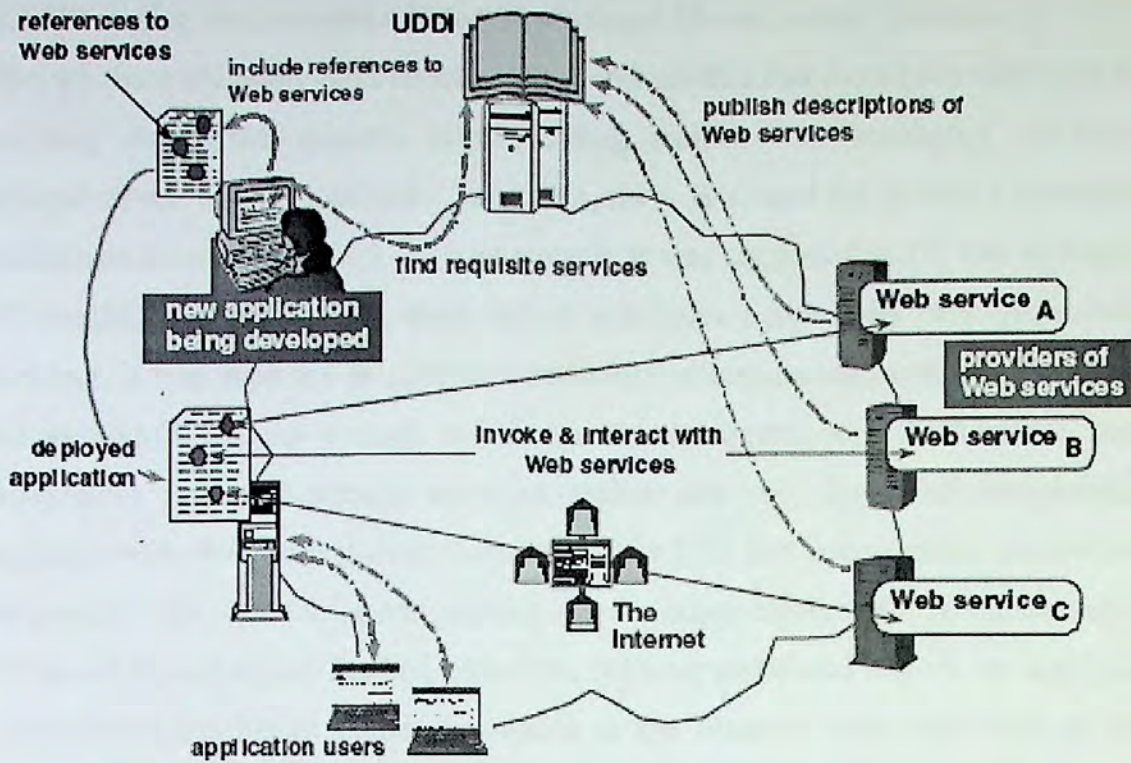


Figure 11 Creating, Publishing, and Invoking Web Services over the Internet [5]

3.3 Managing Web Services

Because of their ability to span across organizational boundaries and corporate firewalls [11], WS are adopted in providing better integration between applications, as well as dynamic services [24]; therefore, WS will be considered a major part of any organization in the near future. However, when it comes to WS-based applications, the environments, in which, those applications are running are characterized with uncertainty as any application of such applications is a network of services from different domains [25]; therefore, a change in the behavior of a single WS might have the effect of bringing the whole services network down [26]. With this increasing use and dependency on WS to enable enterprises to be adaptive [27], assuring reliability has become a must for smooth and profitable operations of WS. As a result, WS management will become a critical requirement as soon as the use of WS moves beyond simple integration to deploying applications that rely on the interaction of multiple WS [14]. After discussing the fundamentals of WS in the previous part, and highlighting the significance of WS to modern CS in the beginning of this part, the rest of this part will discuss the issue of managing WS.

3.3.1 Managing Web Services Using Traditional Management Systems

Although there are many applications, tools, and utilities that have been emerging for managing distributed systems, corresponding facilities for managing WS-based applications are not yet available. Therefore, there is a need for having a consistent architecture for managing WS [28]; as a result, it was suggested in [7] that managing WS should be based on the same set of principles adopted by WS themselves. Moreover, it was reported in [24] that traditional enterprise management standards, such as SNMP, are not enough to manage the dynamism of WS; therefore, such management standards should adapt to realize the new levels of complexities associated with WS. In addition, it was stated in [13] that management capabilities that provide only monitoring and alerting are no longer enough, but rather managing WS should be active; and further, proactive. Because traditional MS are not equipped to consider a significant parameter, which is the business side, such MS do not provide effective solutions in the context of managing WS [11]; consequently, business management should be considered as a new level of management by the MS that target WS [13]. Managing business is the process of managing all the BO at agreed levels of service; and to achieve this, all conducted transactions should be tracked from the beginning to the end [24].

Therefore, there have been some motivations towards a new management paradigm for managing WS that is generic, open, and extensible; and therefore, XML can be a corner stone for such paradigm [14]. For example, [14] suggested that SNMP can be evolved for the purpose of managing WS, such that *Management Information Bases (MIBs)*s are translated into XML elements and SNMP's operations, such as *set*, *get*, *trap*, etc, are also translated into corresponding WS operations.

3.3.2 Tasks Involved in Managing Web Services

Tasks that are involved in the process of managing WS can be summarized as follows:

I. Availability Monitoring

The availability of each WS should be checked periodically; this can be achieved by several approaches, such as the *ping-based approach*. In that approach, an echo request is sent to the node hosting the WS to check whether the node is alive; then, a valid SOAP message is sent to some operation in the WS being tested; the response is

validated by checking the existence of some particular values in the response; finally, an invalid SOAP message is sent to the same operation and the invalidity of the response is checked [13].

II. Performance Monitoring

The operations performed by each WS should be monitored in terms of some metrics, such as response times (e.g. average, minimum and maximum) per operation, number of pending requests, number of errors, fault rate per operation, etc [13].

III. Message Logging

Request and response messages for each WS should be logged in order to help the process of analyzing errors to find a feasible solution [13]. Log messages can be further converted into XML format in order to provide interoperability [21].

IV. Service Level Agreements (SLA) Managing

Based on their SLAs with the providers of WS, clients should be receiving the expected level of service. An SLA is defined out of a set service level objects that capture different information about the WS' health. In order to detect violation, the values carried by service level objects are matched against the threshold values stated in the SLA [13].

V. Providing Redundancy

In order to maximize the availability, multiple instances of the same WS may be deployed. In order to provide hot swapping among the similar instances, the following capabilities should be considered: (1) failure should be determined on both transport and messaging levels, (2) re-routing requests to other WS that either act as a backup to the failed one or provide the same functionality, (3) providing some conversion in messages according to the alternate WS and (4) logging the details of each executed failover [13].

VI. Versioning and Updating

A new version of some WS should be seamlessly introduced without affecting clients or services. However, replacing older version with the new one of a WS is not always necessary, but sometimes the two versions should exist to satisfy different needs [13].

VII. Synchronization

The MO should be synchronized between the MS from one side and the WS from the other side. Event-based triggering can be used as an approach for achieving synchronization. Messages from WS that reflect some events should be gathered,

filtered and delivered to proper components in the MS. Delivering messages is achieved through notification when some events occur; this event notification enables management to be provided at runtime [11].

3.3.3 Web Services Management Framework

WS that implement some MC in the form of MI in order to enable external MS to manage those WS are called a *Manageable Web Services (ManageableWS)* (the details of MI in the context of WS are discussed in section 3.3.6). After describing an MI with the aid of WSDL, it is published into an UDDI registry; and once the MI has been published, an MS can find the WSDL document describing the MI and begins to manage the ManageableWS. If a WS only exposes *Business Capabilities (BC)* through its BI, it is cannot be managed by external MS.

Therefore, a framework of managing WS is composed of two major components: (1) ManageableWS, which can be considered from the viewpoint of MS as an MR that was discussed in the previous chapter and (2) management application (MS) that communicates with the ManageableWS through the MI of the latter for the purpose of management. This generic framework of WS management can be illustrated in figure 12.

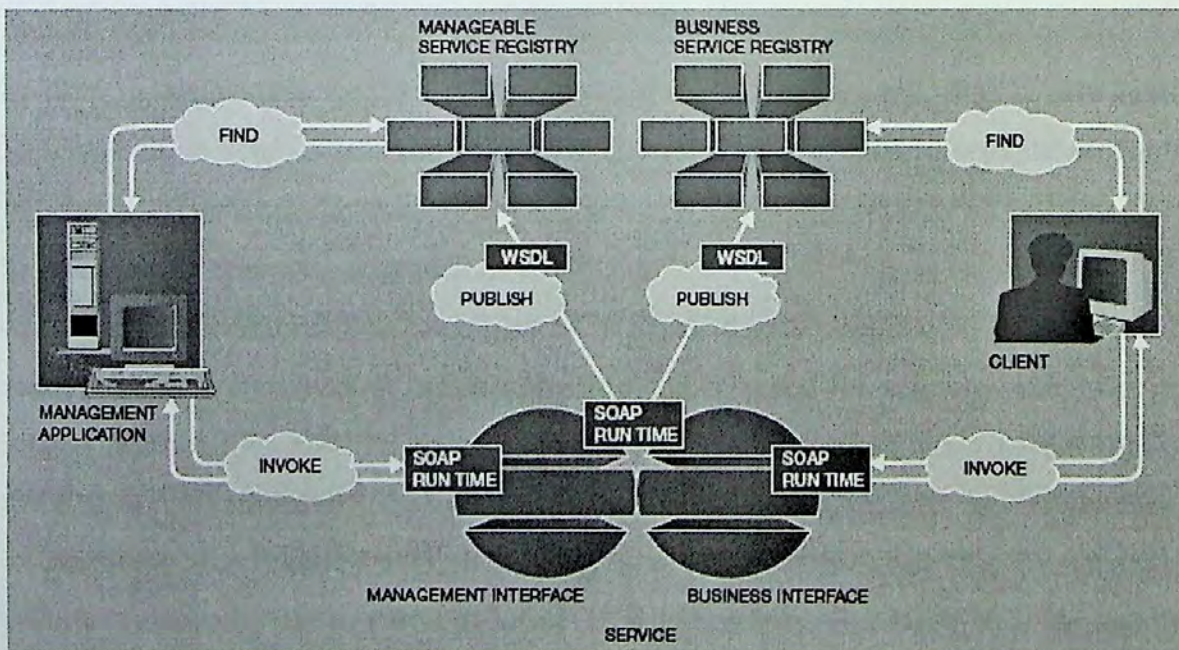


Figure 12 Web Services Management General Framework [7]

3.3.4 Extending Web Services Architecture to Support Management

In order to enable WS to be managed by MS, the WS architecture was revisited by [12] as shown in figure 13.

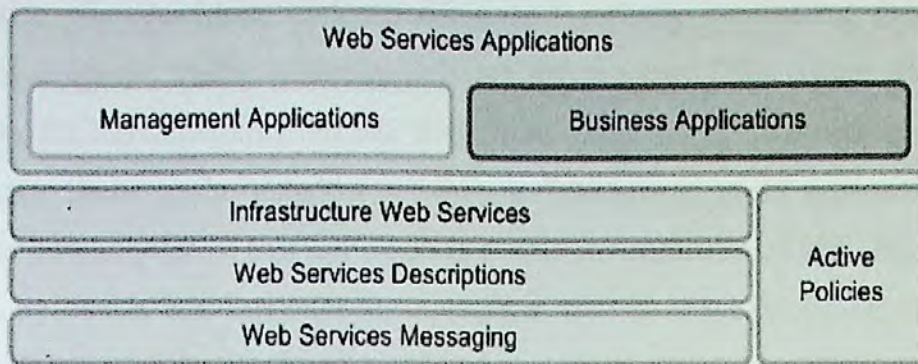


Figure 13 Revisited Web Services Stack

As seen in the figure above, management applications can be built as WS applications the same way as the business applications are built. Therefore, many benefits can be achieved, such as building management applications by using the existing standards and specifications from WS platform that are being developed by major vendors, such as W3C. For examples, HTTP can be used as a transport protocol for carrying management-related messages, XML can be used to represent management-related data, SOAP can be used to represent management-related messages, WS-Security can be used to secure management-related messages being exchanged, WSDL and XML Schema can be used to describe MI, WS-Addressing can be used to provide access to MI and *Web Services Business Process Execution Language (WS-BPEL)* can be used to orchestrate management processes (MO).

Moreover, infrastructural technologies of WS can be also applied in the context of management. For examples, *WS-Notification* can be used for sending management-related notifications and *WS-ResourceProperties* can be used to describe the MC of the WS. As policies are used to govern the execution of BO, policies can be also used to guide the execution of MO. *WS-Policy* is a policy framework that encapsulates security standards, such as *WS-SecurityPolicy*, which provides a policy grammar for security policies. However, more advanced management scenarios may require additional standards to be developed, such as policy grammars that cover different management domains, such as performance, availability, configuration, etc.

3.3.5 Web Service Management Approaches

A typical approach for effectively managing WS should include some capabilities, such as knowing the relationship between various services, understanding service performance characteristics, understanding, and actively controlling the flow and content of message traffic across a network of services [29]. There are multiple approaches for implementing WS MS that cover those management issues; however, an important issue to be considered when a WS management approach is being selected is how disruptive that approach is to the environment of WS [30]. Based on the level, by which, MS interrupt the operation of WS, WS MS can be divided into two categories: (1) *invasive* (embedded) and (2) *non-invasive* (decoupled). The following two subsections discuss the details of the two approaches:

I.Embedded Management

In this approach, either the management logic is hard-coded in the WS, such as embedding some proprietary headers into the SOAP requests and responses or requiring the use of proprietary management libraries and controls at development time [13]. For example, the MS may contain either some runtime libraries that implement a SOAP processor or some standard libraries that wrap the SOAP processor for handling messages; then, ManageableWS should call these management libraries directly at the runtime.

In the context of embedded management, [7] provided a solution for managing WS by using SOAP as the SOAP runtime on the environment hosting a WS receives invocation requests and passes them to the WS implementation class. In other words, a SOAP processor that runs in the WS environment, such as the application server, is responsible for handling both received and sent data. Therefore, WS can be instrumented with the aid of SOAP processor; this can be implemented by using JMX as follows [30]: (1) when the SOAP processor is initialized, it either discovers or instantiates an MBeanServer, (2) within the MBeanServer, the SOAP processor locates or creates an MBean for itself as well as each WS under its management and (3) when a SOAP processor is called, it collects information, such as identification information (e.g. URI), availability of the WS, some metrics (e.g. average response time per method), etc. Therefore, SOAP processors can represent the control points that enable the instrumentation of management information from WS.

Relying on the WS runtime environment to provide the basic levels of instrumentation promotes management and simplifies the task of developing WS as there are no longer needs for implementing instrumentation in the WS. Although the SOAP approach can be used to capture custom-tailored management aspects in the WS [13], not all the requirements for managing WS can be captured by that approach. In order to highlight the deficiency of the SOAP-based management approach, the following justifications were introduced in [13]:

1. Development efforts would be concentrating on management more than the business value, and this may result in inefficiencies of the latter.
2. Even the smallest change in MO will require recoding, recompiling, and re-deploying of the WS; therefore, the process becomes extremely time-consuming.
3. Obviously, a management technique that requires coding can not be applied to WS that span multiple administrative domains.

II. Decoupled Management

In this approach, any ready-made WS can be managed without changing its implementation. Therefore, this approach provides flexibility to enable a WS to be managed dynamically [30]. In this approach, MO are provided through some kinds of middleware between the providers and consumers [13]. Therefore, such approach decouples the management from the exact logic of the WS. The following points summarize the achieved benefits from the decoupled management approach:

1. The efficiency of applications is improved by concentrating on the business value.
2. The management effort is independent of the development effort; thus, the overall time and cost of WS development are reduced.
3. New MO can be added at runtime without the need of recoding the whole service.

3.3.6 Manageability Interfaces in The Context of Web Services

The notion of MI that was discussed in section 2.2.2 of the previous chapter can be revisited by adopting some concepts from of WS. Interface descriptions enable WS to be discovered as well as used dynamically over the Internet; the interface of a WS is described as a port type in a WSDL file, and the target of searching a UDDI registry is

to find that WSDL file. The interface(s) of some WS capture(s) the BC (BO) offered by that WS; and therefore, such interfaces are called BI. Similarly, MC of some WS can be captured by MI to expose the MC that are supported by that WS. An MI may be made generic in order to provide a simple access to significant information that is related to management, such as identification, configuration, and metrics; however, customized MI can be also implemented to deal with specific management aspects. The listing below shows the MO that may be included in a generic MI as suggested by [7]:

```
public interface ManageableWS
{
    public String getWebServiceID();
    public String[ ][ ] getMetrics();
    public String[ ][ ] getConfiguration();
    public String getAdminInterface();
    public Boolean isAvailable();
}
```

Listing 1 Skeleton Manageability Operation of a Generic Manageability Interface [7]

As it was stated in [31], through MI, WS can be managed either *directly* as MS directly exercises MC offered by the WS or *indirectly* as MS subscribes in some third-party facility to receive information about the WS. Moreover, through MI, MS gain access to WS in order to control the underlying computing resources through a set of wrapping management technologies, such as SNMP and CIM. Therefore, computing resources can be managed regardless of the underlying accessing protocols or the implementation details. For example, an MS can retrieve the free disk space on some hard disk by simply sending a SOAP request over an HTTP transport to retrieve the required value that is maintained by some property described by the properties schema of the MI wrapping the hard disk resource. As a result, computing resources can be managed by remote MS over the Internet.

I.Manageability Interfaces Exposure Options

An MI can be created and exposed by multiple entities, where the entity that is responsible for creating and exposing the MI takes the role of MCP; as it was mentioned in [32], those entities can be the WS, an agent representing the WS or the hosting environment of the WS. These options are illustrated in figure 14:

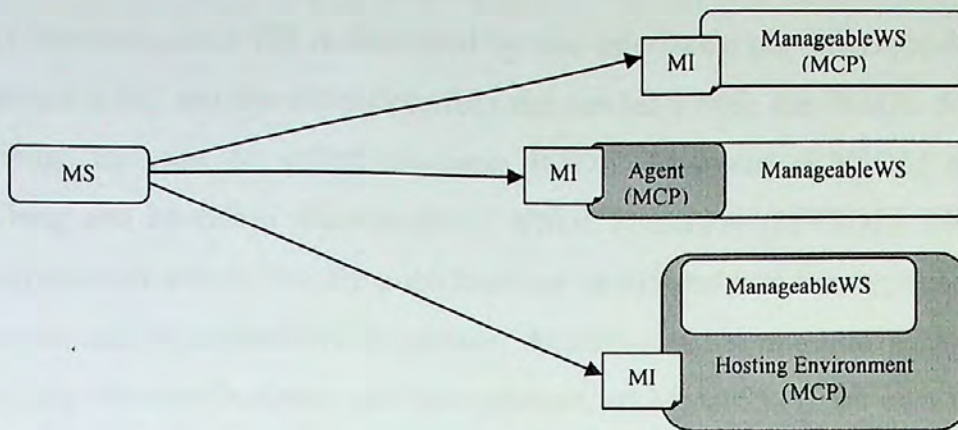


Figure 14 Different Options of Implementing the Role of Manageability Capabilities Provider [32]

II. Manageability Interfaces Implementation Options

There are two options for implementing MI, and the following two subsections discuss those two options as argued by [31]:

1. Consolidated Manageability Interfaces

In this approach, MC and BC are combined together by one consolidated interface, and one WSDL document is used to describe that interface. After publishing the WSDL, both clients and MS can discover the WS to either consume one or more of its BC or MCs. Since the same WS endpoint receives both business and management requests; and based on the content of the received requests, the implementation of the WS takes the proper actions; this approach of managing WS can be referred to as *in-band management*. This approach has the advantage of enabling MS to have some perception on BC as well as clients to be aware of MC, which might be required in some scenarios. However, an important issue is how to customize the visibility so that a client should not be able to retrieve any information related to management. The following figure illustrates the concept of in-band management:

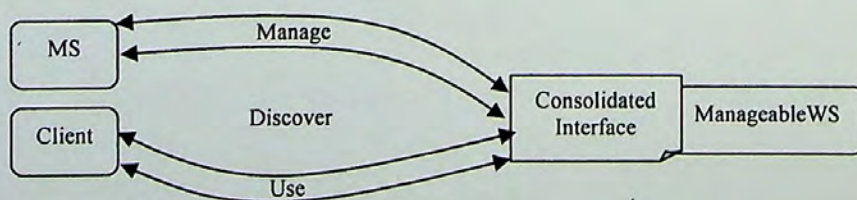


Figure 15 In-Band Manageability

2. Separated Manageability Interfaces

In this approach, each WS is described by two interfaces; one interface describes the service's BC and the other describes the service's MC; the WSDL document describing BC can be called *Business WSDL document (BWSDL)* and that describing MC can be called *Manageability WSDL document (MWSDL)*. Moreover, the registries, in which, WS are published can be divided into two types: *Business Registries* and *Manageability Registries*. Additionally, in order to achieve total decoupling between business and management, an MI can have its own port that differs from the actual service port in the service's WSDL document, such that all MO are performed through the MI port; therefore, this approach of managing WS can be referred to as *out-band management*. This approach has the advantages of (1) avoiding the confusion that results from having interfaces describing both BO and MO and (2) MO can be published separately; and therefore, MS can discover WS in terms of the exposed MO. The following figure illustrates the concept of out-band management:

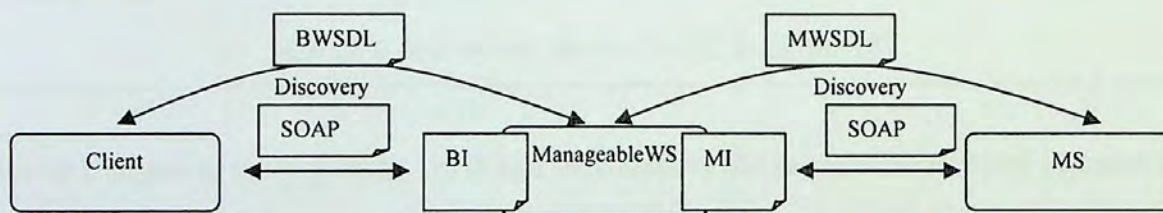


Figure 16 Out-of-band Manageability

3.3.7 Web Services Managing Example: AccountInfo Web Service Example

In this section, the StockQuote WS example that was introduced in [7] is discussed to illustrate the basic principles of WS management by discussing the WSDL documents that capture those principles. First, listing 2 shows portions of the service's description document.

```

<wsdl:message name="GetAccountInput">
  <part name="symbol" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="GetAccountOutput">
  <part name="value" type="xsd:float"/>
</wsdl:message>
<wsdl:portType name="AccountInfoInterface">
  <wsdl:operation name="GetAccountInfo">
    <wsdl:input message="tns:GetAccountInput" />
    <wsdl:output message="tns:GetAccountInfo" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="AccountInfoBinding" type="tns:AccountInfoInterface">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="GetAccountInfo"/>
  <soap:operation soapAction="urn:AccountInfoInterface#GetAccountInfo"/>
  <wsdl:input>
    <soap:body use="encoded" namespace="urn:AccountInfoService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="encoded" namespace="urn:AccountInfoService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </wsdl:output>
</wsdl:binding>
<wsdl:service name="AccountInfoService">
  <wsdl:port name="AccountInfoServicePort"
  binding="sqi:AccountInfoInterface">
    soap:address location="urn"/>
  </wsdl:port>
</wsdl:service>

```

Listing 2 AccountInfo Service WSDL Document [7]

Second, separate management ports can be added to the service description in order to include either generic or customized MI. Therefore, the portion in bold in the listing above can be modified by adding the description of MI as it is shown in listing 3.

```

<wsdl:service name="AccountInfoService">
  <wsdl:port name="AccountInfoServicePort"
  binding="sqi:AccountInfoInterface">
    soap:address location="urn"/>
  </wsdl:port>
  <wsdl:port name="AccountInfoManagementPort"
  binding="sqi:AccountInfoManagementInterface">
    soap:address location="urn"/>
  </wsdl:port>
  <wsdl:port name="AccountInfoGenericManagementPort"
  binding="sqi:AccountInfoGenericManagementInterface">
    soap:address location="urn"/>
  </wsdl:port>
</wsdl:service>

```

Listing 3 Adding Management [7]

As shown in the listing above, the modification involves adding both a customized and a generic MI. Both listing 4 and listing 5 show the description of each MI respectively.

```
<wsdl:message name="startInput">
  <part name="action" type="xsd:string" />
</wsdl:message>
<wsdl:message name="startOutput">
  <part name="newState" type="xsd:string" />
</wsdl:message>
<wsdl:message name="stopInput">
  <part name="action" type="xsd:string" />
</wsdl:message>
<wsdl:message name="stopOutput">
  <part name="newState" type="xsd:string" />
</wsdl:message>
<wsdl:message name="changeRateInput">
  <part name="value" type="xsd:float" />
</wsdl:message>
<wsdl:portType name="AccountInfoManagementInterface">
  <wsdl:operation name="start">
    <wsdl:input message="tns:startInput" />
    <wsdl:output message="tns:startOutput" />
  </wsdl:operation>
  <wsdl:operation name="stop">
    <wsdl:input message="tns:stopInput" />
    <wsdl:output message="tns:stopOutput" />
  </wsdl:operation>
  <wsdl:operation name="changeRate">
    <wsdl:input message="tns:changeRateInput" />
  </wsdl:operation>
</wsdl:portType>
```

Listing 4 Customized MI [7]

```

<wsdl:message name="GetServiceIdOutput">
  <part name="value" type="xsd:float"/>
</wsdl:message>
<wsdl:message name="GetMetricsOutput">
  <part name="name" type="xsd:string"/>
  <part name="type" type="xsd:string"/>
  <part name="value" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="GetConfigurationOutput">
  <part name="name" type="xsd:string"/>
  <part name="type" type="xsd:string"/>
  <part name="value" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="GetAdminInterfaceOutput">
  <part name="AdminInterfaceWSDLrun" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="GetAvailabilityOutput">
  <part name="value" type="xsd:integer"/>
</wsdl:message>
<wsdl:portType name="GenericManagementInterface">
  <wsdl:operation name="GetServiceId">
    <wsdl:output message="tns:GetServiceIdOutput"/>
  </wsdl:operation>
  <wsdl:operation name="GetMetrics">
    <wsdl:output message="tns:GetMetricsOutput"/>
  </wsdl:operation>
  <wsdl:operation name="GetConfiguration">
    <wsdl:output message="tns:GetConfigurationOutput"/>
  </wsdl:operation>
  <wsdl:operation name="GetAdminInterface">
    <wsdl:output message="tns:GetAdminInterfaceOutput"/>
  </wsdl:operation>
  <wsdl:operation name="IsAvailable">
    <wsdl:output message="tns:GetAvailabilityOutput"/>
  </wsdl:operation>
</wsdl:portType>

```

Listing 5 Generic MI [7]

3.3.8 Events-Based Management

Some of the events that take place in an environment of WS, such as, metric value change, failed method invocation, etc, might be significant enough to be reported to the MS. In order to enable the collection of events, a third-party WS can be used to capture events as they occur; this WS can be called *Event-Collecting Web Service (ECWS)* [7]; in addition, an ECWS should expose a simple interface that is described by a WSDL document, such that calling that interface causes the reception of events by the interested entities. A ManageableWS can be either statically bound to an ECWS during the development time, or it can discover it at runtime. Moreover, an MS should register for events of significance so that the ECWS forwards those events to it.

The interface of an ECWS should include at least one method that allows the interface to deliver events. The arguments for such method should identify the WS, specify the source of the event, indicate the severity of the event, and provide a description of the event. Moreover, ManageableWS should provide an operation for receiving events (e.g. *receiveEvent*) from the ECWS. The interface of a typical ECWS might look like the following:

```
public interface EventCollector
{
    public void deliverEvent (String id, String source, String severity, String text);
}
```

Listing 6 Sample ECWS Interface [7]

Moreover, events delivering can be automated by including another operation in the ECWS that could be used by ManageableWS to instruct the ECWS to invoke the *receiveEvent* operation on the ManageableWS based on some specific conditions or the contents of the event.

3.3.9 Web Services Management Patterns

A WS management pattern is an architecture that defines both the components making up the MS and the relationships between WS and MS; the pattern should also specify a set of MO and determines the flow of information. The following subsections review some of the WS management patterns that were introduced by [7]:

I.Events Generator Pattern

By using ECWS, MablWS are completely decoupled from the details of the MS. Instead of implementing the MI by the ManageableWS to provide MO, the interface provides a set of notifying messages that represent significant events.

II.Noninterruptable Pattern

A ManageableWS provides a management object that contains the identity and metric information as well as the current configuration and make this object available to the MS; moreover, events are delivered to MS by the ManageableWS. MS are notified about changes occurring to management objects, such that the ManageableWS either resends updated versions of the management objects or emits specific events for this purpose. This pattern is preferred when a particular MS requires receiving

management data formatted with some specific format. The interface of ManageableWS in this pattern provides messages for both events notifications as well as management objects updating notifications. Generally, management objects are made generic; however, a custom object can be created with the aid of some middleware that provides marshalling/unmarshalling facilities. An MBean is an example of the management object in a JMX environment.

III. Operational

In this pattern, MS can retrieve management data from ManageableWS as well as apply some control actions to the ManageableWS by invoking their MO. ManageableWS must implement MO to be called by the MS and should also provide some facilities for listening and responding to management requests. Because MO are implemented by the ManageableWS, both generic and customized MI are supported. The interface of ManageableWS in this pattern must expose a set of notifications, queries, and operations.

3.3.10 Web Services Management Challenges

While WS provide many integration benefits, they also bring many challenges for the context of managing WS [24]. The following points shed light on the challenges that might hinder the building of WS MS:

- I. Because the stack of WS is still in the phase of evolution to meet emerging needs, such as security, *business-to-business (B2B)* interactions, etc, managing the operations described by the stack, which have many variations by vendors, is a difficult task [24].
- II. WS Management is challenged by the complexity characterizing WS environments [25].
- III. Because of immaturity of the WS technologies, full WS lifecycle management cannot be achieved easily [24].
- IV. Because WS span multiple environments, managing WS cannot be based on using particular management technology. For example, JMX is suitable for WS implemented in Java; however, it is not for those implemented using .Net [7].
- V. A WS should be able to find MS at runtime the same way it discovers other WS and interacts with them [7].

3.4 Using Web Services in Management

After discussing the fundamentals of managing WS in the last part and highlighting the challenges accompanying that, this part draws attention to how WS technologies can be applied in the context of management; or in other words, how WS can manage CS.

3.4.1 Significance of Web Services to the Context of Management

Recently, the adoption of SOA using WS technologies has been demonstrating success by providing scalable solutions to enterprise-wide integration problems. As discussed earlier in this chapter, the stack of WS provides a set of open standards that are used in integrating heterogeneous applications that either are implemented using different platforms or run in different environments. Because building effective MS is challenged by integration challenges, WS can be adopted in the context of management to be used as the integration standard between different management technologies to help addressing that challenge of heterogeneity. Therefore, by using WS in building MS (WS-Based MS), a new layer is added to the management stack rather than replacing the whole stack to evolve MS that are able to manage heterogeneous CS, which has the impact of reducing the costs of both development and maintenance as well as permitting legacy MS to easily evolve towards the paradigm of WS-Based MS.

3.4.2 The Emergence of Web Services-Based Management Systems

Based on the discussion of the last section, MS can be built as WS in the same way regular WS are built; this approach enables MS to be accessible over the Internet by WS. In this document, WS-Based MS are referred to as *Managing Web Services (ManagingWS)*, which can be defined as MS that can provide management capabilities in the form of services that can be found and invoked over the Internet. As a result, WS can be classified into two major classes in terms of the type of provided services: *Business Web Services (BWS)* and *ManagingWS*. Beside the advantage of having both BWS and *ManagingWS* communicating with each other over the Internet, there is another advantage, which is that like BWS, management services can be reused by multiple business processes (e.g. BO) in different environments, which contributes to reducing the overall development and maintenance cost.

Using WS in the context of management introduced another challenge in terms of maintaining the relationship between both BO and MO. In other words, how the activities that are associated with performing BO are coordinated with those activities involved in the execution of MO. As it was mentioned in before, WS architecture provides the WS-BPEL standard, which can be used for orchestrating invocations of WS in some transaction. Therefore, WS-BPEL can be used to simplify the processes of invoking MI in a way that does not contradict with invoking BI.

As it was highlighted in [12], the concept of ManagingWS may be useful for different parties that are involved in WS management frameworks as follows:

- I. Providers of computing resources can provide resources with standardized MI in order to allow a variety of ManagingWS to manage their resources; and therefore, the cost of developing manageability instrumentation will be reduced while the opportunity of having computing resources properly managed is maximized.
- II. Infrastructural components involved in the implementation of MS, such as messaging, events and others are encapsulated by MI; therefore, the cost of adapting MS as requirements change is reduced.
- III. Having MO exposed as services permits scalability. Moreover, the development of ManagingWS will focus on value-added management services instead of investing more in developing the basic communication between MS and MR.
- IV. Because many of the modern CS already leverage or plan to leverage WS to expose BO, the same technology will be used for exposing MO; therefore, MO will be consistently developed with the development of BO.

3.4.3 Composition of Web Services-Based Management Systems

The anatomy of WS-Based MS was discussed in [12], in which, it was concluded that such MS are composed of two major types of WS, which are *infrastructure services* and *managing services* (e.g. managers).

I. Management Infrastructure Web Services

This type of WS provides interfaces for common MO such as metering, events-collecting, monitoring, policy-management, system scanning, etc. There are different approaches for implementing infrastructure services; for example, they can be implemented as distributed agents in order to enhance efficiency and scalability.

II. Managing Web Services

Managing WS, or managers, make use of management infrastructure services to achieve the required management objectives. They control the behavior of MR, such as managing availability, performance, configuration, problem determination, etc; thus, managers should provide a wide range of simple MO, such as monitoring, and sophisticated MO, such as autonomic tuning. Managers can use WS-BPEL to orchestrate multiple infrastructure management services in order to execute some MO. Moreover, some managers may use other managers to perform specific management tasks; therefore, managers should expose standardized interfaces to be used by other managers.

Figure 17 abstracts the composition of WS-Based MS in terms of what has been discussed in this section:

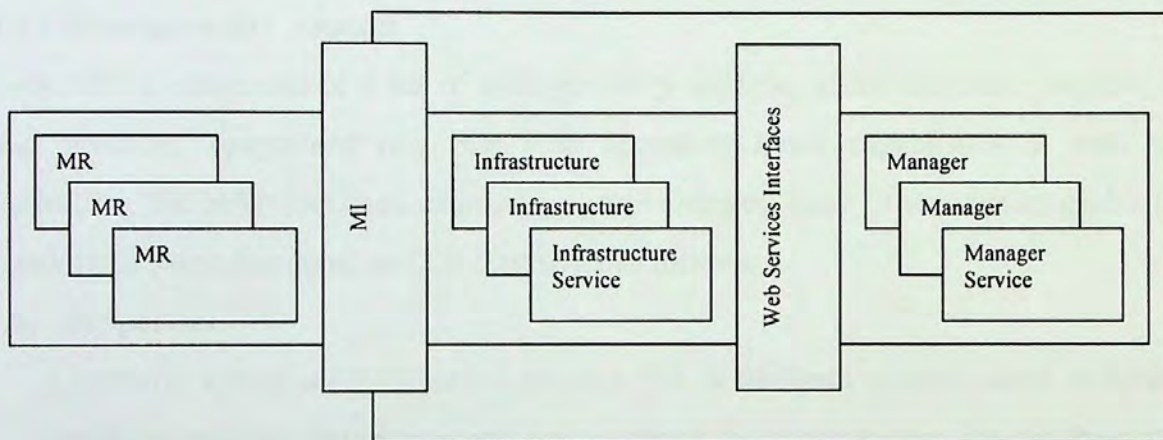


Figure 17 Composition of Web Services-Based Management Systems

3.5 Web Services Manageability

The difference between management and manageability was previously identified in the first chapter. After discussing different aspects related to management in the context of WS in the last two parts of this chapter, this part sheds light on manageability in the context of WS.

3.5.1 Manageability Model

A typical manageability model should define a base set of MC and define how those capabilities are *exposed*, *discovered*, and *accessed*. Moreover, a set of management aspects are defined for each MC as discussed in the following subsection. In addition,

scenarios, such as *Accounting, Usage Auditing and Tracking, Performance Monitoring, Availability, Configuration, Control, Security Auditing and Administration, and Service-level Agreements* should be supported. The figure below shows a typical manageability model as it was introduced by [33]

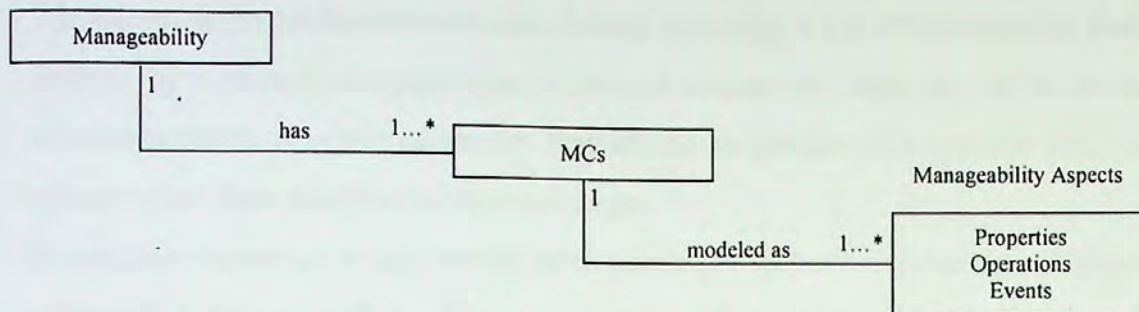


Figure 18 Manageability Model [33]

3.5.2 Manageability Aspects

Each MC is composed of a set of manageability aspects, which includes: *properties* (e.g. metrics), *operations* (e.g. for both accessing those capabilities as well as controlling the behaviour) and *events* (e.g. state change). Each of those manageability capabilities were discussed in [33] discussed as follows:

I. Properties

A property advertises information about a WS in the form of parameters of either simple or complex data types, and the values of those parameters specify the state of WS at some time; XML is used to express properties' parameters. The properties are either *modifiable* or *static*, such that the values of modifiable properties can be dynamically changed by ManagingWS or as a side-effect of the execution of some operations, but static properties cannot be changed; as an example of a static property is the identification property. One property that lists all the available properties can be added to the MI of ManageableWS to enable ManagingWS to introspect what properties are supported by an MI.

II. Operations

Operations are used to retrieve properties information or to manage WS in order to change their behaviors; operations are specified by defining an exchange of messages. Properties are accessed and manipulated through MI operations, such that either generic get or strongly-typed get operations can be used to retrieve the

values of properties; similarly, values of modifiable properties can be set by ManagingWS by using either generic or strongly-typed set operations.

III. Events

Events are the indicators that the behaviour of some WS has changed in some way as events are emitted to advertise change in the values of specific properties. Therefore, an event describes some change by using a set of information that is defined by a named, complex type. Although events are important in the context of management, it is not necessary that all the properties of a specific MC emit events when their modifiable values change.

Events are contained within notification messages that are delivered to interested parties (e.g. ManagingWS). The content of notification messages is based on the event being described; however, all events share a set of common information, such as time of the event, source of event, reporter of the event and situation type; therefore, a common event schema, such as *WSDM Event Format (WEF)*, should be used to ensure interoperability among different components that are responsible for generating, delivering and understanding events. Moreover, a standardized notification mechanism for subscription and publication of an event, such as WS-Notification should be used by both ManageableWS and ManagingWS.

3.5.3 Manageability Capabilities

Manageability can be supported by a WS through a set of MC, such that each MC has its own semantics. An MC can be captured by one or more MI, and the semantics of some MC describe the management-related messages that can be exchanged between a ManageableWS and a ManagingWS. MC can be considered as a layer in the architecture of a ManageableWS; this layer defines a composable set of WSDL interfaces, schemas, and metadata that describe behaviors, which can be advertised by ManageableWS to enable ManagingWS to access its manageability information. The set of MC that is implemented by a ManageableWS is based on the nature of the resources being managed; however, there is basic set of MC that is significant for the management of any ManageableWS, such as metrics and operational status [12], and if the ManageableWS represents some virtual resource, additional MC can be exposed, such as lifecycle.

The appropriate information and details of a ManageableWS that are related to management, such as properties, events, operations, and metadata can be captured by some *resource model*; it is not necessary that all MC have the same resource model; for example, there might be a resource model that describes only properties. Although there are some traditional resource models, such as CIM and SNMP; those models do not provide all the required details; for example, SNMP provides properties and events, but not operations. Therefore, WSDM provides a common semantic and definition for MC by representing MC by using WS [12].

3.5.4 Web Services Manageability Framework

Based on the last discussion of manageability model, manageability aspects, and MC, the following the meta-model was introduced in [33] to describe a typical manageability framework for WS:

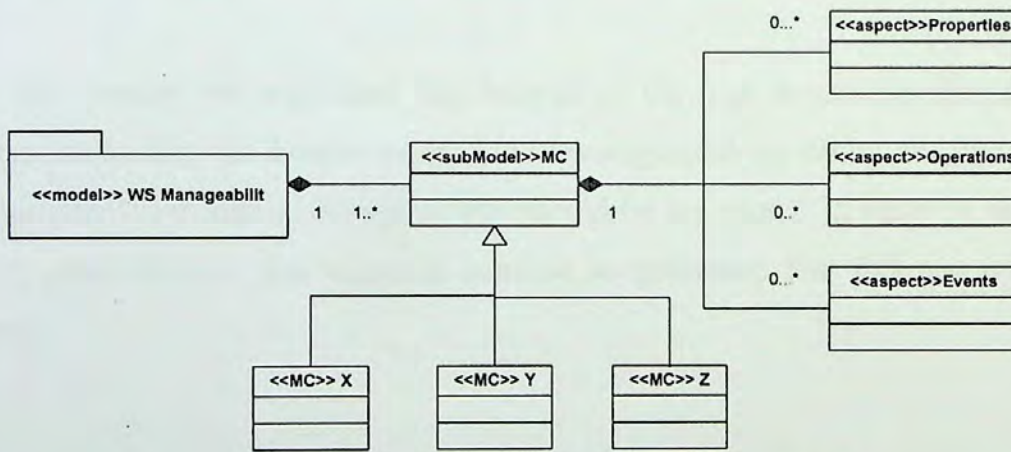


Figure 19 Web Services Manageability meta-model [33]

3.6 Chapter Summary

Traditional computing paradigm, such as Object-Oriented and Component-Based paradigms have been used for many years in building many business CS applications. However, with the emerging of new business requirements that are characterized with highly dynamic relationships, such paradigms did not prove the required efficiency in fulfilling those requirements because of the inherited tight-coupling and static binding in the applications built using those paradigms. As a result, SOA was emerged as a natural extension to legacy CS architecting paradigms. WS can be considered the most commonly realized application of SOA. With the emerging of WS, a new class

of applications appeared to make use of the unique WS characteristic that is platform-independency.

Like legacy CS, WS should be properly managed in order to achieve the expected benefits. The features of WS emphasize that traditional MS are not enough to manage WS; therefore the WSM paradigm was emerged with the target of building efficient MS to WS. The management requirements of a WS can be integrated with the business requirements; however, separating the business and management concerns of WS, and capturing each set of concerns by a separate interface would prove better efficiency and scalability. Moreover, MS can be implemented as WS in order to work around the heterogeneity challenge to facilitate the seamless communication between MS and WS. Therefore, by this point, it should be clear to the reader that the name of the chapter reflects the significance of WS to both modern business and management applications as both paradigms can be integrated with the aid of WS in a way that enables both building and operating of such applications to be conducted in the same way.

After this chapter, we concluded that because of the high dynamism characterizing WS environments, the human-intervention management model is not suitable for managing WS, but rather managing WS should be automated in order to catch and recover from failures that occur at runtime to guarantee that BO are performed properly.

Chapter 4: Related Work

Chapter 4: Related Work

4.1. Chapter Overview

In this chapter, some standards that are concerned with managing WS together to some of the work that is related to the context of our research will be reviewed:

4.2. Related Work

4.2.1 Management Standards

In order to deal with the challenges of managing WS, a new paradigm called *Web Services Management (WSM)* has emerged by highlighting the requirements of managing WS environments [26]. WSM is not only concerned with managing WS, but it is also concerned with providing management capabilities through WS [24]. Under the umbrella of WSM, a lot of efforts have been exerted in order to formalize the process of managing WS; these efforts included standardizing organizations, such as W3C and OASIS as well as reputable vendors, such as HP and IBM. Some of those efforts were fruitful enough and ended-up with some standards, such as WSLA - that is concerned with monitoring service-level-agreements [26]. This part focuses on discussing two important WS management standards, which are *Web Service Management Framework (WSMF)* [34] and *Web Services Distributed Management (WSDM)* [32], [35] and [36] and [17]. However, WSDM will be discussed in some more details as WSDM can be considered the origin of the framework that we propose in this thesis, which will be discussed in chapter five.

I. WSMF

HP launched the WSMF standard, which is later merged with the OASIS's WSDM [27]. WSMF is a logical architecture that is based on the concept of managed objects, where each managed object represents an MR that exposes a set of MI, through which; the underlying resource can be managed and controlled. The architecture unifies WS management by defining a set of functions, interactions, and formats that capture the basic MO that include service discovery, relationships, performance, and control [27]. Moreover, WSMF can be applied to resources that are not Web Services by wrapping such resources by WS that implement the WSMF port types [27]. This has the impact on evolving legacy computing resources to be managed the same way

as WS. Figure 20 shows how WSMF is used for managing both WS and non-WS resources:

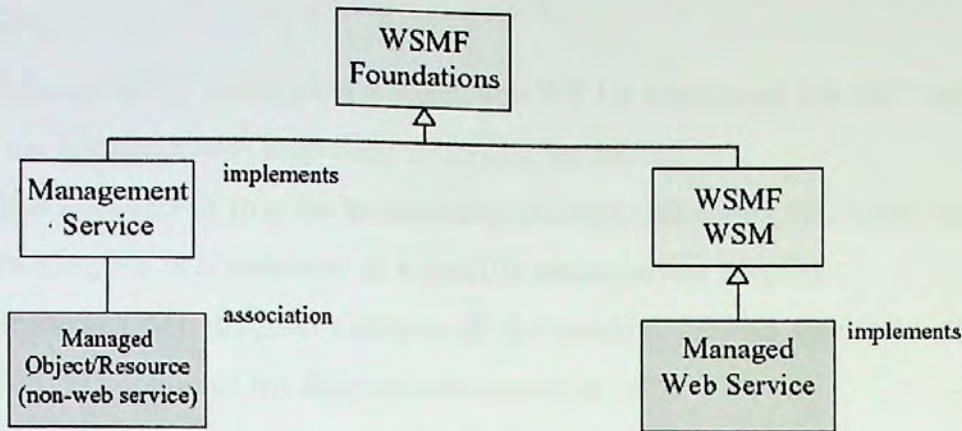


Figure 20 Web Services Management Framework Architecture [27]

II. WSDM

WSDM is a manageability framework for WS that was introduced by OASIS; it is divided into two major standards: *Management Using Web Services (MUWS)*, which is concerned with using WS in the context of management, and *Management of Web Services (MOWS)*, which focuses on the management of WS as software resources. Two versions of MUWS have been released [32], [35], and [36]. The following subsections discuss the fundamentals of WSDM; however, the specifications are detailed in the appendix:

1. General WSDM Architecture

From the viewpoint of [32], the architecture of WS manageability can be described as follows:

- a. A ManageableWS extends the concept of WS in order to support one or more MC by implementing one or more *Manageability Endpoint (MEP)*. One MEP may be used to provide access to multiple ManageableWS, and the same ManageableWS may be accessed via multiple MEPs.
- b. A MEP extends the concept of WS endpoint in order to provide access to its ManageableWS for the purpose of management by offering the MC supported by its ManageableWS.
- c. Like WS endpoints, A MEP is described by one port element in the WSDL document. One or more *Endpoint References (EPRs)* are used to reference

- each MEP, and each EPR should be resolved to the actual address that is used by ManagingWS to locate that MEP in order to access the ManageableWS.
- d. An MI extends the concept of WS interface and is realized by one or more MEPs.
- e. A ManagingWS manages a ManageableWS by exercising the MC supported by the ManageableWS through accessing the MEPs.
- f. A ManageableWS may be managed by multiple ManagingWS, such that each ManagingWS is specialized in a specific management domain.

The following UML diagram captures all the points discussed above; the diagram is a modified version of the diagram introduced in [32]:

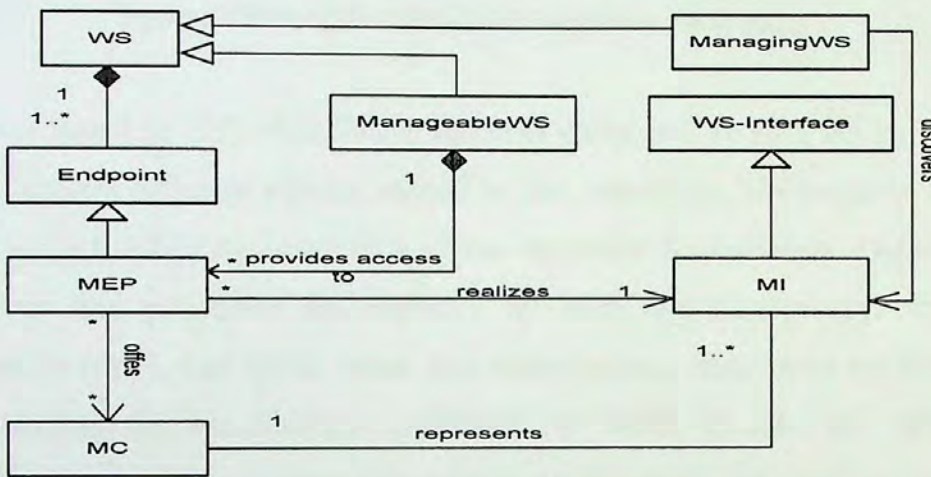


Figure 21 Web Services Distributed Management General Architecture

2. Common vs. Specific Manageability Capabilities

In order to implement the WSDM-MUWS specifications, a minimum set of MC must be implemented by either the ManageableWS or the MCP that expose the MC of the ManageableWS; this required set of MCs is defined by MUWS as *Common Manageability Capabilities (CMC)*. CMC include the ability to identify ManageableWS and to report and notify on the occurrence of some changes. Moreover, WS-Specific MC can be created by extending CMC; extended capabilities are called *Specific Manageability Capabilities (SMC)*. Extending CMC in order to create new SMC can be achieved by two different ways as it was mentioned by [31]: (1) Adding new aspects to existing MC through inheritance or (2) Creating new MC to capture new aspects that are not included by CMC, such as security aspects. SMC are based on the management model of the resource

being managed, the management needs, etc. Figure 22 illustrates how we can implement a specific manageability capability for a Printing WS.

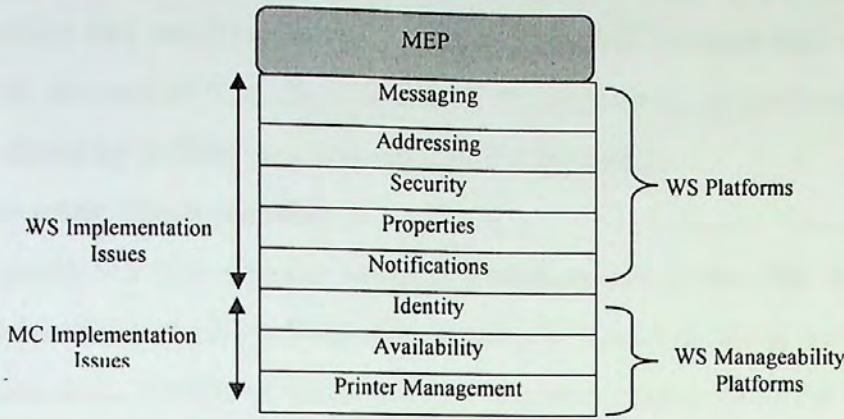


Figure 22 Printing Web Service Manageability Stack [32]

As it was stated in [35], each MC is formally expressed by an *UML class model*, which captures different aspects related to that capability. The name of the class model should reflect the semantics of the capability it represents. Data types of operations and properties are captured by some *information type class* that declares, to which data types, those data types belong. Data types are defined by the XMLSchema specification provided by W3C. Events are defined as properties, but with an `<<event>>` stereotype. Similarly, information type of an event is captured by a class that contains information element definitions. The idea of representing a certain MC (e.g. X-MC) that is specific to X-ManageableWS is illustrated by figure 23.

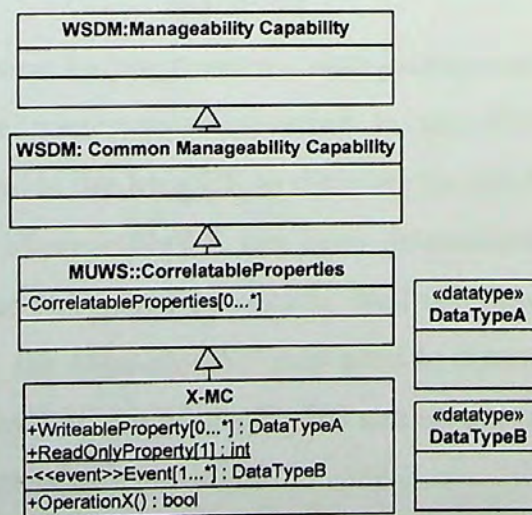


Figure 23 X Specific Manageability Capability UML Class Model

The multiplicity associated with each property determines which properties are mandatory and which are optional. Moreover, the UML constraints capture the metadata associated with each element in the model. Such constraints may define the *mutability* and *modifiability* of a certain property. For any MC, the data type classes are declared at first; then, elements of capabilities are defined. The details of MC defined by WSDM are discussed in the appendix.

3. Discovering Manageability

A ManageableWS can support MUWS specifications if the WS implements at least, the *Identity* capability. Since ManageableWS publish MI in the same way as BI are published, traditional discovery mechanisms employed in the WS platform (e.g. UDDI registry) can be used to discover ManageableWS [31]. Moreover, in order to discover new ManageableWS, a ManagingWS may follow the relationships of the already discovered ManageableWS with other WS [31]. The goal of discovering ManageableWS is to enable ManagingWS to obtain an EPR to the MEP of the ManageableWS.

A ManageableWS is discovered at first; and then, the MC associated with the WS are discovered; this can be achieved as follows [31]: (1) after discovering a WS, the discovering ManagingWS should analyze the WSDL document of the discovered WS in order to decide whether the discovered WS is a ManageableWS or not, (2) If the discovered WS was identified as manageable, the Managing WS should identify whether the in-band or the out-of-band management type is employed, (3) In the first type, the discovered endpoint will offer some standard manageability operations, such as `{urn:wsdm:webservice:endpoint:metrics:data,getRequestCountersIn}`. In the second type, some namespaces appearing in the WSDL document, such as *relationship*, will guide the MngWS to discover the MEP and (4) Once the MEP of the discovered ManageableWS has been determined, the ManagingWS can detect the associated MC either by parsing the MEP description or by asking the MEP; for example, the ManagingWS may need to determine if the MEP offers a specific metric; therefore, the ManagingWS can send a request to retrieve the list of the available metric QNames.

4. Manageability lifecycle

The management relationship between a ManagingWS and a Manageable can be illustrated with the following lifecycle that was introduced by [35]:

- a. The ManagingWS discovers the WSDL document of the ManageableWS.
- b. Obtaining an EPR to the MEP when the ManagingWS discovers the ManageableWS.
- c. In the case of In-Band Management, the ManagingWS should check whether the MUWS *Identity* capability is supported; this check can be conducted by using one of the following methods:
 1. Obtaining the WSDL document and looking in the first level children of the RPD root for the *ResourceId* element.
 2. Requesting the value of the *ManageabilityCapability* property and look for the URI that identifies the *Identity* capability.
- d. MC that are supported by the ManageableWS can be determined by investigating, the values of the *ManageabilityCapability* property.
- e. Based on the MC that will be used by the ManagingWS, and by using the obtained ERP, the ManagingWS builds the messages that will be targeted at the endpoint.
- f. Building management messages is based on understanding the operations and binding that are included in the WSDL document describing the MC.
- g. The ManagingWS exchanges messages with the ManageableWS in order to exert control on the ManageableWS by interacting with the located MEP.

The set of MC that is defined by WSDM is listed in Appendix B. Moreover, based on WSDM, IBM introduced the Autonomic Computing Touchpoint (ACTP) specifications [6], which are discussed in brief in Appendix B.

4.2.2 Autonomic Web Processes

Autonomic Web Processes (AWP) [27] proposed an approach for implementing the self-healing, self-configuration, and self-optimization AC capabilities in the level of Web processes. In the context of this paper, AWP are WS-based processes that support the properties of AC of being self-managed. From the viewpoint of the authors, AWP can be considered as the evolution of AC from individual computing resources to the business processes that regulate the performing of different business

activities. As a contribution, the paper introduced a framework for AWP, which is built upon other researches, such as semantic Web processes, workflows, and AC. The proposed framework consists of four components: autonomic execution engine and three AM that are responsible for supporting the self-configuring, self-healing, and self-optimizing AC functionalities. The following figure illustrates the abstract composition of the proposed framework:

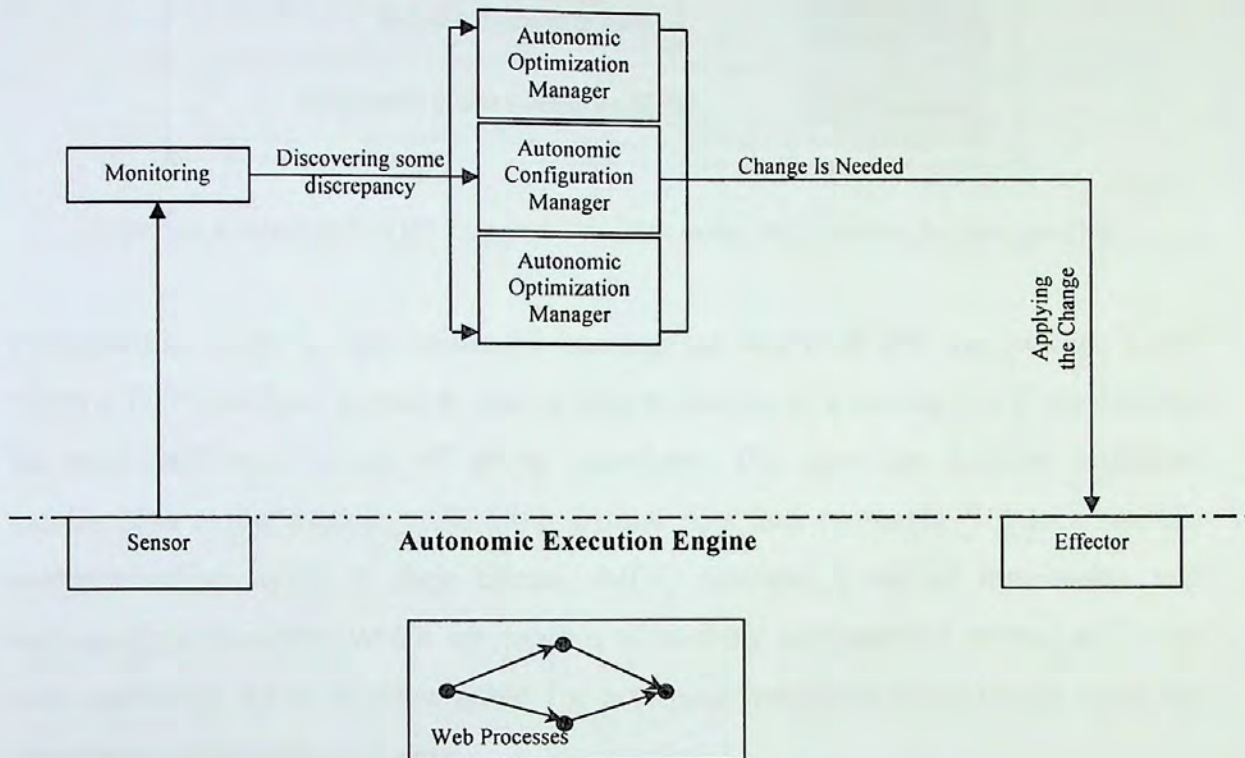


Figure 24 Proposed Autonomic Web Processes Framework

4.2.3 Adding High Availability and Autonomic Behavior to Web Services

The focus of this research [38] is on the reliability issues that are associated with highly-available applications, which are the applications that need to remain optional and rapidly responsive even in the case of failures; such applications are called *mission-critical applications*. Although the WS architecture includes a reliability specification and an underlying reliable message passing technique, the details of the these aspects of the architecture are still immature. Therefore, this paper proposed some extensions to WS architecture; those extensions include health-checking services, integrating some autonomic characteristics into applications, such as self-monitoring, self-analysis and self-recovery and data-mining tools. The following

figure introduces the extensions that are proposed by this paper to be added to the WS architecture:

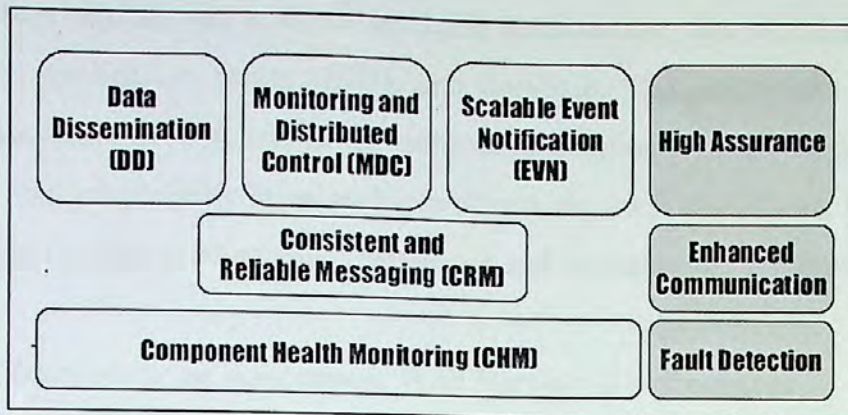


Figure 25 Components of the Proposed Extension to the Web Services Architecture [38]

For example, CHM is responsible for tracking the health of WS components; CRM offers a TCP interface, in which, one or both endpoints of a standard TCP session can be replicated over a set of group members; DD provides reliable multicast mechanisms to be used in replicating critical data and to enable WS to announce communication styles to their clients; MDC provides a set of monitoring and managing mechanisms, which are capable of tracking performance metrics and other state variables; EVN is responsible for notifying interested components upon the occurrence of interesting events.

4.2.4 Adapt: Towards Autonomic Web Services

This paper [39] summarizes the effort that was exerted in the context of Adapt project, which aims at providing autonomic behavior for both single and composite WS. First, Adapt developed the *service specification language* for describing the non-functional properties of services; moreover, in terms of composition of services, Adapt defined a set of non-functional attributes that included attributes related to (1) transactional capabilities of WS, (2) sequencing constraints on WS invocations and (3) WS performance. The first two sets of WS attributes are static attributes as they do not change after deployment; therefore, such attributes are published at the deployment time. The third set of attributes contains the dynamic attributes that represent the performance values, such as response time, throughput, etc. performance attributes can be used at runtime in the context of services selection; moreover,

performance attributes can be used by analysis tools. In addition, Adapt used *replication* as the key technique for providing adaptability for single processes; this technique was applied on a three-tier J2EE architecture that is composed of WS components, application server (EJB), and database. The paper uses workflows to model composition of WS, and in the context of composite services, the project was concerned with adaptability from multiple directions, such as dynamic binding, fault-tolerance and resolution of interface mismatch and transactional semantics.

4.2.5 Architecture for an Autonomic Web Services Environment

This paper [40] proposed an architecture for autonomic WS environment, where only self-configuring and self-optimizing AC characteristics are considered by this paper. A WS environment consists of a collection of components, such as HTTP servers, application servers, database servers, and WS applications. In this architecture, each component is self-managed as well as the whole environment is managed by a site manager; a site in this architecture is a collection of components that are necessary for hosting some WS. Therefore, the architecture is composed of two levels of management that are component level and site level.

Building components in the proposed architecture is based on the principle of *reflective programming*, which is concerned with maintaining a model of self-representation that reflects the current status of the component, and changes to that model are automatically reflected to the underlying system. Each component implements a *Reflective Management Interface*, which is used by the site manager to get information from or set performance goals of the component. Adopting common management interfaces will ensure interoperability between components and site managers. Therefore, a component can be considered as a WS, such that the self-representation can be accessed via WS operations. A component is responsible for managing itself to meet some goals specified by the site manager; on the other hand, the site-wide goals are set by a component called *SLA Negotiator*, which represents the communication point between sites.

The architecture defined two management interfaces for each component: (1) the *performance interface*, which exposes methods to retrieve performance data to enable the site manager to assess the health of the component (2) the *goal interface*, which exposes methods to query and update goals to enable the site manager to set the

component's goals. Moreover, the type of data as well as the goals defined by each component can be discovered with the aid of a set of meta-data methods. With the aid of WS technologies, management interfaces of components can be described and published into a private UDDI registry that is operating in the site. The following figure illustrates the proposed architecture of this paper.

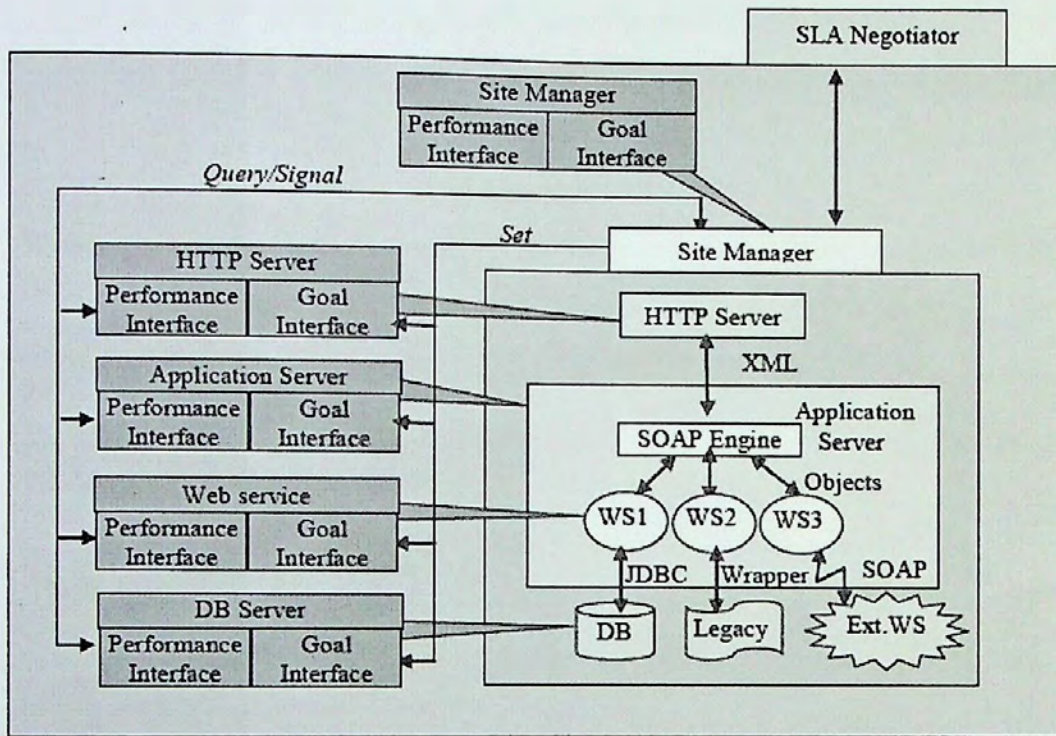


Figure 26 Autonomic Web Services Environment [40]

Chapter 5: Proposed Solution: Web Services-Based Autonomic Computing Framework

Chapter 5: Proposed Solution: Web Services – Based Autonomic Computing Framework

5.1 Chapter Overview

After reviewing different aspects of managing CS, AC and WS in chapters two and three, in this chapter, the significance of the related work that was discussed in chapter four to our research will be established. Following to that, the proposed solution to the problem that is considered by our research, which was identified in chapter one, will be discussed by introducing the WSAC Framework. Only the theoretical aspects of the proposed framework will be discussed in this chapter, and the implementation details of the proof-of-concept prototype that was built to validate the framework are discussed in Appendix D. Moreover, the validation work that was conducted in order to prove the concepts introduced by WSAC will be discussed in the following chapter. At the end of the chapter, the scope of work of WSAC in the context of our research will be identified.

5.2 Significance of The Related Work to Our Research

In this section, the significance of the related work that was reviewed in the previous chapter to our research will be highlighted as follow:

- I. The standardization of WSDM [6] can be considered a significant milestone in the context of our research for the following reasons: (1) WSDM can be adopted by AC to implement AC characteristics in the form of ManagingWS; and therefore, manage MR over the Internet, (2) WSDM specifications can be employed in implementing ATP, such that sensors implement MC, such as *Metrics*, and effectors implement MC, such as *Configuration*, and (3) being based on WS, WSDM enables WS to be automatically managed by ManagingWS.
- II. Although the concept of Autonomic Web Processes that was introduced in [37] did not consider applying AC characteristics to the physical environments that host WS, this concept can be elaborated in our research in order to permit the automated management of WS on the level of WS processes.
- III. The extended WS architecture that was proposed in [38] can be used to build a reliable communication bus, over which AC characteristics can be provided.

- IV. Some of the aspects that were introduced by Adapt project, which was reviewed in [39], can be adopted by our research. For example, describing the non-functional requirements, such as response time, of WS and defining those requirements as the performance attributes can permit the way, by which, ManagingWS manage ManageableWS.
- V. The architecture that was proposed by [40] can be applied in the context of our research to adopt the notion of AC in the level of WS environments that include different components, such as application server, database server, HTTP server, etc.

5.3 The Proposed Solution for Achieving The Thesis Objective

As it was highlighted in the thesis statement, which was declared in chapter one, this thesis is concerned with having autonomic behaviour achieved in CS by using WS technologies in building MR, AM and autonomic control loops so that heterogeneity of MR no longer represents a challenge in the context of AC as well as management no longer introduces a challenge in the context of WS. In order to achieve the objective of our thesis, which was also declared in chapter one, we propose the WSAC framework in this thesis. The framework can be illustrated with the aid of the following three models:

5.3.1 Architectural Model

The architectural model of WSAC framework describes how different components, which are MR, AM and MAPE-cycles are architected and integrated with each other with the aid of WS technologies.

5.3.2 Processing Model

The processing model of WSAC framework describes the data flow and processing among different components in WSAC architecture.

5.3.3 Metamodel

The WSAC metamodel provides an abstract description for both the architectural and processing models of WSAC framework.

The details of the three models will be discussed in the following three subsections.

5.4 WSAC Framework Architectural Model

The architecture of WSAC framework is divided into three major components, which are *Autonomic Computing Web Services (ACWS)*, *MAPE-cycles Web Services (MAPE-WS)* and MR.

5.4.1 Autonomic Computing Web Services

As it was discussed in chapter two, the notion of AC is characterized with four major characteristics, which are referred to as self-CHOP. Consequently, a typical AM should support the four characteristics; however, each AC characteristic may be supported by a separate specialized AM, and the autonomic behaviour is achieved through the integration and cooperation between specialized AM. In WSAC framework, specialized AM are implemented as WS in order to enable seamless and platform-independent communication between AM and MR. Therefore, the proposed architecture defines four major ACWS that are *Configuring Web Service (CWS)*, *Healing Web Service (HWS)*, *Optimizing Web Service (OWS)*, and *Securing Web Service (SWS)*.

In WSAC framework, there are two composability models, by which the management relationships between MR and ACWS can be established. In the first model, ACWS can communicate with MR directly to provide the required AC aspect; and in the second model, an AM can be located between MR and ACWS to act as a broker of AC capabilities. Moreover, the AM will be responsible for locating and establishing management relationships with ACWS on behalf of MR. The second model decouples MR from locating and establishing relationships with multiple ACWS; in addition, security is enhanced as an MR will trust only one entity rather than trusting multiple entities. Nevertheless, the model introduces the problem of single point of failure.

5.4.2 MAPE Web Services

In the heart of each ACWS, there is an autonomic control cycle, or *Monitor, Analyze, Plan and Execute (MAPE)* cycle; the MAPE cycle has some kind of knowledge, such as symptoms, heuristics, etc, in order to enable the ACWS to achieve its goals. In WSAC framework, MAPE cycles are also implemented as WS, which are referred to as MAPE-WS. Based on the composition of MAPE-cycles, there are four kinds of MAPE-WS, which are (1) *Monitoring Web Services (MWS)*, which is responsible for

evaluating the health of the MR being managed, (2) *Analysis Web Services (AWS)*, which is responsible for analyzing the status of some MR being managed if MWS has suspected some discrepancy in the behavior of that MR, (3) *Planning Web Service (PWS)*, which is responsible for validating the actions suggested by AWS against the policies defined by the MR being managed, and (4) *Executing Web Service (EWS)*, which is responsible for translating the management actions that are suggested by AWS and approved by PWS into a set of commands that are understandable by the MR being managed. Therefore, through WS communication, an ACWS can discover and talk with MAPE-WS at runtime. Moreover, in order to permit the notion of specialized AM, MAPE-WS can be also implemented based on some specialization, such as MR type, to capture common MAPE activities in that specialization. For example, there might be MAPE-WS that are specialized in managing a specific *Operating System (OS)*. The following points discuss the details of each MAPE-WS individually:

I. Monitoring Web Service

Upon receiving management-related information (e.g. metric) from MR, a typical Monitor performs some kind of processing in order to assess situations to detect the occurrence of failures. Metrics that can be used for monitoring can be divided into two main categories: (1) *resource metrics* and (2) *application-level metrics*. Resource metrics, such as CPU utilization, memory utilization, and bandwidth utilization, help evaluating the health of the physical environment and/or OS, in which, MR is operating. On the other hand, application-level metrics, such as response time, request time, request queue lengths, and transaction rates, help assessing the status of the MR itself. Moreover, a threshold should be defined for each metric and stored in the knowledge bases of MR. Such knowledge bases are made available to enable ACWS to establish some kind of awareness of the management requirements of MR. By consuming thresholds, MWS will be able to identify abnormal situations if they discover any deviation from the specified thresholds.

A significant note to be mentioned in the context of MWS is that monitoring can be made more sophisticated than only comparing values of the collected metrics against the thresholds defined by MR. This can be achieved by including some filtration and aggregation techniques that contributes to enhancing the decision making process of the MWS. Moreover, as it was mentioned in chapter two, although the goal of AC is

to manage CS without the need for human-intervention, the human-intervention will not be totally eliminated. One of the tasks that still need human intervention in the context of MWS is defining types of metrics that are made available by MR to ACWS, the level, by which, each metric contributes to meeting the management requirements of MR and the values of thresholds.

II. Analyzing Web Service

AWS perform analysis to derive some corrective action(s) to help adjusting MR; thus, each AWS can be considered the brain of the MAPE-cycle, in which it takes the role of the analyzer. When an AWS is invoked, it may not find any problem; for example, if the CPU utilization is over threshold for a single monitoring cycle; after analysis, the AWS may realize that the reported excessive utilization was just a momentary spike. Since analysis is a costly phase in any MAPE cycle, it should be triggered only if the MWS has suspected some abnormality. Each AWS implements a *symptoms database* that captures symptom/directive pairs, such that while solving a problem that is represented by a symptom, the associated directive with that symptom will be followed by the AWS in order to find a solution to that problem. Therefore, the goal of using symptoms database is to minimize the cost of analysis when it is possible.

Symptoms represent situations that occur in IT infrastructures; in order to unify both situations and semantics both semantically and syntactically, there have been many initiatives, such as Common Base Event (CBE) by IBM, which is concerned with unifying the way that different logs are formatted in order to permit problem-solving. CBE defines a set of 10 situations that are common to different IT infrastructures; in addition, such situations are formatted using XML elements in order to permit machine understanding. CBE can be used to enable AWS to understand the semantic of situations that are discovered and reported by MWS so that AWS can find the proper solutions. In addition, each AWS can automate the analysis process by marinating analysis practices, such that the practices are fired based on the semantic of reported situations.

If the symptoms reported by MWS are enough fuzzy to be identified or understood by AWS, AWS should perform some diagnostic analysis, such as root-cause analysis, in order to identify the root-cause of the suspected problems. There might be several approaches for performing root-cause analysis, such as incorporating AI techniques like neural networks; however, the technique that is used in our thesis is *logs-analysis*.

Therefore, logs should be unified at first by using standards, such as CBE or *WSDM Event Format (WEF)* [41]; in addition, IBM provides the *Generic Log Adapter (GLA)* tool [42] - that can be used to translate log information that is MR-specific into log information that is commonly formatted based on CBE [42] standard; however, unified log formatting is outside the scope of our thesis. Unified logs should be made available to AWS to be consumed in the root-cause analysis. Directed graphs can be used to analyze logs by representing each record in the latest logs as a node in the graph; therefore, parent nodes can be identified; and therefore, root-causes of a problem can be identified. In addition, IBM provides the *Log and Trace Analyzer (LTA)* tool [42] that can be employed in correlating events; however, events correlation is also beyond the scope of our thesis. CBE, GLA, and LTA are briefly discussed in Appendix C.

Learning skills can be added in order to enable AWS to realize new situations in order to identify uncommon problems. After realizing new situations, each AWS should update its symptoms database by adding new symptom/directive pair. This pair represents the conclusion of the each diagnostic analysis performed by the AWS, such that a new situation that is encountered for the first time will be treated as a common situation if it is encountered again.

III. Planning Web Service

An AWS may suggest more than one action in order to recover an MR from some failure; in addition, an AWS might recommend actions that violate business policies defined by the MR. In the first case, a PWS should select the best action from all the recommended actions, which is the action that brings a MR back to its normal behavior while best meeting all the business requirements of that MR; and in the second case, the PWS should request another action in order to prevent violating the business policies of the MR. Therefore, a PWS plays the role of the regulator of management decisions that are made by AWS, where that regulation is achieved by validating whether the actions recommended by AWS will violate the business policies defined by MR, which are specified by system administrators of MR, or not. In order to enable PWS to achieve its goal, there are some prerequisites, such as (1) business policies should be defined in a standard way both semantically and syntactically, (2) policies should be made available to be consumed by PWS by maintaining policies in the knowledge bases of MR and (3) translating high-level

business policies into low-level management action in order to establish a link between business and management actions so that management is not contradicted with business.

IV. Executing Web Service

After PWS approves the management actions recommended by AWS, the role of EWS is started in order to translate those actions into a set of commands, which is understandable by MR. Therefore, each EWS implements some kind of databases, in which, management actions are mapped into commands. Additionally, EWS should be able to identify when to apply commands to MR. This can be achieved by consulting the execution policies that are defined by MR the same way as business policies are defined and made available to PWS.

V. MAPE Web Services Implementation Options

Two approaches maybe considered while implementing MAPE-WS, *statically-defined* or *dynamically discovered MAPE-WS*. The first approach has the advantage of specialization because the MAPE cycle of an ACWS will be specialized in a specific autonomic management domain; in addition, because the MAPE-WS are defined to the ACWS at design time, the expected responsivity in solving problems as they occur is high as there will be no cost in discovering and locating MAPE-WS. However, this approach will not be useful in changing environments or operation conditions, where specialization is not of significance as adaptation is; and also, if any of the MAPE-WS failed for any reason, the whole ACWS would not be able to achieve its goals.

In the second approach, each ACWS discovers MAPE-WS at runtime in order to dynamically compose its MAPE cycle to provide the required AC behaviour. Clearly, the second approach is more adaptable than the first one; however, while processing each AC request, ACWS should repeat the procedure of discovering and composing MAPE-WS, and this will introduce some delay in processing AC requests. Moreover, the ACWS might fail in finding the proper MAPE-WS; and this will lead the processing of current AC to fail. This challenge can be worked around by enabling each ACWS to cache some information about the availability and performance of MAPE-WS; therefore, an ACWS will not look up a new MAPE-WS unless it is not satisfied with the MAPE-WS, whose information is currently cached by that ACWS. In addition, to enhance the responsivity of ACWS, each ACWS should periodically check the health of the known MAPE-WS and properly update the cached

information; the *ping-based approach* that was mentioned in section 3.7.1 of chapter three can be employed for this purpose.

5.4.3 Manageable Resources

The concept of MR was discussed in chapter two; however, in the context of this research, MR refers to either a computation resource that is wrapped by a WS interface for the purpose of instrumentation or a BWS.

I. Manageable Resources and Autonomic Computing Web Services

As it was discussed in ACWS, MR can interact with ACWS directly or through AM middleware. Therefore, MR should be able to discover either AM or ACWS and establish management relationships with them to outsource AC aspects. In order to reduce the time, through which, the behavior of MR is automatically kept in normal conditions, MR can cache information about either AM or ACWS so that MR do not need to locate AM or ACWS when it requests AC behavior. Moreover, MR should be enabled to periodically check both the availability and performance of cached AM or ACWS; and when any of those cached WS either becomes unavailable or starts to behave badly, MR locate new instances of those WS.

II. Autonomic Touchpoints

In WSAC, instrumenting WS interfaces, consolidated BI and MI are ATP, through which all the communication between MR and ACWS is handled. Each MR is wrapped with the aid of an ATP; however, one ATP can be used to wrap multiple MR [20]. Because of its extensible nature, WSDM can be used in the context of implementing ATP, where AC capabilities can be built on top of the set of MC that were introduced by WSDM. A typical ATP is a WS interface that is composed of a set of sensors and effectors; a sensor is an endpoint that exposes management-related information about MR to ACWS; and an effector is an endpoint, through which, ACWS can control the behavior of MR by executing some actions. Therefore, an ATP should be equipped with various mechanisms, such as sending events, providing data upon demand (e.g. when queried), accepting commands through APIs, etc.

III. Supporting Tools

Through ATP, MR should expose a set of information and tools to be used by ACWS to provide autonomic behavior to MR; all the exposed information should be understandable by ACWS. For example, metadata can be exposed to communicate the properties of MR, such as identity, metrics, state, etc, to ACWS; moreover, policies

and logs should be made available through some kind of knowledge bases. In addition, MR should expose different management mechanisms, such as operations and events.

Based on the discussed architectural model of WSAC framework, figure 27 illustrates the proposed architecture:

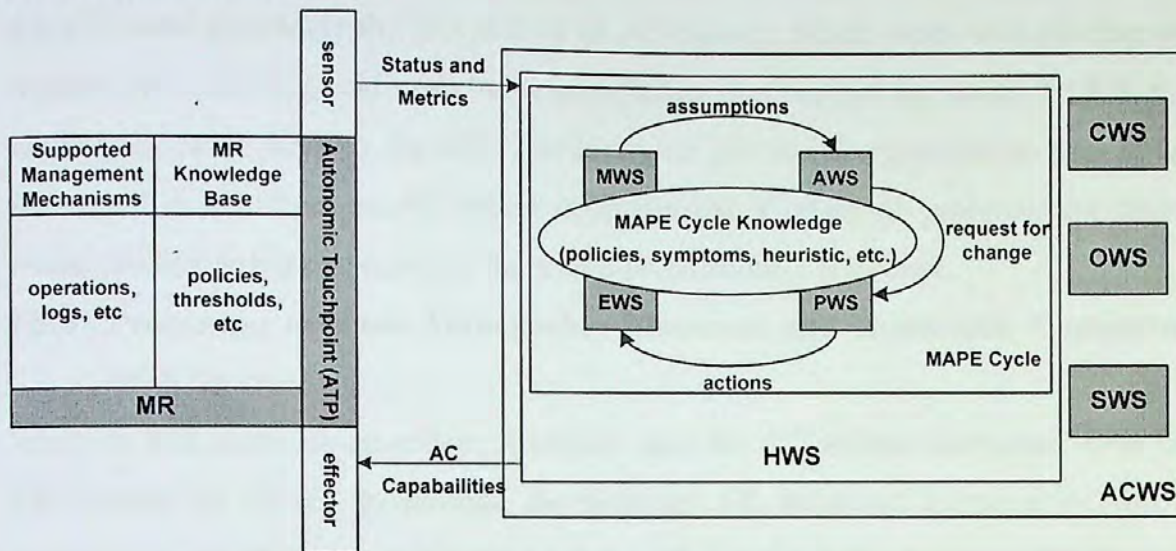


Figure 27 Architectural Model of WSAC Framework

5.5 WSAC Processing Model

Before any data can be processed by WSAC framework, WS of MR, which are sensor and effector WS, ACWS, and MAPE-WS should be published into some public WS registry so that such WS can be discovered and invoked at the runtime. In addition, when the WS of a MR are published, the type of the MR is associated with the information being published; similarly, both ACWS and MAPE-WS include the type of resources they can manage while they are publishing their WS. The processing model of WSAC framework can be divided into two major sections; the first section describes the management relationships among MR and ACWS, and the second section describes the relationships among ACWS and MAPE-WS. The second section can be further divided into four sub-sections, such that each subsection describes the relationships among ACWS and each of the MAPE-WS. Before discussing the details of WSAC processing model, the following terms should be defined at first.

5.5.1 Autonomic Computing Requests

An *AC request* is a request that is sent by an MR to an ACWS in order to outsource an AC aspect from some ACWS.

5.5.2 Autonomic Computing Infinite Loops

An AC infinite loop intends to keep autonomic computing requests to be sent automatically and periodically. An infinite loop is terminated only if the MR is taken off-line for any purpose.

5.5.3 Autonomic Computing Rounds

An *AC round* represents the lifecycle of an *AC request*, which starts with sending the request by some MR and ends with completing the request by some ACWS and sending the result back to the MR. Therefore, an AC round represents on turn in the AC round; in addition, an AC round is terminated if either no problem was found within the MR being managed, or the found problem has been fixed.

5.5.4 Processing between Manageable Resources and Autonomic Computing Web Services

When an MR starts its operation, it should start the AC infinite loop; and then, the MR locates an ACWS to provide the required AC behavior. Locating ACWS is guided by some criteria (e.g. *resource type*), such that the MR locates ACWS that can provide AC capabilities to a specific type of resources (e.g. Windows XP OS, Routers, Apache Tom Cat application server, etc). Once the required ACWS has been located, the management relationship between the MR and the located ACWS is established by invoking an *entry method* on ACWS. Such relationship is established by communicating the required management-related information to the ACWS, such as the significant parameters that should be monitored (e.g. CPU utilization), the reference to the knowledge base maintained by the MR, the reference to the logs repository, etc.

During that management relationship, the ACWS retrieves management-related information from the MR through the sensor of the latter; such information can be retrieved from MR in different approaches. One approach is polling the MR for getting the required information per each AC round; another approach is to enable the MR to push information of significance to the ACWS. A Hybrid approach that combines both approaches can be used, such that *push-mode* is the default, and if the MR is unable to push the information to the ACWS, *poll-mode* is automatically activated. Moreover, an event-driven mechanism, by which, the MR notifies the ACWS when significant events occur can be used in this respect.

In addition, ACWS applies management actions to the MR through the effector of the latter. After receiving a feedback from ACWS, the AC round is repeated; however, if no feedback has been received from ACWS for a specific period of time, the MR finds another ACWS.

5.5.5 Processing between Autonomic Computing Web Services and MAPE Web Services

The following scenario describes the lifetime of processing an AC request by an ACWS:

- i. When the management relationship is established between an MR and an ACWS, as discussed in the previous point, the ACWS receives an AC request for providing some AC aspect.
- ii. The ACWS composes the MAPE cycle if the cycle is not statically defined. If the ACWS will compose the MAPE cycle, the composition will be guided by the same criteria that were used by the MR to locate the ACWS. Therefore, if the used criterion is the resource type, the ACWS should locate MAPE-WS that are related to the management domain of the MR resource type.
- iii. If the ACWS failed to retrieve such information from the MR, it would invoke the located Analyzer directly to find the reason behind this failure.
- iv. If the ACWS has successfully retrieved the required information, the ACWS sends that information together with the reference to the knowledge base of the MR to the located MWS.
- v. MWS checks the received information against the thresholds defined in the knowledge base of the MR, and reports the result to the ACWS.
- vi. If no feedback has been received from the MWS for a specific period of time, the ACWS finds another MWS.
- vii. If the thresholds were not exceeded, this means that the MR behaves normally. Therefore, the ACWS do not apply any action, but it only notifies the MR in order to terminate the active AC round.
- viii. For each exceeded threshold, ACWS should complete the remaining phases of the MAPE cycle to find the cause; and in the case of failure, find the recovery action, and apply that action.
- ix. ACWS passes the result of the MWS to the located AWS, which analyzes the situation. The AWS starts analyzing any situation by checking its Symptoms

Database to check whether the reported situation is known or is a new situation. If the situation is known to the AWS in the form of symptom, the AWS will use the directives associated with that symptom to check whether there is a real problem or not. If the situation is unknown, the AWS performs logs-analysis to the logs of the MR to identify the cause of the failure and recommends some recovery actions. In addition, in order to upgrade the analysis capabilities of AWS, an AWS should update its Symptoms Database by adding the new symptom together with the recommended recovery action.

- x. If no feedback has been received from the AWS for a specific period of time, or the AWS has reported its inability to analyze the reported situation, the ACWS finds another AWS.
- xi. ACWS passes the result of the AWS to the located PWS, which validates the recommended action by the AWS against the business policies of the MR. If the recommended action by the AWS does not violate any of the business policies of the MR, the PWS approves it. Otherwise, the PWS will request another action. In the last case, the ACWS will either report to the located AWS the result of the PWS, or it might locate another AWS to find another recovery action.
- xii. If no feedback has been received from the PWS for a specific period of time, or the PWS has reported its inability to plan the recovery from the situation, the ACWS finds another PWS.
- xiii. ACWS passes the approved recovery action by the PWS to the located EWS, which translates the action into a set of commands that are understandable by the MR. Executing the commands is scheduled according to the policies of the MR.
- xiv. Eventually, the ACWS applies the commands representing the recovery action to the MR in the suitable time, and by executing those commands the active AC round is terminated.

Based on the discussed processing model of WSAC framework, the activity diagram that is shown in figure 28 illustrates different activities that are performed during one AC round.

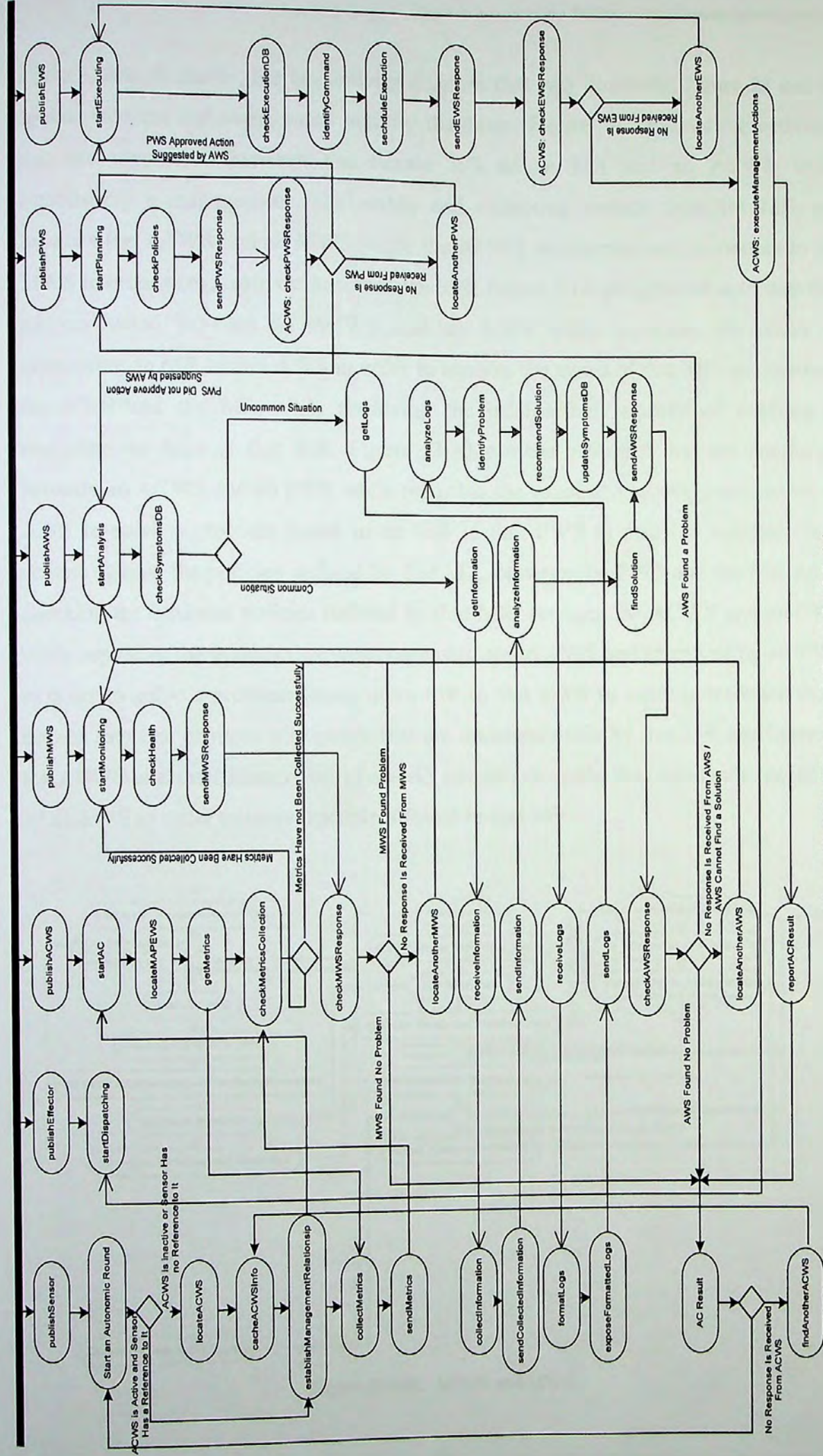


Figure 28 WSAC Framework Processing Model

For the sake of clarity, the big activity diagram that was illustrated figure 28 can be broken into the following small activity diagrams. Figure 29 captures the activities that are conducted between the Sensor WS of an MR and an ACWS while establishing a management relationship and collecting metrics from the MR and between the ACWS and an MWS while the ACWS is communicating metrics to the MWS in order to evaluate the health of the MR. Figure 30 highlights the activities that are conducted between an ACWS and an AWS while reporting the result of monitoring an MR to that AWS in order to analyze the status of that MR and between the AWS and the MR while retrieving the information required of analysis or analyzing the logs of that MR. Figure 31 shows the activities that are conducted between an ACWS and an PWS while reporting the suggested recovery actions by an AWS to solve a problem found in an MR to that PWS in order to validate those actions against the policies defined by that MR, between the PWS and the MR while checking the business policies defined by that MR, between the ACWS and an EWS while reporting the actions that were suggested by an AWS and approved by an PWS in order to solve a problem found in an MR to that EWS in order to translate those actions into one or more commands that are understandable by that MR and between the ACWS and the Effector WS of an MR in order to apply the commands identified by an EWS in order to solve a problem found in that MR.

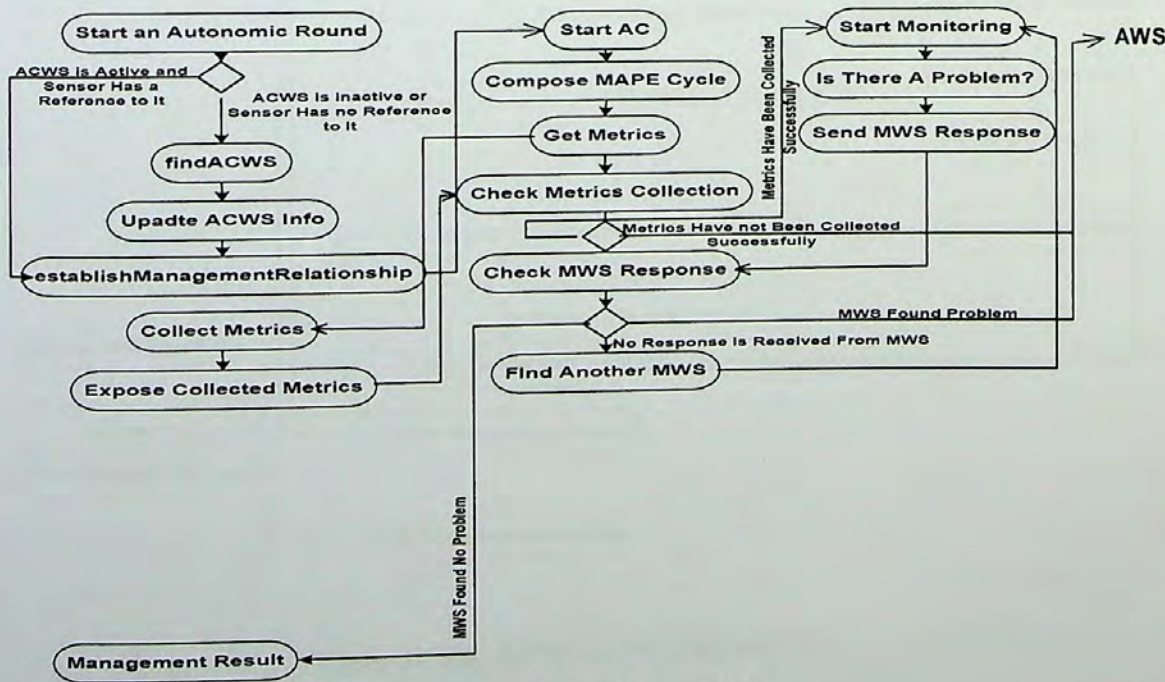


Figure 29 MR, ACWS and MWS

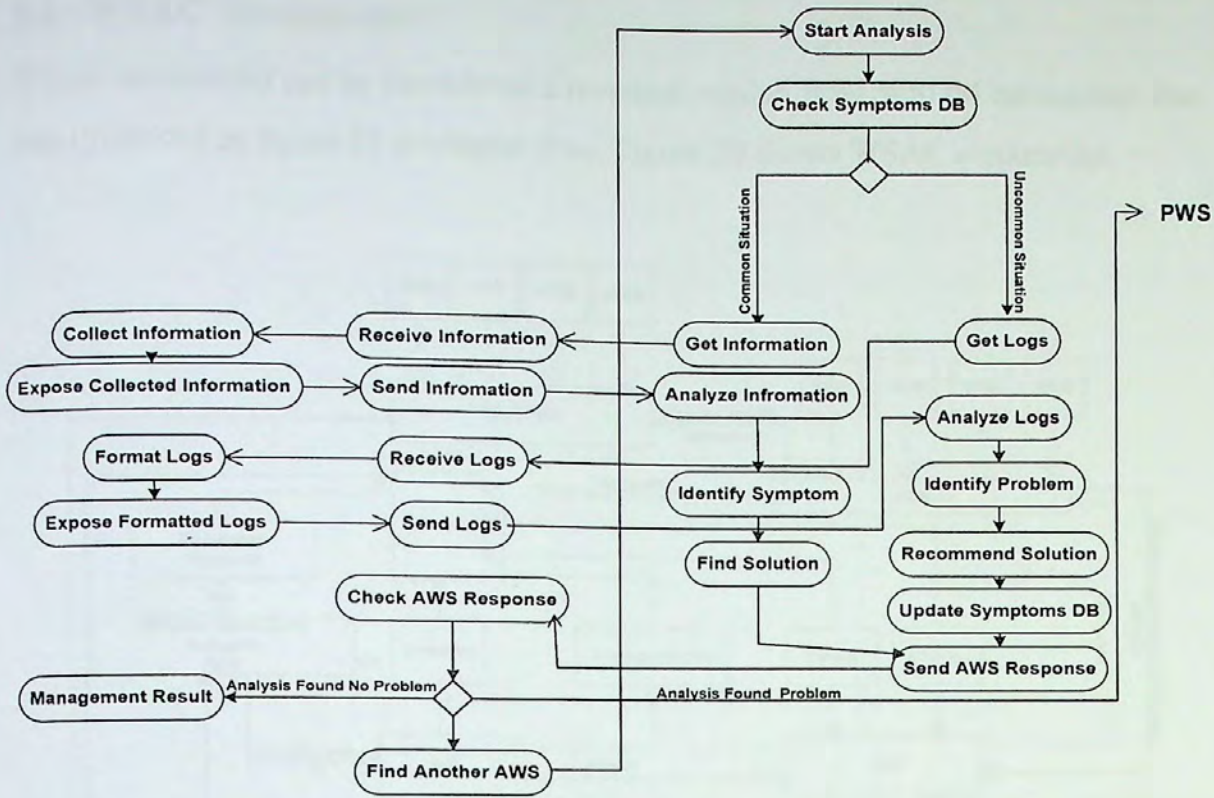


Figure 30 MR, ACWS and AWS

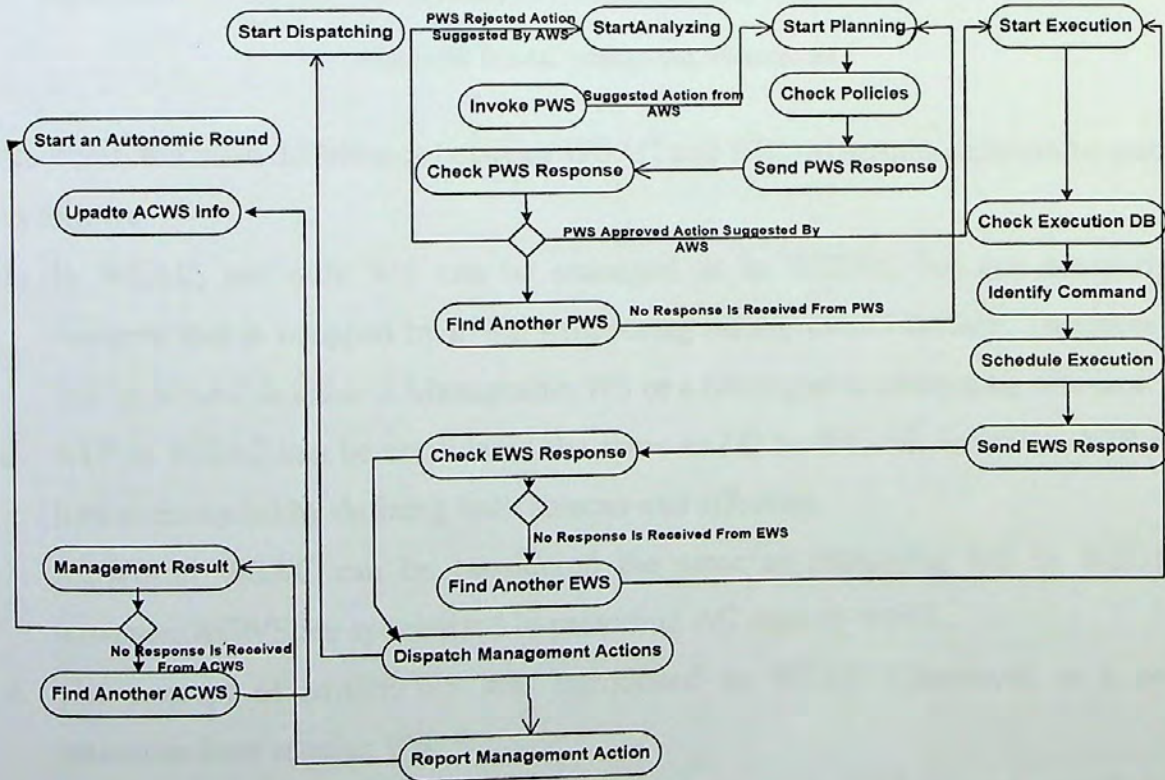


Figure 31 MR, ACWS, AWS, PWS and EWS

5.6 WSAC Metamodel

WSAC metamodel can be considered a revisited version from WSDM metamodel that was illustrated in figure 21 in chapter four. Figure 29 shows WSAC metamodel:

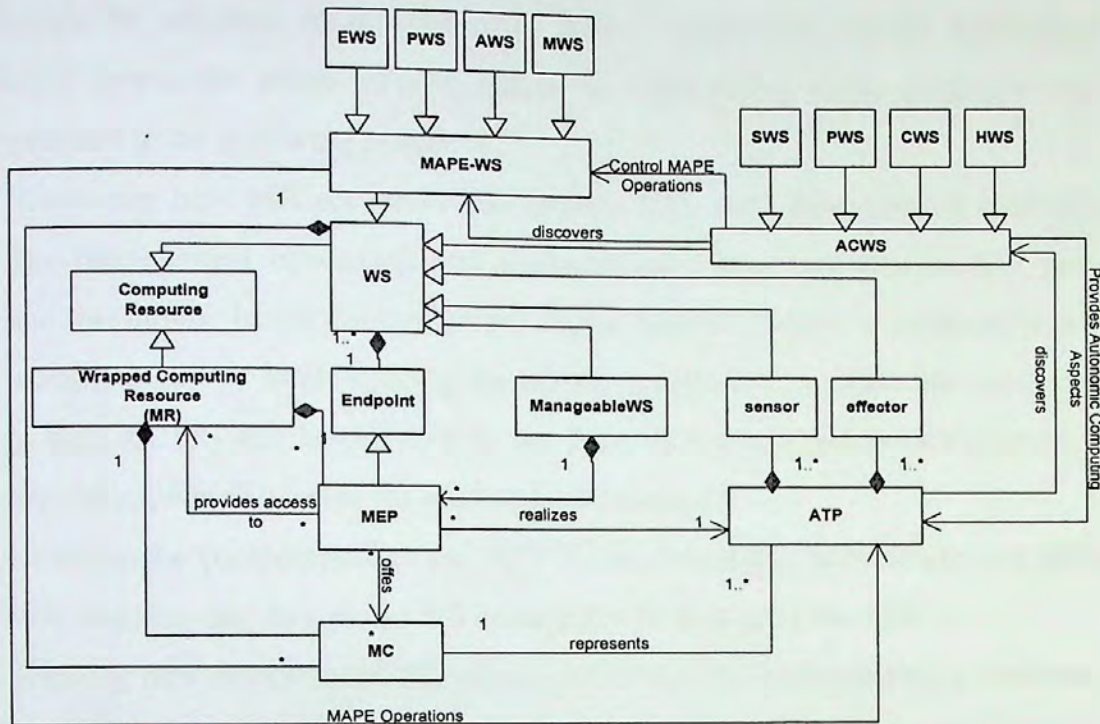


Figure 32 WSAC Framework Metamodel

However, the main differences between WSAC and WSDM metamodels can be stated as follows:

1. In WSAC, not only WS can be managed as in WSDM, but any computing resource that is wrapped by a WS wrapping management interface. Therefore, a MR in WSAC is either a Manageable WS or a Manageable computing resource.
2. ATP in WSAC can be considered the same as MI in WSDM; however, ATP are further extended by defining both sensors and effectors.
3. ACWS in WSAC can be considered the same as Managing WS in WSDM; however, ACWS are specialized in providing AC aspects to MR.
4. The concept of MAPE-WS was introduced by WSAC framework as a new extension from regular WS.
5. Unlike WSDM, in WSAC, a MR can discover ACWS and establishes management relationships with those discovered ACWS. Therefore, management

actions are initiated by MR, not by MS, in order to permit the notion of self-management.

5.7 Scope of Work

Although the concepts of the proposed WSAC framework can be highlighted in different ways, the scope of our thesis in highlighting those concepts can be summarized in the following points:

1. Illustrating how MR resources can expose MC, such management mechanisms like management operations and management-related information like policies and thresholds. In the context of our thesis, this illustration is achieved in a very primitive way by implementing the knowledge base of the MR that made shared to both ACWS and MAPE-WS in the form of a simple database that contains a few tables, which capture the required information.
2. Defining the composition of the ATP by implementing both sensor and effector WS, and showing how those WS contributes to managing the MR.
3. Showing how discovery of WS can be performed by implementing a database that acts as a private WS registry, and using this database to simulate publish/discover operations.
4. Illustrating how an ACWS can act as a middleware between MR and MAPE-WS, such that management relationships are established among MR and ACWS; and then, ACWS locates MAPE-WS to compose MAPE-cycles based on the type of MR. In addition, on behalf of MR, ACWS control all MAPE operations.
5. Introducing the functionalities of MAPE-WS in a primitive way, in which, each MAPE-WS in a typical MAPE-cycle performs its functionality in a straightforward manner that is based on performing some simple comparisons for identifying problems, finding solutions, approving those solutions; and eventually, executing them.
6. Highlighting the achieved reliability by using WS that enable MR to locate new ACWS and ACWS to locate new MAPE-WS when the currently located ones failed or started to behave badly.
7. Emphasizing how MR can be dynamically managed by ACWS.

5.8 Chapter Summary

By reviewing the promises and challenges of both paradigms of AC and WS that were discussed in chapters two and three, the significance of each paradigm to the other has become clear. This significance has motivated the work of adopting each paradigm by the other, such that the paradigm of WS is used in the context of AC in order to overcome the obstacle of heterogeneity that was highlighted in the end of chapter two, and the paradigm of AC is used in the context of WS in order to automate the management of WS as it was requested in the end of chapter three. Achieving this two-directional goal is the focus of this research. Therefore, the WSAC framework is proposed by this research; the framework is concerned with implementing AC aspects with the aid of WS technologies; however, the focus of our research is on the self-healing AC characteristic. In addition, by using WS technologies, MC of either legacy computing resources or BWS can be exposed in the form of ATP to dynamically outsource AC aspects that are also implemented as WS.

Chapter 6: WSAC Framework Validation

6.1 Chapter Overview

The focus of this chapter is on discussing the work that was conducted for validating the concepts that were introduced by WSAC framework and proving the framework achieves the objective of our thesis. In order to validate WSAC framework, a proof-of-concept prototype was built. The details of that prototype, such as design, used tools, and the results will be discussed in this chapter. At the end of the chapter, the contribution of the prototype to validating WSAC will be highlighted

6.2 Proof-of-Concept Prototype

In order to validate the introduced concept of WSAC, a proof-of-concept prototype was built. The prototype captures all the architectural and processing issues of the framework that were discussed in the previous chapter. Therefore, our prototype can be thought of as the initiative of implementing fully-fledged CS that are complied with the concept of WSAC. Before discussing the details of the different components contributing to building the prototype, the management scope of the prototype is declared in the following subsection.

6.2.1 Prototype Management Scope

The focus of the prototype is on the healing AC aspect; therefore, a HWS is implemented to introduce the concept of ACWS. In addition, the management scope of the prototype is limited to the following:

- I. Releasing the CPU of the MR being managed in the case of CPU over-utilization.
- II. Ensuring that metrics that are always available by the MR being managed to ACWS.

6.2.2 Prototype Architecture

For the sake of simplicity, the first composability model that was discussed in section 5.6.1 of the previous chapter, in which MR communicates directly with ACWS, is adopted in the building of the proof-of-concept prototype. The following components were used in building the architecture of our prototype:

I. Manageable Resource

As it was mentioned in section 5.6.1 of the previous chapter, an MR can be either a computing resource that is instrumented by a WS interface, or a BWS that provides both BC and MC. In the context of the prototype being discussed; a Microsoft Windows XP OS was used to take the role of MR in our prototype. The advantage of that selection is to involve a realistic environment in our prototype; and therefore, the applicability of our prototype to industrial CS will be clearer than simulating the MR as simulation will not capture different operational scenarios. The BC of the MR in our prototype are the capabilities that the OS can provide to applications running in its environments; specifically, maintaining the CPU performance in the normal behaviour is the focus of our prototype as it is discussed below.

II. Supported Manageability Mechanisms

In our prototype, the OS supports a set of mechanisms, through which, the OS can expose its MC to HWS. These mechanisms include the following:

1. The OS provides a set of *native APIs* that expose some real-time information that is related to management, such as the started services. In addition, through those APIs, HWS can apply management actions, such as starting and stopping services.
2. *Nagios Agent* (e.g. NSClient Service) [43] runs in the OS real-time environment in order to expose information about the metrics of the OS, such as CPU utilization, free disk space, networking status, etc. Through this service, the OS communicates its metrics to an HWS in order to start the MAPE cycle. The exposed set of metrics in the context of our prototype includes only the CPU utilization.
3. The *Command Line Process Viewer/Killer/Suspender for Windows NT/2000/XP* (e.g. *process utility*) [44] runs in the MR-real-time environment to expose MC related to the management of processes running within the context of the OS. Such MC include starting, killing and suspending processes.
4. The OS also provides system logs in the form of log files that can be accessed and analyzed by AWS. In order to permit interoperability between MR and ACWS, logs must be unified both syntactically and semantically by standards, such as CBE and WEF; however, this issue is beyond the scope of our prototype.

Although the external tools that were used in implementing our proof-of-concept prototype will be defined in Appendix C, the following table summarizes those tools

Table 1 List of External Tools Used in Implementing the POC prototype

| Tool | Use |
|-----------------|--|
| Nagios Agent | Instrumenting the OS for retrieving the real-time values of the OS counters |
| Process Utility | Instrumenting The OS for retrieving the active processes running within the OS and controlling the lifetime of the running processes |

III. Knowledge Base

For the sake of simplicity, the knowledge base of the MR is implemented in the form of a Database that includes different tables for capturing different kinds of information that is related to the context of management. Although, the knowledge base of a typical MR should provide a rich set of information to be consumed by ACWS to achieve as efficient and effective management as possible, the knowledge base of the MR (e.g. operating system) implemented by the prototype stores the following information:

1. Important metrics to be monitored together with the threshold of each metric. Stored metrics can be used by the MR to inform an HWS about the significant metrics that should be monitored. Moreover, associated thresholds can be used by MWS to determine whether the measured values of metrics exceed the thresholds; and therefore, to determine whether there is a need for analysis or not.
2. Processes that cannot be killed and should be running always, such as system processes. Recorded processes can be consumed in the planning phase by PWS to approve or reject some recommended management actions by AWS.
3. Services that should be started to activate some features. Tracked services can be consumed in the analysis phase by AWS to build an overview of the started and stopped services in the MR environment.
4. Delays that should be enforced during the execution of some commands. EWS can use those delays in order to schedule the execution of commands on the MR.

IV. Autonomic Touchpoint

In our prototype, the ATP decouples the OS from HWS as it represents the communication point between them. Therefore, a legacy computing resource like the OS can be managed by HWS without requiring any changes to be applied to the OS as all the required changes will be developing the ATP that captures and exposes the MC of the OS to evolve its management to the paradigm of ManagingWS. Moreover, as mentioned before in the previous chapter, multiple ATP can be used to represent the OS, such that each ATP captures a specific aspect related to the management. However, in our prototype, only one ATP is implemented to provide the communication between the OS and HWS. The ATP is implemented by using two WS, which are Sensor WS and Effector WS. Generally, the Sensor WS retrieves the required information from the OS by performing get operations with Windows Native APIs, NSClient Service and the Process Utility, and the Effector WS applies the required actions to the OS by performing set operations with both the Windows Native APIs and the Process Utility.

However, the exact operations performed by the Sensor WS are: (1) running the infinite loop that keeps the OS behaving autonomically by maintaining management relationships with ACWS as long as the OS is running, (2) locating an HWS based on the resource type, which in our prototype is Windows XP OS, (3) establishing a management relationship with the located HWS, (4) periodically checking the health of the located HWS, and locating another HWS if the located one has failed and (5) exposing the information that is related to management, such as available metrics (e.g. CPU utilization), active processes, running services, reference to the knowledge base of the OS, etc. Therefore, the Sensor WS wraps the following manageability mechanisms supported by the OS: (1) the OS native APIs in order to expose the running services, (2) the NSClient service in order to expose the CPU utilization, (3) the process utility in order to expose the active processes together with their CPU utilization, and (4) the log files in order to expose system logs. In addition, through the Effector WS, the HWS applies actions, such as kill process, start service, etc. Therefore, the Effector WS wraps the following manageability mechanisms supported by the OS: (1) the OS native APIs in order to control the lifetime of services (e.g. start and stop) and (2) the process utility in order to control the life time of processes (e.g. kill, suspend, resume, etc).

V. Healing Web Service

Since the focus of this research is on the healing AC aspect, only the HWS is implemented in our prototype. HWS decouples the OS from MAPE-WS; therefore, the OS is only aware of HWS and knows nothing about MAPE-WS; the advantage of this model was discussed in the previous chapter. HWS implements a WS interface, through which HWS communicates with the ATP, MWS, AWS, PWS, and EWS. HWS retrieves information from the sensor; and if any MAPE-WS requires any information from the OS, requesting the information is handled through HWS. From the sensor, HWS (1) knows the available metrics to be collected, (2) collects each of the available metrics, (3) retrieves active processes and running services and (4) accesses the log files of the OS. In addition, through effector, HWS applies the healing commands to the OS. A typical HWS is specialized in healing resources of a specific type; in our prototype, the HWS is specialized in healing Windows XP OS resources.

Moreover, as it was discussed before, HWS is responsible for locating and checking the health of MAPE-WS. In our prototype, locating MAPE-WS is based on the resource type. After locating the MAPE-WS, HWS invokes each of the MAPE-WS separately and processes the response of each one before it invokes the next one in the MAPE cycle; the invocation procedure was discussed in the processing model. In addition, HWS checks the health of MAPE-WS, such that if any of the MAPE-WS does not send a response back to the HWS within a specific period of time after invoking it, the HWS will locate another MAPE-WS to take the role of the not-responding MAPE-WS in the MAPE cycle.

VI. Monitoring Web Service

MWS implements a WS interface, through which, it can communicate with HWS as MWS is invoked by HWS in order to assess the status of the OS in terms of the CPU utilization that was measured and reported by the HWS. When MWS receives the metrics from HWS, it checks the measured values against the thresholds that are associated with each metric in the knowledge base of the OS. The response of the MWS reflects whether the measured value of each reported metric is below its corresponding threshold or not. In our prototype, MWS is specialized in monitoring Windows XP OS metrics; specifically, CPU utilization.

VII. Analyzing Web Services

AWS implements a WS interface, through which, it communicates with HWS as AWS is invoked by HWS to analyze the status of the OS. AWS is invoked in two cases: (1) if the HWS is unable to collect metrics from the OS and (2) if the HWS has received a response from MWS indicating that the threshold of some metric has been exceeded. Regardless of the condition, under which, AWS is invoked, AWS receives a description of the situation (e.g. a symptom) of OS from HWS. The first action in the analysis procedure that is taken by AWS is to check its symptoms database for the reported situation. If the symptom is common to AWS as it is tracked by the symptoms database, the AWS follows the directives in finding a solution for the problem represented by symptom. If the symptom is not common to AWS (the symptom is not recorded in the symptoms database), the AWS accesses the log file of the OS through HWS, and locates the most recent recorded events in the file. A typical recorded event may inform the AWS that some service has been stopped. By correlating those events, AWS would be able to identify the cause of the problem; and therefore, suggest the solution.

Finally, the response of AWS may reflect that the analysis has concluded that no problem was found and that the OS behaves normally; in this case, the response of MWS is likely affected by some instantaneous deviation in the performance. On the other hand, if AWS discovers a problem, its response will be one or more healing actions to fix that problem. In our prototype, AWS is specialized in analyzing situations occurring in the Windows XP OS environment; specifically, abnormal CPU utilization and failed metrics collection; the recommended action in the first situation will be killing a specific process, and the recommended action in the second situation will be starting the service that is responsible for exposing metrics.

VIII. Planning Web Service

PWS implements a WS interface, through which, it communicates with HWS as PWS is invoked by HWS whereas PWS is invoked only if AWS has a detected some discrepancy and reported it to HWS. As mentioned before, the goal of PWS is to permit policy-based management, such that no management actions are taken within a CS if these actions violate the policies of that CS. In our prototype, the recommended healing action by AWS is delivered to PWS to check the applicability of that action. Therefore, in order to achieve its goal, PWS accesses the policies of the OS through

HWS. The response of PWS is either approving the recommended action by AWS or requesting another action to be recommended in order to maintain the policies. In the first case, HWS will proceed through the MAPE cycle and invoke EWS. In the second case, HWS will either require another healing action from the same AWS or locating another AWS to recommend the new action.

IX. Executing Web Service

EWS implements a WS interface, through which, it communicates with HWS as EWS is invoked by HWS after PWS has approved the recommended healing action by AWS. EWS acts as a translator that translates the healing action into commands that can be understood by the OS. In our prototype, EWS is specialized in executing commands in Windows XP OS environment; therefore, EWS maintains a simple database that maps the action to actual command. Before passing the command to be executed to HWS, EWS checks if there are any execution policies defined by the OS to decide when the command should be executed; or in other words, execution policies help EWS scheduling commands as required. For example, a backup action may be scheduled after working hours. Therefore, the response of EWS to HWS is the command to be executed together with the time period, after which, the command should be executed. The response of EWS is then delivered by HWS to the OS through the effector.

X. Private Web Services Registry

As mentioned in the discussion of the components involved in the architecture of our prototype, ATP is implemented out of two WS that are sensor and effector; moreover, HWS and MAPE-WS are also implemented as WS. In order to enable dynamic discovery and communication between the ATP and HWS from one side and HWS and MAPE-WS from the other side, all those WS should be published in a service registry that is known to all the WS. In our prototype, a database called *Private UDDI* was implemented to simulate the function of that service registry. The concept of specialization is captured by associating an identifier with the record of each registered WS to identify the resource type of the ManageableWS or the resource type that the ManagingWS can manage. Moreover, each WS in our prototype has a method called *publish ()* that is manually invoked to add its record in the service registry database.

Based on composition of the prototype that has been discussed, figure 33 illustrates the architecture of that prototype in terms of the composition of that architecture:

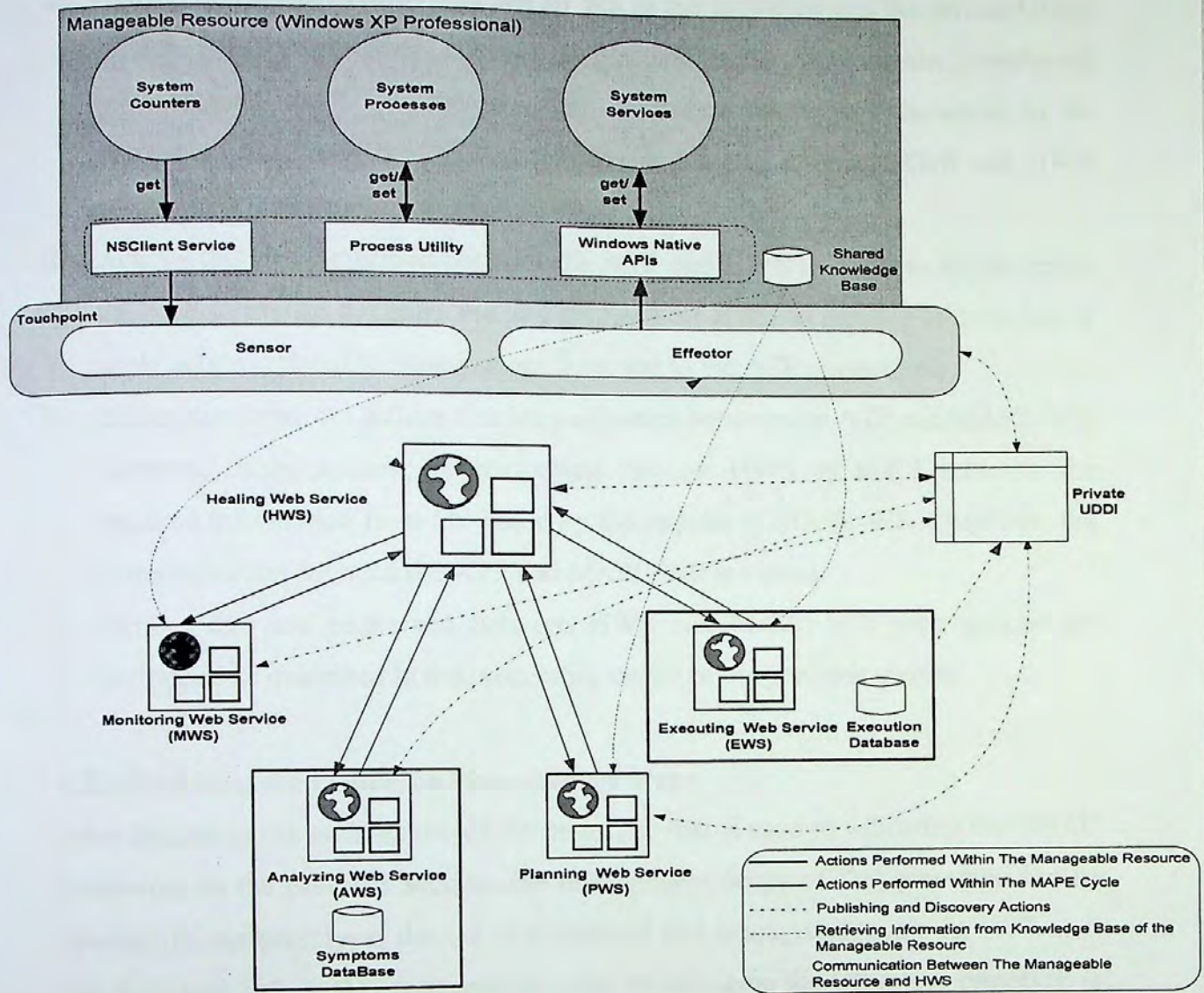


Figure 33 Architecture of the Prototype

6.2.3 Prototype Communication Patterns

As it is shown in the previous figure, the actions that are performed within our prototype (e.g. patterns of communication between different components in the prototype) are divided into five types of actions as follows:

- I. Actions that are performed within the MR are the actions that are performed in the OS environment. Those actions are either *get* or *set* operations that are performed by (1) NSClient service to get system counters by the sensor, (2) process utility to either get active processes by the sensor or set the status of some process by the

- effector and (3) OS native APIs to either get the running services by the sensor or set the status of some service by the effector.
- II. Actions that are performed between all WS in our prototype and the private UDDI service registry to either publish or discover some WS. In the context of publishing, all the WS in the prototype can access the registry; however, in the context of discovery, the ATP access the registry to discover HWS and HWS access the registry to discover MAPE-WS.
 - III. Actions that are performed between the ATP and HWS in order to enable either the ATP to invoke the entry method on HWS or HWS to retrieve information or apply actions related to management from and to the ATP respectively.
 - IV. Information retrieval actions that are performed between the ATP and MAPE-WS; however, those actions are performed through HWS as HWS retrieves the required information from OS based on the request of MAPE-WS. Therefore, the communication between the ATP and MAPE-WS is virtual.
 - V. Actions that are performed between HWS and MAPE-WS; such actions are performed as described in the processing model of the previous chapter.

6.2.4 Revisiting the Prototype Management Scope

After discussing the architecture of the prototype that is used in validating the WSAC framework in the previous section, the management scope of that prototype can be detailed. In our prototype, the OS is monitored and managed through the ATP as it was discussed before, the management scope of managing the OS in our prototype is limited to two simple management activities that can be dynamically applied to the OS, which are (1) killing some process that over utilizes the CPU and (2) starting/stopping some service. Therefore, MC of the OS as well as the management capabilities of HWS and MAPE-WS are also limited to the following capabilities: (1) the OS exposes real-time information only about CPU utilization, (2) monitoring CPU utilization, (3) detecting CPU over utilization; an important note is that it may be normal for some processes to over-utilize the CPU; therefore, such processes should be identified as CPU over-utilizing because of such processes should not be considered a problem, (4) analyzing CPU over utilization by analyzing running processes, (5) determining processes that over utilize CPU, (6) killing processes that over utilize CPU based on the OS policies that are recorded in the knowledge base of

the OS to determine which processes cannot be and can be killed, (7) analyzing logs to determine the root-cause of some detected problem, (8) detecting services that have been stopped, (9) starting services that should be started and (10) scheduling the killing of processes and starting of services based on the OS execution policies recorded by the OS knowledge base.

6.3 The Prototype Implementation

The details of the proof-of-concept prototype implementations are discussed in Appendix D; however, the following sections introduce an overview of the implementation by discussing the assumptions considered while implementing the prototype, listing the tools used in the implementation, and highlighting the technique adopted in the implementation process. In addition, at the last section of Appendix D, a set of use cases are introduced to discuss different scenarios occurring during the operation of the prototype.

6.3.1 Prototype Implementation Assumptions

- I. MR is able to discover the capabilities of HWS, and HWS can discover the capabilities of MAPE-WS.
- II. HWS are able to understand the MC of MR and the management capabilities of MAPE-WS.
- III. The available set of metrics to be monitored includes only the CPU, and the threshold associated with CPU is 10%.
- IV. The following two situations are considered by our prototype:
 1. Abnormal CPU Utilization.
 2. Metrics Could Not Be Collected.
- V. The following two healing actions are considered by our prototype:
 1. Killing a process that over utilizes the CPU.
 2. Starting the NSClient service if it is intentionally stopped.
- VI. The following list contains the processes that cannot be killed:
 1. AGRSMMSG.exe.
 2. alg.exe.
 3. ccApp.exe.
 4. ctfmon.exe.

5. eclipse.exe.
 6. explorer.exe.
 7. java.exe.
 8. javaw.exe.
 9. Rtvscan.exe.
 10. services.exe.
 11. svchost.exe.
 12. WCESCOMM.EXE.
 13. All process owned by the System.
- VII. The two following execution policies are considered by our prototype:
1. Killing a process must be started 5 seconds after specifying the proper command.
 2. Starting a service must be started 2 seconds after specifying the proper command.
- VIII. The following information is recorded per each registered WS in the private UDDI:
1. *portType* name that represents the interface, which captures the signature of the WS.
 2. *namespace* that represents the namespace, in which, data types used in the WS are defined.
 3. *endpoint* that represents the physical location of the WS, to which, requests are sent.
 4. Management scope, which is Windows XP in our prototype.
- IX. All the information recorded by the knowledge base of the OS is formatter in an understandable way for both HWS and MAPE-WS.
- X. The OS log file is formatted in an understandable way to AWS.
- XI. All the activities occurring in the OS environment are logged.

6.3.2 Prototype Implementation Tools

The following table lists the tools that are used in implementing the proof-of-concept prototype; however, those tools are discussed in Appendix C:

Table 2 List of Tools Used in Implementing the Prototype

| Tool | Version | Use |
|-----------------------------------|---|---|
| Java | J2sdk-1_4_2_08 [45] | Coding Language |
| Java Virtual Machine (JVM) | J2re-1.4.2_08 [45] | Java Runtime Environment (JRE) |
| Eclipse | 3.1.0 [46] | Software Development Kit (SDK) – Developing Environment. |
| Eclipse Web Tools Platform (WTP) | 3.1.0 [47] | Plug-in for Eclipse SDK to Support Developing WS by the SDK. |
| Microsoft Windows XP Professional | Version 2002 Service Pack 2 | Developing and Operational OS |
| Microsoft Office Access | 2003 | Implementing Private UDDI, shared knowledge base and symptoms database. |
| Apache Tom Cat | ApacheTomCat5.0 [48] | Application Server for Hosting Developed WS |
| JAX-RPC | Jax-1_1-fr-qname-class jaxrpc-1_1-fr-spec-api [49] | Java API for XML-Based RPC |
| nsclient4j | nsclient4j-0.1b [50] | Java API for NSClient Service |

6.3.3 Prototype Developing Technique

In our prototype, the followed strategy in developing WS was that a J2EE Java class is created, such that the class captures all the public and private methods of each WS in the prototype; and then, with the aid of Eclipse SDK that is listed in the previous table, the created J2EE Java class is converted into a WS, such that all the required documents, such as WSDL documents of WS are automatically generated. Moreover, by integrating the Eclipse SDK with the Apache Tom Cat application server, the generated WS are automatically deployed to the application server so that each WS can be tested as it will be discussed in the testing section.

6.4 Prototype Testing

For each generated WS by the Eclipse SDK, a test *Java Servlet Page (JSP)* client was created in order to test both the functionality of each WS in our prototype separately and the integration between those WS as required. Testing JSP clients are automatically generated from the created WS with the aid of the SDK. Moreover, testing JSP clients are running on the Apache Tomcat application server that is

integrated with the SDK. In addition; and in order to add some credibility to our prototype, the prototype was tested with the aid of two different setups, which are discussed below. After discussing each setup individually, the following two subsections introduce two test cases that are performed in order to: (1) verify the management requirements to make sure that the management scope that was discussed before in this chapter is covered by the prototype and (2) validate the functionalities of the prototype.

6.4.1 Prototype Testing Setups

I. Single Machine

In this setup, all the WS of the prototype are hosted by the same machine. Because the same machine hosts the functionalities of both MR and ACWS; this setup can be applied in the context of applying a layer of autonomic behaviour to ACWS, such that ACWS do not only provide AC capabilities to MR, but also provide such capabilities to themselves. In this setup, only one instance of the application server is required to be installed on this machine to host sensor, HWS, MWS, AWS, PWS, EWS, and effector. In addition to hosting the WS, the machine hosts the knowledge base of the OS, the private UDDI database and the symptoms database of the AWS.

II. Two Machines

In this setup, the ATP and the knowledge base of the OS are hosted by one machine and the HWS, MAPE-WS, the private UDDI database, and the symptoms database of the AWS are hosted by another machine. Therefore, both sensor and effector are hosted by the first machine and published into the second one. One instance of the application server is installed on each machine to host WS on that machine. The goal of implementing this setup is to emphasize the concept that BWS can be remotely and dynamically managed by ManagingWS.

6.4.2 Prototype Test Cases

Before performing any tests, all the WS must be published into the private UDDI; moreover, all the WS must be deployed to the application server(s). Each test case is performed by generating a JSP client to the sensor WS. By running this client, the AC infinite loop starts, and all the subsequent invocations are automatically generated.

The two following test cases were performed for each setup of the testing setups discussed above.

I. Elevating CPU Utilization

1. Test Case Description

As our prototype is concerned with healing the OS in the case of CPU over utilization, in order to validate this functionality of our prototype, the following tasks were performed as the prerequisites for executing this test case: (1) setting the threshold of the CPU utilization to 10 percent; although 10% utilization does not represent any practical meaning, it was selected in order to validate the capabilities of MWS in assessing the status of the OS in terms of the CPU utilization by executing normal applications, (2) making the *Abnormal CPU Utilization* the common symptom to AWS, and the associated directive is to analyze the active processes to find the process of the maximum CPU utilization and kill it to release the CPU utilization; this aims at validating the advantage of using symptoms database in accelerating the analysis process; an important note is that instead of killing processes, the priority of some processes can be set to be lower than the priority of critical processes in order to guarantee CPU cycles for critical processes, (3) defining all the system, eclipse and java processes as the processes that cannot be killed; this aims at validating the capabilities of PWS in order to validate recommended healing actions against the OS policies, and (4) associating a period of 5 seconds in order to kill any process; this aims at validating the capabilities of EWS in scheduling the healing commands.

2. Test Case Results

- a. When the CPU utilization was less than 10 percent, there was no problem and the active AC round was terminated, and the next round was successfully started as expected.
- b. When the CPU utilization exceeded 10 percent, MWS reported *Abnormal CPU Utilization*.
- c. The AWS was invoked by HWS; and then, AWS requested the active processes from the OS through HWS. Once the active processes have been retrieved, the analysis algorithm was activated to find the target process. When there was no process over utilizing the CPU, the AWS reported *No Problem Found* to HWS, and the active AC round was terminated, and the next round

was started. When there was a process over utilizing the CPU, AWS reported killing that process as a healing action.

d. The PWS was invoked by HWS, and the input to PWS from HWS was a process to kill as well as the type of the process (e.g. running under the supervision of the OS or running under the supervision of some application). When the reported process was one of the processes listed in the implementation assumptions that were mentioned in section 6.3.1 of this chapter, PWS reported a request for a new action to HWS, which invoked AWS to find another process; otherwise, PWS reported its approval regarding killing some process to HWS.

e. EWS was invoked by HWS when the latter received the approval from PWS on the action recommended by AWS; and then, EWS translated kill process into the *process -k* command, which is understandable by the process utility that is running in the OS environment. Since killing a process should be applied after 5 seconds, EWS reported the command together with the period, after which, the command should be executed to HWS.

f. HWS delivered the command to the effector which was successfully executed after 5 seconds.

g. In order to simulate CPU over utilization, CPU consuming applications, such as *Acrobat Reader* were manually started. The prototype succeeded in killing the *acrobat* process after it was manually started.

h. The reported messages by MAPE-WS to HWS are reported to the system console, while the result of each AC round is reported to the JSP client.

II. Stopping NSClient Service

1. Test Case Description

As our prototype is also concerned with healing the OS in the case of stopping some important service, and in order to validate this functionality of our prototype, the following tasks were performed as the prerequisites for executing this test case: (1) defining the Nagios Agent Service as one of the services that should be running; this aims at validating the capabilities of AWS of discovering the services that should be started, (2) Assuming that stopping the Nagios Agent Service (e.g. NSClient Service) is tracked by the system logs; this aims at enabling AWS to keep track of the event of stopping the service, (3) When the

HWS is unable to collect metrics for OS, it reports Metrics Could Not Be Collected, and (4) associating a period of 2 seconds in order to start any service; this aims at validating the capabilities of EWS in scheduling healing commands.

2. Test Case Results

- a. When the Nagios Agent Service was stopped, HWS was unable to collect metrics from the OS; therefore, HWS invoked AWS and reported the Metrics Could Not Be Collected message to it.
- b. When AWS was invoked, it checked the system log file and found that the service was stopped.
- c. AWS checked the services that should be running in the environment of the OS and found that the stopped service is one of those services.
- d. AWS reported that the service should be started.
- e. In our prototype, no planning actions are taken in this test case; therefore, HWS delivered the AWS response to EWS.
- f. EWS translated start service into the `net -start` command, which is understandable by the OS native APIs.
- g. Since starting a service should be applied after 2 seconds, EWS reported the command together with the period, after which, the command should be executed to HWS.
- h. HWS applied the command to the effector which was successfully executed after 2 seconds.
- i. The prototype succeeded in starting the service after it was stopped and HWS was able to collect metrics from the OS.
- j. The reported messages by MAPE-WS to HWS are reported to the system console, while the result of each AC round is reported to the JSP client.

6.4.3 Prototype Testing Screen Shots

I. Sensor Web Service Web Interface

From this Interface, the AC infinite loop is started by invoking the highlighted method in the figure below:

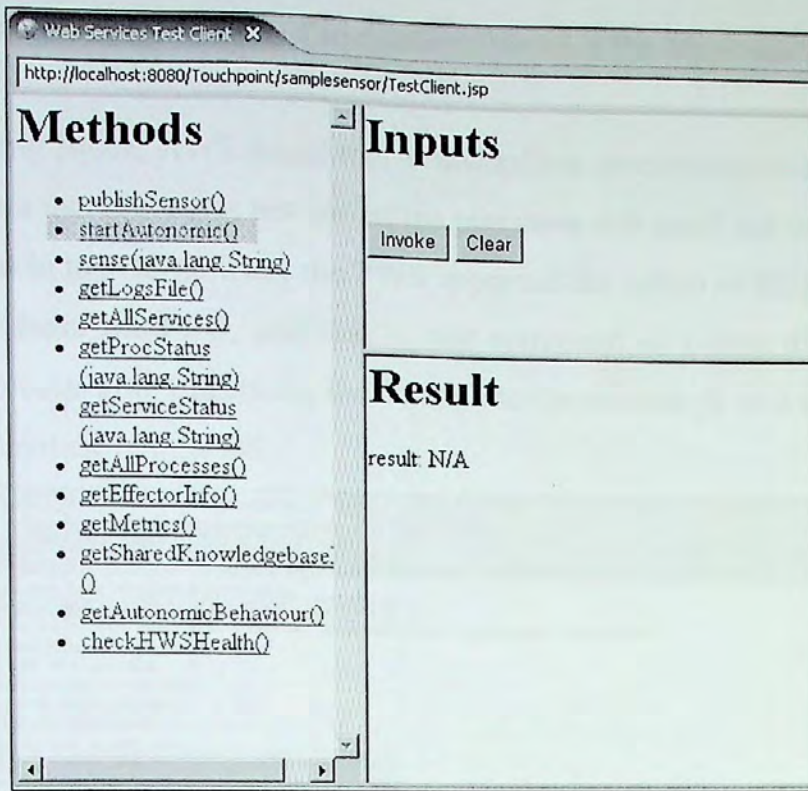


Figure 34 Sensor Web Service Web Interface

II. No Problem Was Found at the End of an Autonomic Computing Round

In the figure below, although CPU Utilization was found 23% which is greater than the 10% CPU utilization threshold, AWS found no problem, and the AC round was terminated.

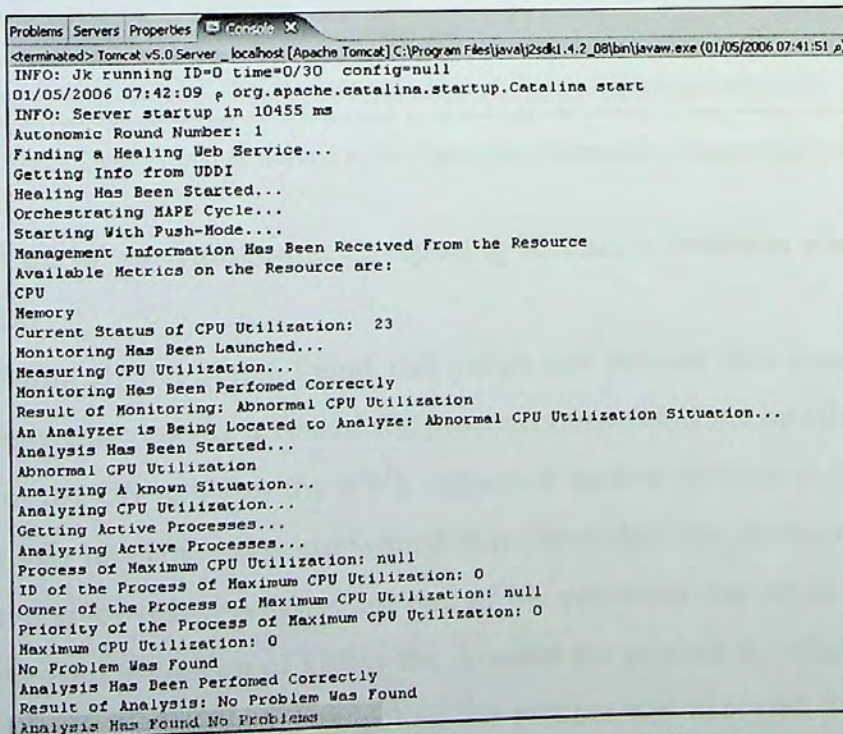


Figure 35 No Problem Found at the End of an Autonomic Computing Round

III. At the End of an Autonomic Computing Round, a Problem was Found and Solved (1)

In the following figure, AWS found that Winword.exe process was over-utilizing the CPU, and since that process is not one of the processes that could not be killed, which are mentioned in the OS policies, the PWS approved the action of killing the process in order to release the CPU; and finally, the command of killing the process was executed 5 seconds after translating the action into the process -k as it is mentioned in the execution policies of the OS.

```

Web Services Test Client
Problems Servers Properties
<terminated> Tomcat v5.0 Server localhost [Apache Tomcat] C:\Program Files\Java\j2sdk1.4.2_08\bin\javaw.exe (01/05/2006 08:02:01 p)
Monitoring Has Been Performed Correctly
Result of Monitoring: Abnormal CPU Utilization
An Analyzer is Being Located to Analyze: Abnormal CPU Utilization Situation...
Analysis Has Been Started...
Abnormal CPU Utilization
Analyzing A known Situation...
Analyzing CPU Utilization...
Getting Active Processes...
Analyzing Active Processes...
Process of Maximum CPU Utilization: WINWORD.EXE
ID of the Process of Maximum CPU Utilization: 852
Owner of the Process of Maximum CPU Utilization: sherif
Priority of the Process of Maximum CPU Utilization: 8
Maximum CPU Utilization: 30
Status Can Be Analyzed
Kill Process WINWORD.EXE 852 sherif 8 30
Analysis Has Been Performed Correctly
Result of Analysis: Kill Process WINWORD.EXE 852 sherif 8 30
A Planner is Being Located to Plan The Execution of: Kill Process WINWORD.EXE Solution
suggestedAction: Kill Process WINWORD.EXE 852 sherif 8 30
Planning Has Been Started...
Planning Has Been Performed Correctly
Result of Planning: Kill Process WINWORD.EXE
The Suggested Kill Process WINWORD.EXE Action by AWS Will Be Executed...
An Executor is Being Located to Execute The Action
Action Has Been Passed to EUS
Execution Has Been Started...
Command: process -k
Schedule: 5
process -k WINWORD.EXE Is Being Dispatched...
Actual Command Being Executed: process -k WINWORD.EXE
Command: process -k WINWORD.EXE Will Be Executed After: 5 Seconds
Execution Has Been Performed Correctly
process -k WINWORD.EXE Has Been Executed Successfully...Resource Status Has Been Healed...

```

Figure 36 A Problem was Found & Solved at the End of an Autonomic Computing Round: Scenario 1

IV. At the End of an Autonomic Computing Round, a Problem was Found and Solved (2)

In the following figure, AWS found that javaw.exe process was over-utilizing the CPU, and since that process is one of the processes that could not be killed, which are mentioned in the OS policies, the PWS requested another process to kill; therefore, another analysis procedure was performed that concluded that Acrobat.exe was also over-utilizing; since that process is not one of the processes that could not be killed, the PWS approved the action of killing the Acrobat.exe process in order to release the CPU; and finally, the command of killing the process was executed 5 seconds after

translating the action into the process -k as it is mentioned in the execution policies of the OS.

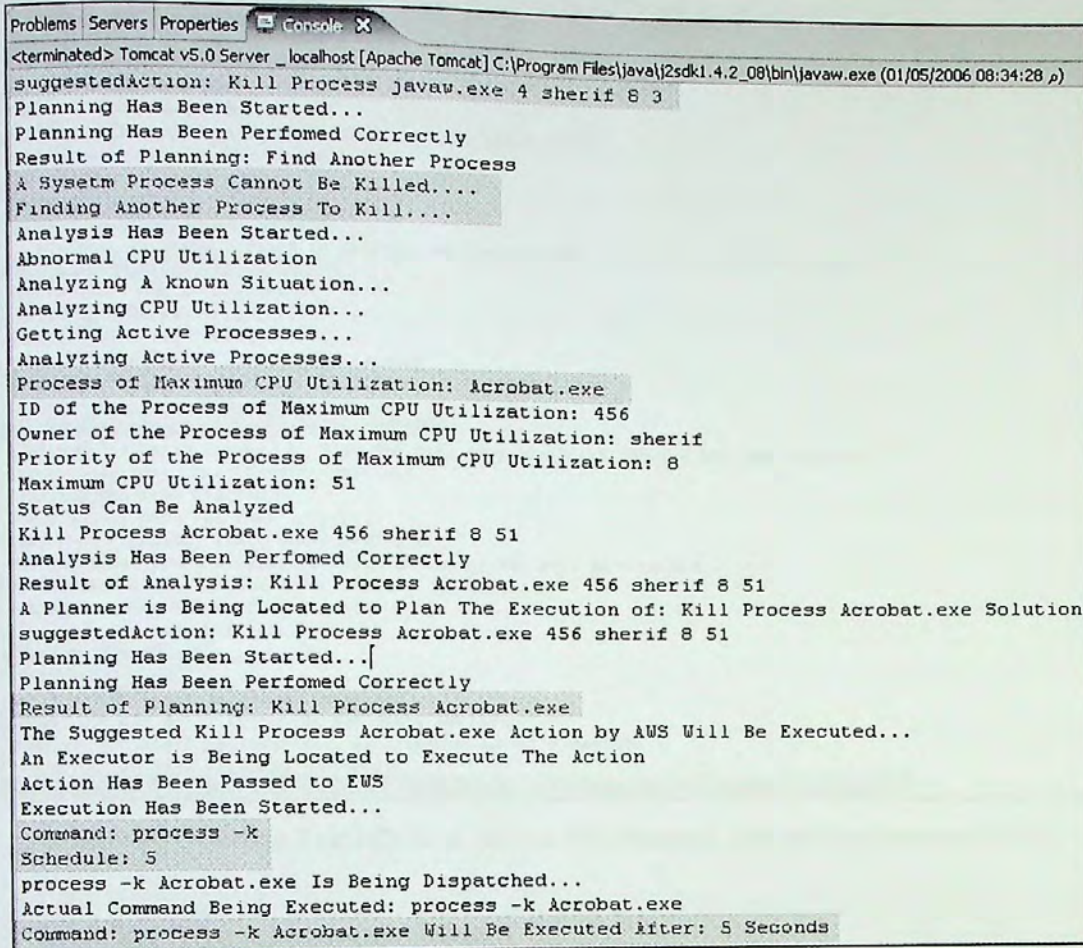


Figure 37 A Problem Was Found & Solved at the End of an Autonomic Computing Round: Scenario 2

V. The Impact of Closing the NSClient Service

1. Before The Autonomic Round

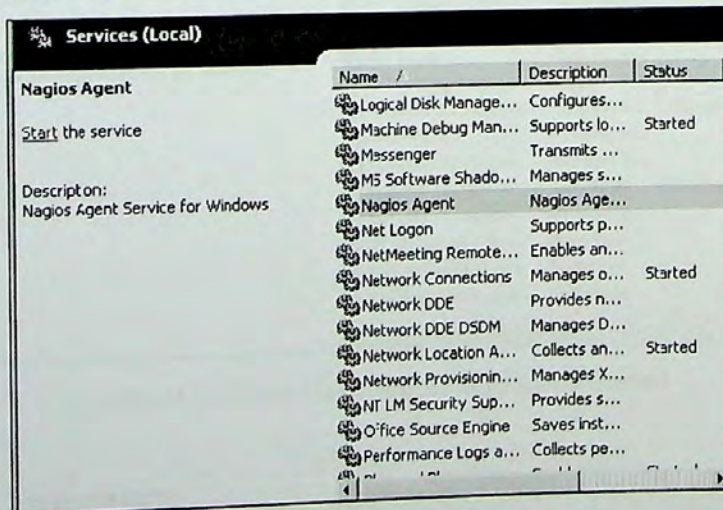


Figure 38 Stopping the NSClient Service

2. During The Autonomic Round

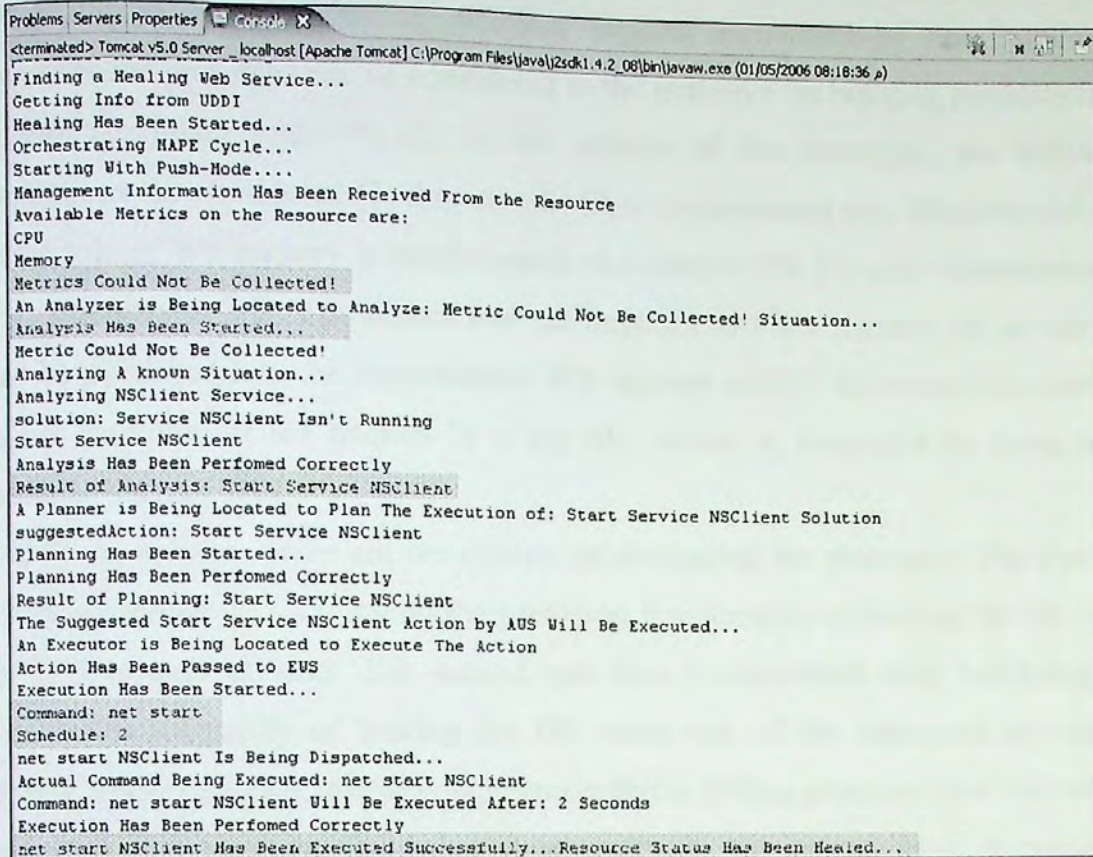


Figure 39 Detecting That NSClient Service Was Stopped, and starting it automatically

3. After The Autonomic Round

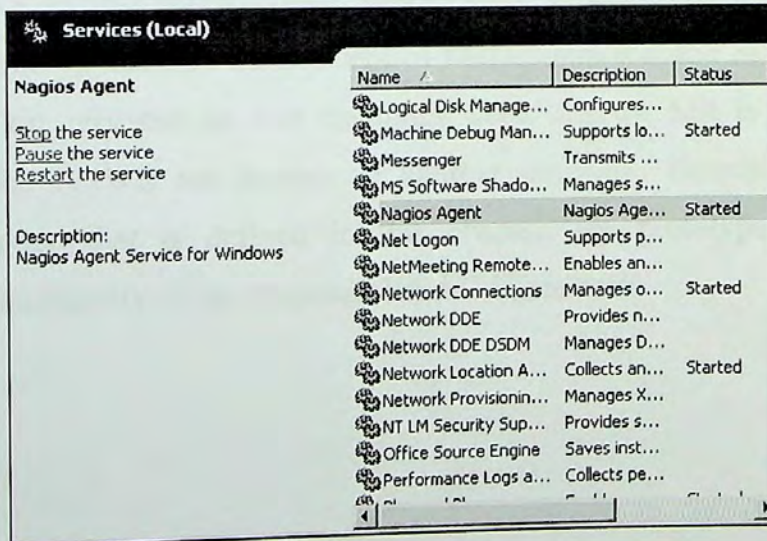


Figure 40 NSClient Has Been Automatically Started

6.5 Chapter Summary

In this chapter, the reasonableness of the proposed WSAC framework was evaluated with the aid of a proof-of-concept prototype that implements both the architectural

and the processing models of WSAC, which were discussed in the previous chapter. The prototype helps evaluating different aspects introduced by the framework. Therefore, the prototype can be considered as the initiative for building production CS that are compliant with WSAC. In the context of the prototype, the following assumptions are considered (1) the role of MR is implemented as a Windows XP OS, (2) the role of WS registry is implemented as a simple DB, (3) only communication information about WS is published into the implemented WS registry, (4) all the WS must be published into the implemented WS registry and (5) all events that occur in the OS environment are tracked by a log file, which is, formatted by using some standards, such as WEF.

In addition, two test cases are considered for evaluating the prototype. The first test case is concerned with validating the prototype functionality of healing the OS when the CPU is over utilized. The second test case is concerned with validating the prototype functionality of healing the OS when one of the important services is stopped. In the first case, the prototype succeeded in killing processes that over utilize CPU; and after that, the OS behaved normally. In the second case, stopping the NSClient service made the HWS unable to collect metrics from the OS, the prototype succeeded in detecting the stopped service and in starting it, such that after starting the service, HWS became able to collect metrics from the OS.

Moreover, in order to validate the concept of ManageableWS and ManagingWS the two test cases are executed on two machines, such that the MR is hosted on one machine and the ACWS are hosted on another machine. Therefore, as per the management scope that is defined in this chapter, the prototype succeeded in evaluating the practicality of the proposed WSAC framework.

Chapter 7: Conclusion

Chapter 7: Conclusion and Future Work

7.1 Chapter Overview

This chapter summarizes the conducted research, sheds light on the achieved milestones, highlights the contributions, discusses the challenges and conflicts that are not resolved yet, and finally, suggests possible future work.

7.2 Research Summary

Our research is motivated by two motives; the first was recognized after studying the paradigm of AC, or self-management. Heterogeneity is an inherited characteristic in any modern CS, where such systems accommodate a variety of computing resources that differ in platforms, vendors, protocols, etc. Therefore, in order to adopt the notion of AC by modern CS, an AM will be implemented to provide autonomic behavior to each group of computing resources. For that reason, and after reviewing the challenges found in the context of AC, heterogeneity was realized as one of the challenges that hinder the capabilities of AM; and therefore, hampers the goal of building ACS that can be employed in modern CS. Consequently, the first motive of doing this research is to find a solution that addresses the problem of heterogeneity in the context of AC in order to enable modern CS to evolve to be self-managed without requiring such systems to implement radical changes in their IT infrastructures.

The second motive was realized after studying the paradigm of WS. The characteristics of WS encourage the development of a new class of software applications, in which, WS communicate with each other dynamically, and in a loosely-coupled manner. Therefore, managing such applications through human-intervention is expected to be inefficient, as changes that occur dynamically cannot be tracked by human as well as the human's responsiveness to fix problems when they occur will not be fast enough to take the required recovery actions at proper times. For that reason, automating management of WS was recognized as a key requirement for achieving the expected business benefits from WS. Consequently, the second motive of doing this research is to find a solution that addresses the issue of automating the management of WS while accommodating the expected level of dynamism in the lifecycles of WS applications.

By combining the two motives that are behind our research, the goal of our research was identified as a two-directional goal, such that the first direction is concerned with the issue of heterogeneity in the context of AC, and the second direction is concerned with the issue of automating management in the context of WS. Moreover, by considering the promises of both the AC and WS paradigms, the significance of each paradigm to the other became clear, or in other words, the significance of WS to the first direction of our research's goal and the significance of AC to the second direction of this research's goal. First, WS technologies can be used in building ACS, such that computing resources are instrumented with the aid of WS interfaces and AM are implemented as ManagingWS that provide AC capabilities to computing resources. The significance of this approach can be highlighted as follows: (1) AM can publish their interfaces into a public service registry so that computing resources, which are referred to as MR, can discover AM at run time, (2) AM can provide AC capabilities to MR regardless of the underlying details of MR and (3) AC capabilities are not required to be implemented by ACS as such capabilities can be outsourced at runtime.

Second, the notion of AC can be applied to WS, such that a WS can establish and maintain a relationship with some ManagingWS in order to be monitored and managed dynamically. The significance of this approach can be highlighted as follows: (1) management overhead is decoupled from WS and (2) management of WS is automated by allowing WS to dynamically find and talk with ManagingWS. An interesting finding is that accomplishing the first direction of the research's goal automatically leads to accomplishing the second one. This is because when AM are architected as WS in the form of ManagingWS, it will be a trivial task to enable BWS to dynamically discover those ManagingWS to start a management relationship, through which, ManagingWS automatically monitor and manage BWS.

In order to achieve the goal of our research, we proposed a framework called WSAC. In WSAC, the major aspects of AC are called ACWS, which are implemented as the following set of WS: (1) CWS, which is an AM that is responsible for the self-configuring aspect, (2) HWS, which is an AM that is responsible for the self-healing aspect, (3) OWS, which is responsible for the self-optimizing aspect and (4) SWS, which is responsible for the self-securing aspect. Moreover, a typical MAPE cycle is implemented as a set of WS that are: (1) MWS, which implements the monitor role of

the MAPE cycle, (2) AWS, which implements the analyzer role of the MAPE cycle, (3) PWS, which implements the planner role of the MAPE cycle and (4) EWS, which implements the executor role of the MAPE cycle. MWS, AWS, PWS, and EWS are called MAPE-WS. In WSAC, MAPE-WS are either defined to ACWS statically or discovered by ACWS dynamically, such that in the second case an ACWS will compose the MAPE cycle after discovering the MAPE-WS.

Moreover, in WSAC, MR are wrapped by WS interfaces that are called ATP, such that an ATP is composed of a set of sensors and an effectors. Through sensors, the MR is instrumented to allow ACWS to collect information about its status; moreover, through its sensors, an MR can discover the available ACWS and establish management relationship with one of them. Through its effectors, an MR is managed by accepting management commands. In the context of our research, the focus will be on the self-healing AC aspect; and therefore, the focus will be on HWS.

In order to evaluate the proposed WSAC framework, a proof-of-concept prototype that implements both the architectural and processing models of the framework was built. The prototype is composed of a set of roles that are MR, ATP (a sensor and an effector), HWS, MWS, AWS, PWS, EWS, knowledge base of MR, symptoms database of AWS and service registry. For the sake of simplicity, the prototype's components are implemented in the form of: (1) a Microsoft Windows XP OS acting as the MR, (2) two WS playing the roles of the sensor and the effector, (3) a WS acting as the HWS, (4) a WS acting as the MWS, (5) a WS acting as the AWS, (6) a WS acting as the PWS, (7) a WS acting as the EWS, (8) a Microsoft Office Access Database implementing the knowledge base of the MR, (9) another Microsoft Office Access Database representing the symptoms database of the AWS and (10) a Microsoft Office Access Database acting as the WS service registry.

In addition, the management scope of the prototype is limited to: (1) monitoring the CPU utilization of the OS, (2) detecting CPU over-utilization situations, (3) killing processes that over utilize the CPU, (4) identifying critical processes that cannot be killed (policy-based management), (4) detecting the situations, in which, important services are stopped, (5) starting important services that have been stopped, (6) analyzing system logs to identify situations that are not common and (7) scheduling the execution of healing commands.

After building the prototype, two test cases were executed to validate the functionality of the prototype to kill non-system processes and to start important services after stopping them. Moreover, the two test cases were executed one time on a single machine and another time on two machines. In both cases, the prototype succeeded in achieving the expected goals.

7.3 Contributions

This section summarizes the contributions of our thesis to the context of the research.

1. Loosely-Coupled Autonomic Computing

In the context of WSAC, autonomic behaviour can be provided by ACWS to MR dynamically without requiring any static binding. This is because MR only need to know about the interfaces of ACWS, through which, MR can invoke methods on ACWS, and ACWS only need to know about the ATP of the MR, through which, information can be retrieved and actions can be applied.

2. Autonomic Computing Web Services

In the WSAC framework that we propose in this thesis, the concept of ACWS was introduced. Since each ACWS is developed separately, new techniques can be easily accommodated as well as related capabilities can be outsourced from similar ACWS. For example, an HWS can outsource some advanced healing capabilities from other HWS. Moreover, by decoupling ACWS from MR, the latter will be capable of finding new ACWS if the currently located one(s) failed for any reason. For example, an MR will be able to locate another HWS to outsource the healing AC aspect if the currently located HWS failed to respond to MR requests.

3. MAPE Web Services

The concept of MAPE-WS was also introduced by the WSAC framework. By breaking the MAPE cycle into four integrated MAPE-WS, each MAPE-WS can be incrementally developed. This has the advantage of easily adopting new techniques as they evolve in the fields of monitoring, analyzing, planning, and executing; moreover, capabilities can be added, deleted, and/or changed according to the changes in business requirements. In addition, each MAPE-WS can outsource related capabilities from similar MAPE-WS; for example, an MWS can outsource some advanced monitoring capabilities from other MWS. Additionally, by decoupling MAPE cycles from ACWS, the latter will be capable of finding new MAPE-WS if the currently

located one(s) failed for any reason. For example, an HWS will be able to locate another AWS to complete the MAPE cycle if the currently located AWS failed to respond to HWS requests.

4. Hybrid Instrumentation

In the context of our WSAC framework, *hybrid instrumentation* is the technique, by which, ACWS can know about the status of MR. This technique can be explained as follows: by enabling ACWS to poll the required information from MR if the latter does not push the expected information within a specific period of time, ACWS can automatically discover failures occurring in MR without relying on any notifications from MR.

5. Upgrading Analytical Skills

Although adding learning skills to AWS will be covered in the future work section of this chapter, the proposed processing model of WSAC framework in this thesis enables AWS to continuously update its symptoms databases by adding new symptoms and directives recognized from analyzing logs; and therefore, analytical skills of AWS are always upgraded.

6. Evolving Legacy Computing Resources

By using ATP to wrap legacy computing resources, such as operating systems, by WS interfaces, as it was mentioned in the architectural model of the WSAC framework, the notion of AC can be adopted by such resources without the need for replacing or redeveloping them to support AC characteristics.

7. Splitted Channels

In the context of WSAC framework, WS can be generally classified into WS that provide BC (BWS) and WS that provide management capabilities (ManagingWS). Therefore, the communication channel of any BWS can be logically splitted into two channels *business channel* and *management channel*. Through the business channel, a BWS can conduct business with other BWS through BI; and through the management channel, the BWS can be managed by ManagingWS through MI. As a result, the management overhead is decoupled from BWS in order to allow the latter to focus on delivering the required business. Figure 41 illustrates the concept of splitted channels, which provides an abstract view of the WSAC framework.

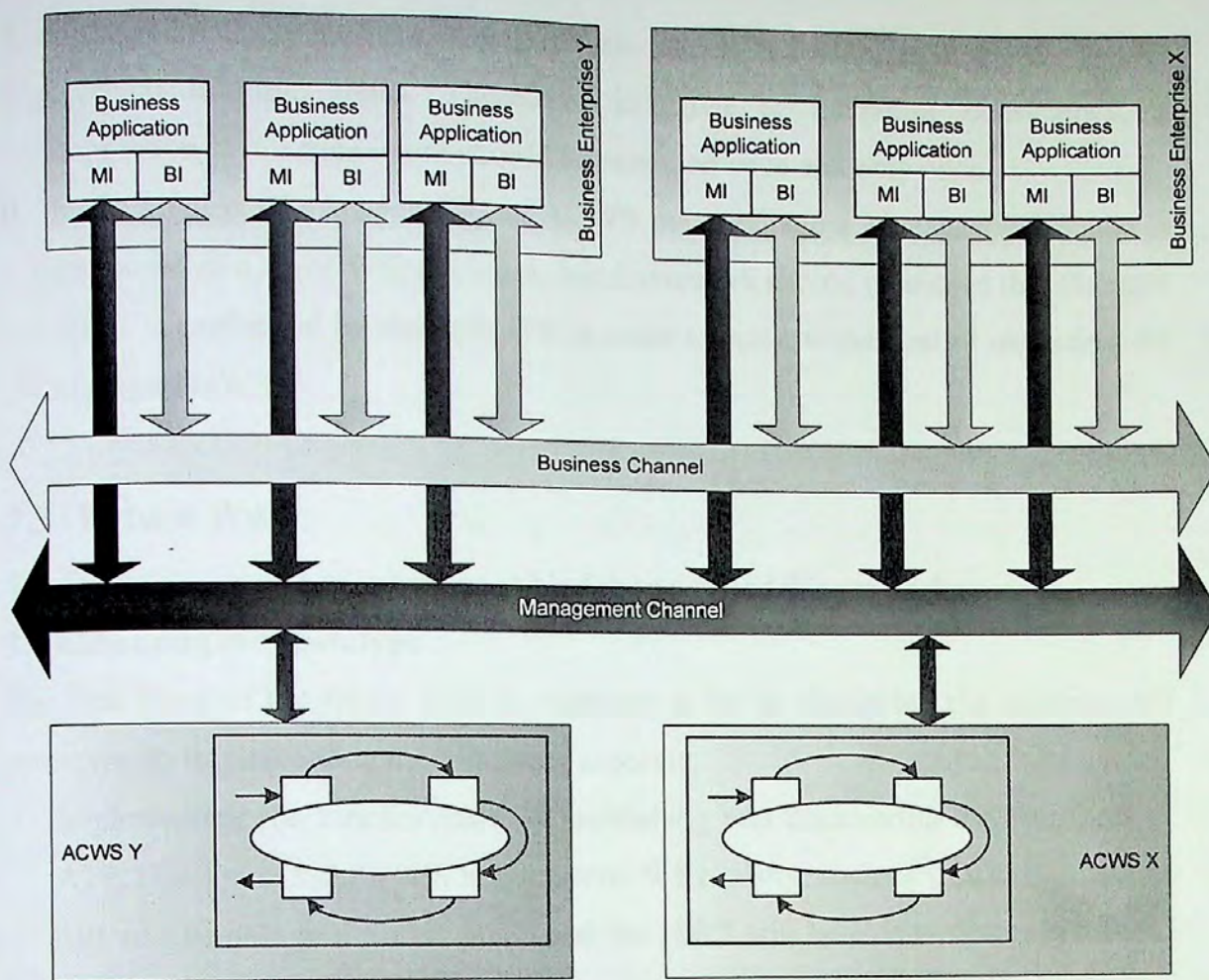


Figure 41 Split Business and Management Channels

7.4 Challenges and Non-Resolved Conflicts

In this section, the challenges and conflicts that are still open in the WSAC framework are listed:

- I. Managing the degree of coupling between the components affected by the healing. For example, healing the OS might have a negative impact on the running applications.
- II. Managing the situation, in which an MR fails to locate the required ACWS.
- III. Managing the situation, in which an HWS fails to locate the required MAPE-WS.
- IV. Managing the situation, in which any MAPE-WS cannot process the submitted request from some ACWS.
- V. Adding an AC layer to provide AC capabilities to ACWS and MAPE-WS to guarantee their performance while providing AC capabilities to MR.

- VI. Because of some failures, some request within a certain transaction may be received multiple times. Therefore, in order to guarantee consistency, a transaction in our framework should be executed once and only once.
- VII. In the context of management, an ACWS may manage a ManageableWS that is composed of a set of WS; therefore, our framework should guarantee that the right action is performed by the right WS in order to achieve the goal of managing the ManageableWS.

7.5 Future Work

The following points summarize possible future work of this research.

1. Enhancing the Prototype

The first focus of the future work is expected to be on enhancing the implemented prototype by implementing the following aspects:

- I. Implementing the functionalities of publishing and discovering the interfaces of ATP, HWS, and MAPE-WS into a public WS registry, such as UDDI. So that, the MR will be able to discover HWS and the HWS will be able to discover MAPE-WS at runtime.
- II. Performing some statistical analysis for the impact of the prototype on the CPU utilization as well as for the delay associated with the processing of AC requests.
- III. In order to permit discovery of MC supported by MR, metadata can be used to describe different properties, supported management mechanisms, and available information in the knowledge base to enable ACWS to perform some resonance about the MC of MR. Moreover, metadata can also describe the relationship between an MR and other MR in order to enable ACWS to track the dependencies between MR being managed and other MR. This can be illustrated with the aid of the following figure, which is a revisited version of figure 27 that described the architectural model of the WSAC framework in section 5.6.1 of chapter 5.

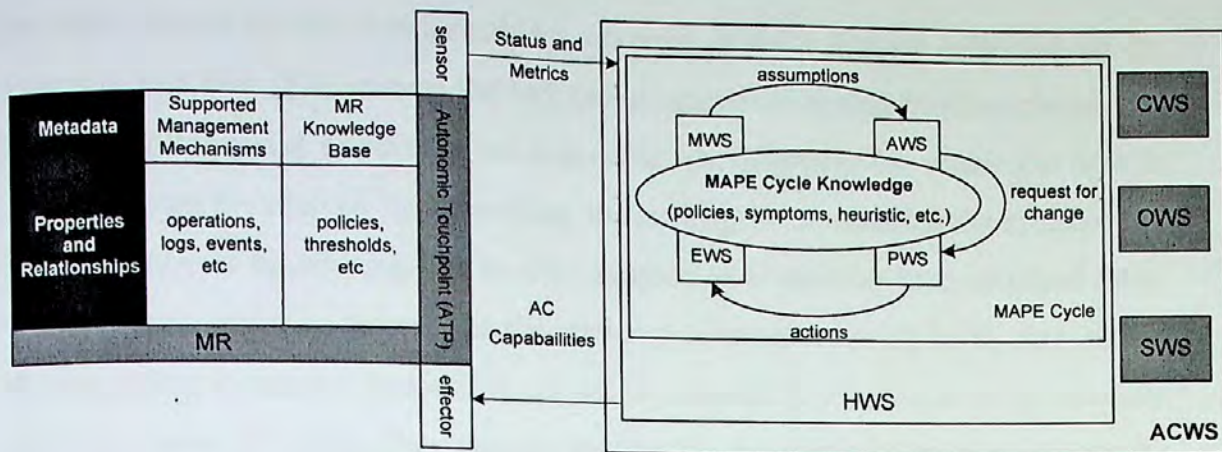


Figure 42 Enhanced WSAC Architectural Model

2. Expanding the Management Scope of the Prototype

- I. Considering more situations in the context of managing OS, such as managing the networking facilities, managing the disk spaces, etc.
- II. Using another MR, such as an application server, and implementing its ATP. So that, HWS can heal the application server when it behaves badly (e.g. the server hangs up).
- III. Implementing a simple BWS as another MR and equipping it with an ATP to enable the HWS to heal it when it behaves badly (e.g. slow response to submitted request).
- IV. Performing some statistical analysis for the delay associated with the processing of AC requests.

3. Improving the Capabilities of MAPE-WS

I. Monitoring Web Service

Adaptive management is concerned with tracking management actions in order to identify the actions that need to be adapted so that response to future changes increases. The cornerstone of adaptive management is the monitoring that provides the essential feedback that is used for evaluating management actions; therefore, monitoring must be also adaptive. The discussion of the adaptive monitoring that was mentioned in [51] from the viewpoint of ecological systems can be applied to the context of MWS. For example, an adaptive monitoring technique should be able to identify the proper monitoring intensity level or action, when it should be applied, and, in which, component it should be applied. In addition, changing monitoring

priorities should be also considered; for example, metrics that are collected can be changed over time. The state of the MR being monitored should be characterized by some indicators; and depending on that state, the intensity by which the MR is monitored can be adapted by increasing, decreasing, or terminating the monitoring; the frequency of monitoring can be also adapted as a reaction for a changed state. The level of variability determines the intensity of the monitoring level, such that if the variability increases, monitoring should be adapted to a higher level of intensity and vice versa. Changing the intensity and/or the frequency of monitoring can be guided by a set of thresholds defined by the MR; moreover, defined thresholds may be also adapted to fulfill new requirements.

II. Analyzing Web Service

Although the current version of our prototype includes the feature of upgrading the symptoms database of AWS by adding new symptoms and directives concluded from analyzing logs, this feature is implemented for the sake of introducing the capabilities of reasoning and learning that can be adopted by AWS. Therefore, *Artificial Intelligence (AI)* approaches can be used to implement those features to provide more advanced analyzing capabilities. Moreover, as it was suggested in [52], AWS may implement a set of specialized analysis processes that are invoked based on the semantics of failure. Moreover, as it was mentioned in [42], analyzing will not be accurate unless it considers the whole view including other resources, such as networking devices; therefore, AWS should have a global vision of the MR, whose status is being managed.

III. Planning Web Service

Planning capabilities can be permitted by allowing PWS to trace dependencies between the MR being managed and other components in the environment of MR. For example, starting a service might require starting some other services at the beginning. Therefore tracing dependencies will enable the PWS to achieve its goals while guaranteeing that applying some management action on the MR that was approved by the PWS will not have some negative impact on related computing resources.

4. Adding New Features to the Prototype

- I. Implementing a simple event-notification technique to enable both ATP to emit events as they occur and HWS to receive notifications of the significant events as they occur.
- II. Adopting the WSDM specifications in implementing ATP and HWS.
- III. Implementing an adapter for formatting the system logs based on the WEF standard to permit the root-cause analysis performed by AWS.
- IV. Using XML to format different information contained in the knowledge base of MR to enable HWS and MAPE-WS to understand that information.
- V. The autonomic WS selection approach that was proposed in [53] can be adopted in the context of our WSAC framework. For example, establishing management relationships between MR and ACWS from one side and ACWS and MAPE-WS from the other side can be guided by the reputation of each entity. This can be achieved by extending the record of ACWS and MAPE-WS that is registered into the UDDI registry by associating a reputation parameter with each record. For example, at the end of some management relationship between an MR and some ACWS, the MR accesses the UDDI and update the reputation of that ACWS; and similarly, at the end of some management relationship between an ACWS and some MAPE-WS, the ACWS accesses the UDDI and update the reputation of that MAPE-WS. Reputation of either ACWS or MAPE-WS can be assessed based on several criteria, such as response time, availability, effectiveness in solving problems, etc.
- VI. Establishing management relationships between MR and ACWS from one side and ACWS and MAPE-WS from the other side can be conducted with the aid of a negotiation process, through which, involved parties agree on the expected output from each party. Moreover, during the lifetime of a management relationship, the agreed on level of service can be monitored and when the management relationship is terminated, the result of monitoring can be used to update the reputation of each involved WS.

5. Accommodating Other Autonomic Computing Characteristics

- I. Extending the prototype to support the remaining AC characteristics by implementing other ACWS than HWS that are CWS, OWS, and SWS.

- II. Extending the prototype by adding capabilities to MAPE-WS to support the contexts of configuring, optimizing, and securing.

6. Security Considerations

Trust is a key characteristic in order to enable ACWS to execute actions on MR to control the behavior of the latter. Therefore, there should be some trust model to enforce security policies regarding the relationships between ACWS and MR. That trust model should consider issues, such as how parties are identified and authenticated, how policies and tokens are formatted and exchanged [9]. Such issues are currently addressed by many standards, such as *Extensible Access Control Markup Language (XACML)*, *Extensible Name Service (XNS)*, *Authentication, Authorization, and Accounting (AAA)*, etc. In particular, *Web Service Security Architecture (WSSA)* can be applied in the context of our framework. For example, *SOAP* and *WS-Security* can be used to provide secure communication channels, *WS-Policy* and *WS-Authorization* can be used to provide trusted services, and *WS-Trust* and *WS-Federation* can be used to secure management.

List of References

- (1) J. O. Kephart and D. M. Chess, "The vision of autonomic computing", IEEE Computer, 36(1):41-50, http://www-03.ibm.com/autonomic/pdfs/AC_Vision_Computer_Jan_2003.pdf, 2003.
- (2) Grid Technology Partners, "The Autonomic Computing Report: Characteristics of Self Managing IT Systems", <http://www.gridpartners.com/toc-ac.html>, 2003.
- (3) "Autonomic computing architecture: a blueprint for managing complex computing environments", IBM articles, <http://www.iids.org/publications/saacs05.pdf>, October 2002.
- (4) "The Web Services Management Platform: Managing the impact of Change in an Enterprise Web Services Network", Actional, http://www.webservices.org/index.php/categories/management/the_web_services_management_platform_managing_the_impact_of_change_in_an_enterprise_web_services_network#, March 2003.
- (5) "Web Services: A Practical Introduction to SOAP Web Services", Systinet, http://www.bitpipe.com/data/detail?id=1032958105_530&type=RES, 2003
- (6) "Autonomic Computing Manageable Resource Interface Specification and Touchpoint Implementation Guide", IBM, <http://www.alphaworks.ibm.com>, Oct. 2005.
- (7) J.A. Farrell and H. Kreger, "Web Services Management Approaches", IBM SYSTEMS, VOL. 41, NO 2, <http://www.research.ibm.com/journal/sj/412/farrell.html>, 2002.
- (8) "Autonomic Computing: Enabling Self-Managing Solutions", an IBM Autonomic Computing White Paper, http://www-03.ibm.com/autonomic/pdfs/SOA_and_Autonomic_Computing.pdf, Dec. 2005.
- (9) "Managing On Demand With a Service-Oriented Management Architecture: Virtualizing the Management of Virtual Systems", a META Group White Paper, http://www.ca.com/de/inf/mod/managing_on_demand_with_soa.pdf, Oct. 2004.
- (10) Heather Kreger and James Phillips, "Toward Web Service Management Standards-An architectural approach to IT system design"; <http://www.syscon.com/webservices>, Web Services Journal.

- (11) Toral Mehta, "Adaptive Web Services Management Solutions", Enterprise Networks and Services, Vol. 17 No. 5, May 2003.
- (12) Heather Kreger et al., "Management Using Web Services: A Proposed Architecture and Roadmap", IBM, hp and ca, <http://www.presentationselect.com/hpinvent/archive.asp?eventid=977>, June 2005.
- (13) "A Non-Invasive Approach to Managing Web Services", An AmberPoint Technology Brief, <http://www.developers.net/node/view/1003?partner=cisco%20skillsoft&ref=new?partner=cisco%20skillsoft>, June 2002.
- (14) Jereon van Sloten et al., "On the Standardization of Web Services Management Operations", Eunice, <http://www.simpleweb.org/nm/research/results/presentations/pras/2004-06-EUNICE-WS-standardisation.pdf>, 2004.
- (15) P. Horn, "Autonomic computing: IBM's perspective on the state of information technology", IBM articles, <http://www-03.ibm.com/industries/government/doc/content/resource/thought/278606109.html>, 2001.
- (16) K. Ekanadham et al., "Anatomy of Autonomic Server Components", Research Report RC 22637, IBM T. J. Watson Research Center, Yorktown Heights, NY., <http://www.research.ibm.com/journal/sj/421/jannref.html>.
- (17) Mazeiar Salehie and Ladan Tahvildari, "Autonomic Computing: Emerging Trends and Open Problems", DEAS 2005, ICSE 2005, <http://www.deas2005.cs.uvic.ca/deas-2005-procs-final.pdf>, May 2005.
- (18) "Autonomic computing, the 8 elements", IBM, <http://www.research.ibm.com/autonomic/overview>.
- (19) Heather Kreger and Thomas Studwell, "Autonomic Computing and Web Services Distributed Management", IBM articles, <http://www-128.ibm.com/developerworks/autonomic/library/ac-architect/>, June 2005.
- (20) Brent A. Miller, "The autonomic computing edge: Keeping in touch with touchpoints", IBM articles, <http://www-128.ibm.com/developerworks/autonomic/library/ac-edge5/>, Aug. 2005.

- (21) Daniel H. Steinberg, "What you need to know now about autonomic computing, Part 4: Deploying and logging"; <http://www-136.ibm.com/developerworks/>, August 2003.
- (22) Dennis Smith et al., "Interoperability Issues Affecting Autonomic Computing", DEAS 2005, ICSE 2005, <http://www.deas2005.cs.uvic.ca/deas-2005-procs-final.pdf>, May 2005.
- (23) IBM Web Services Architecture Team: "Web Services architecture overview: The next stage of evolution for e-business", <http://www-128.ibm.com/developerworks/>, Sept. 2000.
- (24) "Web Service Management", TekPlus, <http://www.tekplus.com/TP0129R03V01-Web%20Services%20Management-final.pdf>, 2003.
- (25) Jim Webber and Savas Parastatidis, "Demystifying Service-Oriented Architecture", <http://www.sys-con.com/webservices/>, Web Services Journal.
- (26) Whit Andrew et al., "Web Services Management: Making the Enterprise Ready", Gartner Symposium ITXPO, <http://symposium.gartner.com/story.php.id.3144.s.5.html>, 23-27 March 2003.
- (27) Sven Graupner and Tilo Nitzsche, "Web Services Management for Adaptive Control", OST&T, HP Labs / SGBU MSO, <http://uk.builder.com/whitepapers/0,39026692,60099757p-39000988q,00.htm>, May 2004.
- (28) Mike Papazoglou and Willem-Jan van den Heuvel, Tilburg University, "Web Services Management: A Survey", IEEE Internet Computing, <http://infolab.uvt.nl/pub/papazogloump-2005-86.pdf>, Dec. 2005.
- (29) Judith Hurwitz, "How to Choose a Web Services Management Solution", Actional, http://www.actional.com/products/docs/how_to_choose_a_web_services_management_solution.pdf, April 2004.
- (30) "Practical Considerations for Implementing Web Services: The Role of Web Services Management", AmberPoint, <http://www.smcm.edu/~dnemerick/com520/downloads/webservices.pdf>, Jan. 2004.
- (31) Mark Potts et al., "Web Services Manageability – Concepts (WS-Manageability)",

http://www3.ca.com/Files/SupportingPieces/web_service_manageability_concepts.pdf, Sept. 2003.

(32) William Vambenepe, "Web Service Distributed Management: Management Using Web Services (MUWS 1.0) Part1", OASIS Standard, <http://docs.oasis-open.org/wsdm/2004/12/muws/cd-wsdm-muws-part1-1.0.pdf>, March 2005.

(33) Mark Potts et al., "Web Services Manageability – Specifications (WS-Manageability)", <http://www-128.ibm.com/developerworks/library/specification/ws-manage/>, Sept. 2003.

(34) Nicolas Catania et al., "Web Services Management Framework – overview (WSMF – Overview) Version 2.0", HP, <http://devresource.hp.com/drc/specifications/wsmf/WSMF-Overview.jsp>, July 2003.

(35) William Vambenepe, "Web Service Distributed Management: Management Using Web Services (MUWS 0.5)", Committee Draft, <http://docs.oasis-open.org/wsdm/2004/04/muws-0.5/cd-wsdm-muws-0.5.pdf>, April 2004.

(36) William Vambenepe, "Web Service Distributed Management: Management Using Web Services (MUWS 1.0) Part2", Committee Draft, <http://docs.oasis-open.org/wsdm/2004/12/muws/cd-wsdm-muws-part2-1.0.pdf>, December 2005.

(37) Kunal Verma and Amit P. Sheth, "Autonomic Web Processes", Proceedings of the Third International Conference of Service-Oriented Computing (ICSOC), http://lstdis.cs.uga.edu/library/download/KS_05-ICSOC.pdf, 2005.

(38) Ken Birman et al., "Adding High Availability and Autonomic Behavior to Web Services", Proceedings of the IEEE 26th International Conference of Software Engineering (ICSE'04), <http://portal.acm.org/citation.cfm?id=998675.999394>, 2004.

(39) Alberto Bartoli et al., "Adapt: Towards Autonomic Web Services", <http://adapt.ls.fi.upm.es/adapt.htm>, 2001.

(40) Wenhui Tian et al., "Architecture for an Autonomic Web Services Environment", School of Computing, Queen's University, <http://www.cs.queensu.ca/home/cords/wsmdeis.pdf>.

(41) Igor Sedukhin, "Web Service Distributed Management: Management of Web Services (WSDM-MOWS 1.0)", OASIS Standard, <http://docs.oasis-open.org/wsdm/2004/12/mows/cd-wsdm-mows-1.0.pdf>, March 2005.

- (42) Edson Manoel et al., "Problem Determination Using Self-Managing Autonomic Technology", IBM Redbooks,
<http://www.redbooks.ibm.com/abstracts/SG246665.html?Open>, Jun 2005.
- (43) NSClient Official Site, <http://nsclient.ready2run.nl/download.htm>.
- (44) Command Line Process Viewer/Killer/Suspender for Windows NT/2000/XP,
<http://www.beyondlogic.org/consulting/processutil/processutil.htm>.
- (45) JAVA SDK 1.4,
<https://sdcl1a.sun.com/ECom/EComActionServlet;jsessionid=5B770A740DD256966785217086D242CF>.
- (46) Eclipse SDK 3.1.2, <http://www.eclipse.org/downloads/>.
- (47) Eclipse WTP Project, <http://www.eclipse.org/webtools/>.
- (48) The Apache Software Foundation, Apache Tomcat, Tomcat 5,
<http://tomcat.apache.org/download-55.cgi>.
- (49) Java Technology and XML Downloads - Java API for XML-Based RPC (JAX-RPC) Downloads & Specifications, <http://java.sun.com/xml/downloads/jaxrpc.html>.
- (50) Java.net, The Source for Java Technology Collaboration, NSClient4j Java API for accessing Windows Performance Monitor counters, <https://javatools-incubator.dev.java.net/>.
- (51) Anne-Marie Smit, "Adaptive Monitoring: an overview", Doc Science Internal Series 138, <http://www.doc.govt.nz/Publications/004~Science-and-Research/DOC-Science-Internal-Series/PDF/dsis138.pdf>, Oct. 2003.
- (52) Haydarlou, A.R. Overeinder, B.J. Brazier, F.M.T., "A Self-Healing Approach for Object-Oriented Applications", Proceedings of the 3rd International Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS 05), August 2005.
- (53) E. Michael Maximillien and Munindar P. Singh, "Toward autonomic web services trust and selection", International Conference on Service Oriented Computing, Proceedings of the 2nd international conference on Service Oriented Computing, <http://icsoc.dit.unitn.it/abstracts/A035.pdf>, 2004.

List of Publications

(1) Amir Zeid and Sherif A. Gurguis, "Towards Autonomic Web Services", European Conference in Object-Oriented Computing 2004, EWOOS 2004 workshop, <http://www.cs.ucl.ac.uk/staff/g.piccinelli/eoows.htm>, Oslo, Norway, June 2004.

(Paper)

(2) Sherif A. Gurguis and Amir Zeid "Web-Based Autonomic Computing MAPE Cycle", AICCSA 2005, <http://enr.smu.edu/cse/AICCSA-05/>, AUC-Cairo, Egypt, Jan 2005. (Paper)

(3) Sherif A. Gurguis and Amir Zeid, "Towards Autonomic Web Services: Achieving Self-Healing Using Web Services":

a. ICSE 2005, DEAS 2005 Workshop, <http://www.deas2005.cs.uvic.ca>, Missouri, USA, May 2005. (Short Paper)

b. ACM SIGSOFT Software Engineering Notes, Volume 30, Issue 4, <http://portal.acm.org/affiliated/citation.cfm?id=1082983.1083069&coll=portal&dl=ACM&CFID=72593439&CFTOKEN=25319059>, July 2005.

(4) Sherif A. Gurguis and Amir Zeid, "Towards Autonomic Web Services: Applying Self-Healing to Web Services", SCC 2005, DEAS 2005 Workshop, <http://conferences.computer.org/scc/2005/>, Florida, USA, July 2005. (Poster)

Appendix A: Acronyms and Glossary

Appendix A: Acronyms and Glossary

1. Acronyms

Table 3 List of Acronyms

| | |
|--------------|---|
| AC | Autonomic Computing |
| AI | Artificial Intelligence |
| AM | Autonomic Manager (s) |
| AMS | Autonomic Management System (s) |
| ACS | Autonomic Computing System (s) |
| ACSC | Autonomic Computing Systems Component (s) |
| ACWS | Autonomic Computing Web Service (s) |
| ATP | Autonomic Computing Touchpoint (s) |
| AWS | Analyzing Web Service (s) |
| BC | Business Capability (s) |
| BI | Business Interface(s) |
| BO | Business Operation (s) |
| BWS | Business Web Service (s) |
| BWSDL | Business WSDL document |
| CBE | Common Base Event |
| CIM | Common Information Model |
| CMC | Common Manageability Capabilities |
| CS | Computing System(s) |
| CWS | Configuring Web Service (s) |
| ECWS | Event-Collecting Web Service (s) |
| EMS | Enterprise Management Solution (s) |
| EPR | Endpoint Reference |
| EWS | Executing Web Service (s) |
| HWS | Healing Web Service (s) |
| IMS | Inter-systems Management Solution (s) |
| IT | Information Technology |
| JMX | Java Management Extensions |
| JSP | Java Servlet Page |
| ManageableWS | Manageable Web Service (s) |
| MAPE Cycle | Monitor, Analyze, Plan and Execute Cycle |
| MAPE-WS | MAPE Cycle Web Service (s) |
| MBean | Management Bean |
| MC | Manageability Capability |
| MCC | Manageability Capability Consumer |
| MCP | Manageability Capability Provider |
| MEP | Manageability Endpoint |
| MI | Manageability Interface(s) |
| MIB | Management Information Base |
| ManagingWS | Managing Web Service (s) |
| MO | Manageability Operation(s) |
| MOWS | Management of Web Services |
| MR | Manageable Resource(s) |

| | |
|-----------|---|
| MS | Management System (s) |
| MWSDL | Manageability WSDL Document |
| MUWS | Management using Web Services |
| MWS | Monitoring Web Service |
| OASIS | Organization for the Advancement of Structured Information Standards |
| OS | Operating System |
| OWS | Optimizing Web Service (s) |
| PWS | Planning Web Service (s) |
| Self-CHOP | Self Configuring, Self Healing, Self Optimization and Self Protecting |
| SMC | Specific Manageability Capabilities |
| SNMP | Simple Network Management Protocol |
| SOA | Service-Oriented Architecture |
| SWS | Securing Web Service (s) |
| URI | Universal Resource Identifier |
| WS | Web Service (s) |
| WS-BPEL | Web Services Business Process Execution Language |
| WSDM | Web Services Distributed Management |
| WEF | WSDM Event Format |
| WSAC | Web Services Autonomic Computing Framework |
| WSDM TC | Web Services Distributed Management TC |
| WSM | Web Services Management |
| WSMF | Web Service Management Framework |

2. Glossary

2.1 Analyzing Web Service (AWS)

It is a Web Service that provides the analysis phase functionalities of the Autonomic Control Loop over the Internet. The concept of AWS is introduced by our research.

2.2 Autonomic Computing System (ACS)

It is a computing system that adopts the notion of Autonomic Computing to be self-manageable.

2.3 Autonomic Computing System Component (ACSC)

It is a computing resource in an Autonomic Computing System; the component should behave autonomically on its own. The composition of a typical Autonomic Computing System Component adheres to the Autonomic Computing reference architecture.

2.4 Autonomic Computing Touchpoint (ATP)

It is the interface, through which, Autonomic Managers can communicate with Autonomic Components. A typical Autonomic Computing Touchpoint is composed of a set of sensors and effectors.

2.5 Autonomic Computing Web Service (ACWS)

It is a Web Service that provides one of the Autonomic Computing characteristics to Manageable Resources over the Internet. For example, there are Healing Web Services that provide autonomic healing capabilities to Manageable Resources over the Internet. The notion of Autonomic Computing Web Services is introduced by our research.

2.6 Autonomic Control Loop

It is a closed control loop implemented by Autonomic Managers in order to provide functionalities of monitoring, analyzing, planning, and executing. A typical control loop is called MAPE cycle.

2.7 Autonomic Manager (AM)

An Autonomic Manager represents the core of any Autonomic Computing System Component as it is responsible for providing self-management to that Component.

2.8 Business Interface (BI)

It is the interface that is exposed by a computing resource to allow requesters of the resource's services to execute some business operations and receive the results of those operations.

2.9 Business Capability (BC)

It is a capability of some computing resources to execute some Business Operations.

2.10 Business Operation (BO)

It is an operation that implements a part of the services provided by a computing resource. Business Operations are performed by the resource by external requesters. If the Business Operations of some computing resource cannot be executed, that resource must be managed so that its behaviour returns to be normal.

2.11 Business Web Service (BWS)

It is a Web Service that implements one or more Business Interface in order to expose a set of Business Operations to be performed by remote clients over the Internet.

2.12 Business WSDL Document (BWSDL)

It is a WSDL document that describes the Business Capabilities of some BWS.

2.13 Capability

A capability describes a set of properties, operations, events, and metadata. The semantics of a capability is described by some descriptive information, such as metadata.

2.14 Common Base Event (CBE)

It is an IBM initiative for unifying the way, by which logs are represented both syntactically and semantically.

2.15 Computing Resource

A computing resource may be physical, such as a printer, or logical, such as a software application.

2.16 Computing System (CS)

It is an aggregate of computing resources of different platforms, protocols, and vendors. A Computing System is built to fulfill a set of business requirements.

2.17 Configuring Web Service (CWS)

It is a Web Service that can provide the configuring Autonomic Computing characteristics to Manageable Resources over the Internet. The concept of CWS is introduced by our research.

2.18 Enterprise Management Solution (EMS)

It is a management solution that can be applied to computing resources within the boundaries of one administrative domain regardless of the type and platform of those resources.

2.19 Event-Collecting Web Service (ECWS)

It is a middleware Web Service that can be used for publishing and subscribing to events to build events-based management systems.

2.20 Executing Web Service (EWS)

It is a Web Service that provides the executing phase functionalities of the Autonomic Control Loop over the Internet. The concept of EWS is introduced by our research.

2.21 Healing Web Service (HWS)

It is a Web Service that can provide the healing Autonomic Computing characteristics to Manageable Resources over the Internet. The concept of HWS is introduced by our research.

2.22 Inter-Systems Management Solution (IMS)

It is a management solution that can be applied to different computing resources that span multiple administrative domains.

2.23 Manageability

It is the ability of a resource to be managed.

2.24 Manageability Capability (MC)

It is an integrated set of properties, operations, events, metadata and other semantics that supports a particular management concern. Therefore, a manageability capability can be thought of an abstract interface, similar to "marker" interfaces in Java.

2.25 Manageability Capabilities Consumer (MCC)

A manageability consumer acts as a managing system, such that it consumes the manageability capabilities offered by some managed system in order to control it.

2.26 Manageability Endpoint (MEP)

It is used to provide access to the exposed manageability capabilities

2.27 Manageability Interface (MI)

It is used to represent one or more manageability capabilities to be exposed to offer those manageability capabilities.

2.28 Manageability Operation (MO)

It is an operation that can be performed on a manageable resource through its manageability interface in order to allow external management systems to remotely control the behaviour of the resource. Manageability Operations should be performed regardless of the Business Operations being performed.

2.29 Manageability Capabilities Provider (MCP)

A manageability provider is used to expose manageability capabilities of some managed computing resource in order to enable that resource to be manageable. This role is not significant, and can be omitted as it might be implemented by the resource being managed.

2.30 Manageability WSDL Document (MWSDL)

It is the WSDL document that describes the Manageability Capabilities of some Manageable Web Service.

2.31 Manageable Resource (MR)

It is a computing resource that implements one or more manageability interfaces in order to be manageable by some external management systems.

2.32 Manageable Web Service (ManageableWS)

It is a Web Service that supports one or more manageability capabilities by implementing one or more manageability interfaces that expose the supported manageability capabilities.

2.33 Managing Web Service (ManagingWS)

It is a Web Service that supports a set of management capabilities and consumes a set of manageability capabilities offered by a Manageable Web Service in order to control that Web Service.

2.34 MAPE Web Services (MAPE-WS)

A MAPE Web Service provides one of the MAPE cycle's functionalities, such as Monitoring Web Service that provides monitoring functionalities. The concept of MAPE-WS is introduced by our research.

2.35 Monitoring Web Service (MWS)

It is a Web Service that provides the monitoring phase functionalities of the Autonomic Control Loop over the Internet. The concept of MWS is introduced by our research.

2.36 Optimizing Web Service (OWS)

It is a Web Service that can provide the optimizing Autonomic Computing characteristics to Manageable Resources over the Internet. The concept of OWS is introduced by our research.

2.37 Planning Web Service (PWS)

It is a Web Service that provides the planning phase functionalities of the Autonomic Control Loop over the Internet. The concept of PWS is introduced by our research.

2.38 Securing Web Service (SWS)

It is a Web Service that can provide the protecting Autonomic Computing characteristics to Manageable Resources over the Internet. The concept of SWS is introduced by our research.

2.39 Self-CHOP Characteristics

It is the set of major Autonomic Computing characteristics, which includes self-configuring, self-healing, self-optimizing, and self-protecting.

2.40 Web Service Endpoint

An endpoint has an address for delivering messages to the Web Service. In WSDL documents, an endpoint is defined by a *port* element.

2.41 Web Service Interface

An interface is used to describe one or more operations performed by the WS. In WSDL documents, an interface is defined by a *portType* element.

2.42 WSDM Event Format (WEF)

It is the standardized OASIS specification of IBM's Common Base Format initiative.

Appendix B: Overview of Relevant Standards

Appendix B: Overview of Relevant Standards

1. Web Services Stack

Web Services stack is composed of the following protocols suite that includes the de-facto standards for Web Services:

1.1 SOAP

SOAP is an XML-based messaging protocol that implements the bind and use operations, in the Web Services' triangle, by providing a simple and consistent mechanism that allows one application to send an XML message to another application [5] through standard communication protocols, such as HTTP, SMTP, FTP, etc; being based on XML, SOAP is both language and platform-independent [14]. A SOAP message is a one-way communication from a sender to a receiver that supports different communication behaviors, such as request, response, and notification. Moreover, an attachment to a SOAP message may be used to support the transport of non-XML data, such as multimedia files [5]. The SOAP protocol suite is composed of the following parts [5]:

1.1.1 Envelope

The envelope is used to identify the contents of a message and to explain how the message should be processed by the receiver. A typical SOAP envelope includes a header and a body: the header provides some directive or control information about the message, such as security, and the body contains the payload of the message.

1.1.2 Transport Binding Framework

SOAP 1.1 defines bindings for HTTP, and SOAP 1.2 defines bindings for both HTTP and SMTP; however, SOAP 1.2 is not officially supported yet as it is in the phase of specification.

1.1.3 Serialization Framework

Data included in SOAP messages are encoded using XML, such that data are passed as either literals or encoded values; however, there is no default serialization mechanism that maps application-defined data types to XML elements. A serialization mechanism can be customized if users do not want to use the mechanism that is defined by SOAP encoding rule. SOAP serialization mechanism supports common features existing in the type systems of most programming languages and

databases that include both scalar types, such as “integer”, “string”, etc and complex types, such as “struct”, “array”, etc.

1.1.4 RPC Representation

The SOAP sender sends a message, and the SOAP receiver decides what to do with the message; therefore, SOAP supports loosely-coupled relationships between applications. All the information a SOAP sender needs about a service is the format of the message and the access point of the services. Based on the content of the message, the receiver determines what is being requested by the sender and how to process that request. However, SOAP has an RPC representation that provides a bit tightly-coupled relationship. In SOAP RPC, the SOAP request is formulated as a method call with some or no parameters, and the SOAP response is a return value together with some or no parameters. SOAP RPC request and responses are serialized into a *struct* datatype and passed in the SOAP body. Although SOAP seems to be perfect for Web Services communication, other solutions can be also used, such as using HTTP-GET and HTTP-POST messages instead of SOAP messages [14].

1.2 WSDL

WSDL provides a set of XML vocabulary for implementing the publish operations, in Web Services’ triangle, by describing Web Services through the definition of abstract interfaces and bindings. Each Web Service can be described with the aid of a WSDL document, which is an XML-structured document that includes the following pieces of information: (1) the operations that the Web Service can perform, (2) how the Web Service could communicate with requesters; this includes describing the formats of the messages the Web Service can accept and process, the protocols that are supported by the Web Service, and (3) how the Web Service can be accessed by the requesters; this is achieved by stating the access point of an instance of the Web Service. From a WSDL document, the requester automatically generates a proxy, from which, the Web Service can be accessed.

From the viewpoint of WSDL, a Web Service is defined as collection of endpoints; each endpoint represents a set of operations; each operation is defined in terms of a set of messages that are either received or sent. A port type is a set of operations that are supported by one or more endpoint. A binding maps a port type to a specific protocol and data format. A Port is a single endpoint that instantiates a port type and

binding at a specific network address. Generally speaking, there are five major elements within any WSDL document:

1.2.1 Types <types>

It is the element that defines the datatypes that are used within messages. Datatyping system defined in the W3C XML schema is used by WSDL; however, any datatyping system may be used. Schema types can be mapped to most programming language type systems. SOAP tools use the type information in the process of encoding/decoding data in SOAP messages.

1.2.2 Message <message>

It is the element that defines the format of messages. A message consists of one or more logical parts; each logical part is associated with a type. In the case of SOAP RPC, each logical part represents a method parameter.

1.2.3 Port Type <portType>

It is the element that defines a set of operations; therefore, each portType element is composed of a set of <operation> elements, and within every <operation> elements, both input messages (<input>) and output messages (<output>) are defined. In SOAP RPC, each operation corresponds to a method. Each input and/or output element corresponds to a SOAP message exchanged between the client and the service.

1.2.4 Binding <binding>

It is the element that maps the operations and messages defined in each portType element to some protocol, such as HTTP transport protocol as well as data format specification, such as SOAP data encoding system.

1.2.5 Service <service>

It is the element that defines a collection of related ports, where each <port> element maps a binding to a location of an instance of the Web Service. Each Web Service is defined by a <service> element that can be accessed through endpoints. An endpoint specifies the location (e.g. IP address) at which a particular service can be accessed and which protocol is used for that Web Service. Web Services can be accessed using both SOAP messages and HTTP-GET and one endpoint can be specified for each type [14].

An interface is an abstract description of the Web Service operations, while a <service> element is a concrete description of the interface's location. An abstract

definition of a Web Service is given by the type, message, and port type elements. Actually, what is described by such elements is a service type. The service type is mapped to a specific protocol using the binding element; in order to provide reusability, the binding element can be maintained a separate WSDL document. Eventually, the service element maps the service type together with the binding to a specific instance of the Web Service [5]. A binding specifies what kind of messages are exchanged and in which style; therefore, it is used to map an abstract description to a location. Moreover, interfaces can be extended to provide more operations by new interfaces. This is achieved by using *extends* attribute for the `<interface>` element with the value of the attribute pointing to one or more other interfaces [14]; this feature is shown in the listing below:

```
<interface name="intf1">
  <operation name="op1" .../>
  ...
</interface>
<interface name="intf2" extends="intf1">
  <operation name="op2" .../>
  ...
</interface>
```

As it was mentioned by [14], a Web Service can be modularized by splitting the WSDL document into parts and storing each part in a separate document, even at separate locations. Moreover, messages and interfaces could be defined in one WSDL document and imported by another WSDL document. This can be achieved by using the `<import>` and `<include>` elements. The UDDI Technical Committee recommends a division of WSDL documents into two separate WSDL definitions: (1) service definition part (e.g. interface) that contains the `<types>`, `<message>`, `<interface>` and `<binding>` and (2) service implementation definition that contains the `<service>` element. This splitting would be useful in some scenarios, such as deploying a Web Service that can be accessed by the same binding and deployed at many different locations. The WSDL description of Web Services can be further modularized by separating the document into an abstract part containing the messages and interfaces (e.g. what part) and two other parts: a binding (e.g. how part) and a service (e.g. where part) [14].

1.3 UDDI

UDDI is a general-purpose registry that implements the find-operations by providing a mechanism to register and categorize Web Services so that consumers can easily locate Web Services that match their needs by communicating with the registry using SOAP messages. UDDI includes information about businesses and the services they offer; this information is organized as follows [5]:

1.3.1 Business Entity

A business entity contains general information about the business, such as name, description, and contact information. A unique business identifier and a list of categorizations that describe the business can be also associated with each business.

1.3.2 Business Service

Each business entity is associated with a list of services offered by the entity. Each business service contains the following information: service description, categories that describe the service, a list of binding templates.

1.3.3 Binding Templates

Binding templates provide information on where a service can be found and how that service can be used. For example, a typical binding template may contain the access point of the service as well as a pointer to a WSDL document describing the service.

1.3.4 Service Types

A construct called *tModel* is used to define an abstract service.

1.4 WS-Resource

In addition to WS Stack Suites, there are some other standards related to WS that are significant to the context of management, such as: WS-resource, which is also called WS-RF that is a suite of the following three OASIS standards:

1.4.1 WS-ResourceProperties

It is a way to use WS-Addressing in order to associate 'state' with Web Services by associating an XML schema with the WSDL description of a web service. Every web service implementing WS-Resource properties defines 1 mandatory and 3 optional web service operations. The mandatory operation is *Element [] GetResourceProperty (QName qn)*. The QName is the element name of a top-level node in the stateful DOM that WS-RP associates with the web service. *GetResourceProperty ()* returns

the top-level elements matching the QName (along with their children). The optional operations are *SetResourceProperties*, which is used for inserting, updating, and deleting top-level nodes; *QueryResourceProperties*, which evaluates an XPath against the web service's DOM and a variant, get called *GetMultipleResourceProperties*. *WS-ResourceProperties* also defines a property-change notification format for *WS-Notification*.

1.4.2 WS-ResourceLifetime

It defines a way to lease web services using a web service operation to set lease termination time, two Elements to be used by *WS-ResourceProperties* as properties, and an expiration message format for *WS-Notification*. Leasing is a way to write fault-tolerant distributed applications. It allows locks to 'expire' gracefully even if the connection between lock holder and lock manager is severed.

1.4.3 WS-Notification

It is a suite of 3 OASIS specifications: *WS-BaseNotification*, *WS-BrokeredNotification*, and *WS-Topics*. *WS-Notification* defines an envelope format for notifications. The format of the *WS-Notification* envelope is provided by another specification, such as *WS-ResourceProperties*, *WS-ResourceLifetime*, or *WSDM*. *WS-Notification* also specifies two web service operations on the *NotificationProvider* (*subscribe()* and *getCurrentMessage()*), and a web service operation <operation> to notify recipient, *notify()*. *WS-Notification* 'Subscriptions' are themselves web-services with *WS-Addressing* and *WS-ResourceLifetime*. *WS-ResourceLifetime*'s leasing is used to renew/cancel subscriptions.

2. Logs Representation

2.1 Common Base Events

The issue of disparate log information should be resolved in order to permit the building of self-healing and problem determination systems. Therefore, there is a need to adopt a formal and standard approach for representing and reporting such information. CBE was evolved by IBM to address that issue by structuring information provided by log files in a standard way to enable the seamless analysis of log information. This section introduces and overview about CBE.

2.1.1 What is a CBE?

CBE defines the structure of an event in a consistent and common format using XML schema definition. An event can be defined as any significant change in the state of a computing resource and can be generated for various reasons, such as reporting a problem or the completion of task. Events are indicatives of situations; a situation is the state of a computing resource, such as START situation. Events are mapped to CBEs; and then, emitted in the form of messages to report situations as they occur. In addition to ensuring consistency in representing events, CBE contains prescribed elements that guarantee that accurate and sufficient information will be captured to represent events. A CBE schema describes the structure of the generated CBE and defines the mandatory fields; a CBE schema is provided by GLA.

2.1.2 CBE Structure

Because any CBE consists of the three elements; CBE structure is referred to as the 3-tuple structure.

- I. The identity of the component reporting the situation.
- II. The identity of the component affected by the situation, which is usually the same as the reporting component.
- III. The observed situation.

The details of situation are contained in the *situation element*; the element is extensible to include extra information to accommodate specific information. CBE defines a set of 11 predefined situations that includes: *startSituation*, *stopSituation*, *connectSituation*, *configureSituation*, *requestSituation*, *featureSituation*, *dependencySituation*, *createSituation*, *destroySituation*, *reportSituation*, and *availableSituation*. In addition, *otherSituation* can be used to define component-based situations.

2.1.3 Significance of CBE

CBE provides an approach for achieving interoperable, effective intercommunication among disparate computing resources. Therefore, CBE supports problem determination, which is a key capability of autonomic managers, in an enterprise-wide.

2.2 WSDM Event Format (WEF)

Information that represents management events is represented in XML format, such that data that are related to management events are categorized into three basic categories: 1) *event reporter*, 2) *event source* and 3) *situation data*. An event contains a set of basic and additional information as follows:

- I. **ReportTime** is a recommended element of type `dateTime`; it includes the date and time, at which, the event was reported.
- II. **EventId** is a required element of type `URI`; it is the primary identifier of an event; therefore, it must be unique within the scope of each source.
- III. **SourceComponent** is a required element that refers to the source of the event. It includes the following child elements:

1. **ResourceId** is an optional element of type `URI`; it identifies the MablWS that is associated with the event. If this element is provided, it must correspond to the `ResourceId` property for the source of the event.

2. **Component Address** is a recommended element that contains specific elements that are used to identify the address of a component; more than one child element can be included, where each one represents an alternate address of the same source. The element supports `{any}` in order to support any XML representation of the component address; one commonly used address type is the `EPR`. In the case of MablWS, it is recommended that the `MEP` is used as the address type.

- IV. **ReporterComponent** is a required element only if the reporter is different from the source and it refers to the reporter of the event. When the element isn't included, the source is realized as the same as the reporter. The content of the `ReporterComponent` is similar to that of the `SourceComponent`.

- V. `{any}` is an XML open content that provides a container for additional data associated with the event. The message content `GED` is inserted in that container.

VI. **SituationCategory**

The `SituationType` element is a required element that is used to support classification of events in order to categorize the situation that caused the event. Situations are categorized into 11 predefined situations that include *AvailabilitySituation*, *CapabilitySituation*, *ConfigureSituation*, *StopSituation*, *StartSituation*, *RequestSituation*, *DestroySituation*, *CreateSituation*,

DependencySituation, *ConnectSituation*, and *ReportSituation*. In addition there is 1 user-defined situation, which is *OtherSituation*. The order of the categories above depends on the precedence, such that the use of a higher precedence category permits more effective correlation and analysis of reported events in order to indicate whether there is a serious problem or not.

VII. SituationTime

It is a required element that represents the date and time, at which, an event has been observed; therefore, the element is of type `dateTime`.

VIII. Priority

It is an optional element that is used to represent how an event is important. A range of priorities is constrained to values from 0 to 100, where the predefined values are: 10 (e.g. low), 50 (e.g. medium) and 70 (e.g. high).

IX. Severity

It is an optional element that is used to represent how an event is severe. Used severity levels are as follows:

1. **Unknown (0):** The severity level cannot be determined.
2. **Information (1):** An expected message has been generated (e.g. a process has begun).
3. **Warning (2):** There might be a problem; and therefore, the situation should be analyzed.
4. **Minor (3):** A small problem that has no impact on the normal use of the service has occurred.
5. **Major (4):** A minor problem that might hinder the normal use of the service has occurred.
6. **Critical (5):** Although the service is still available, an error has occurred and an immediate action is required.
7. **Fatal (6):** Because of the occurrence of some unrecoverable error, the service is no longer available.

3. Web Services Management

3.1. Using Web Services Platform Notions in the context of management

3.1.1. Resource

The concept of WS-Resource that was introduced by WSRF is used by MUWS in defining a manageable resource, which is created by composing a MEP with the resource and making the resource accessible through that MEP.

3.1.2. Properties

Properties are describable information that can be queried and/or may be set. Properties are associated with operations and messages and they are all described by schema documents, which enables discovery of, reading and writing to a certain property. For each property, there are two attributes: 1) name (e.g. declaration) and 2) type (e.g. description). For the purpose of management, a property may be used to represent metrics, configuration values, etc. The concept of management properties is based on the concept of properties that was introduced by WS-ResourceProperties (WSRF). A ManageableWS exposes its properties in an RPD and makes it available to the ManagingWS. In an RPD, the children of the document root represent all the properties of the ManageableWS. An interface that includes properties should define methods for accessing those properties. Some methods must be implemented by the interface, such as `GetResourceProperty()`, and some other methods may be implemented, such as `GetMultipleProperties()`, `SetResourceProperties()`, or `QueryResourceProperties()`. The last one should support the XPath 1.0 expression dialect.

MUWS 1.0 uses XML Schema in order to describe properties, such that each property is defined as XML GED. The process of creating a property involves the following steps:

1. Schema for the property.
2. Describing the semantics of the property.
3. Defining the cardinality of the property.
4. Including any relevant metadata.

3.1.3. Operations

MUWS 1.0 uses WSDL to describe operations. The process of creating an operation involves the following steps:

1. A WSDL portType containing a WSDL operation corresponding to the operation.
2. Describing the semantics of the operation.
3. Including any relevant metadata.

3.1.4. Events

One or more events should be offered by a MEP in order to reflect a change in the supported properties. There are two components of any event: 1) *event type* and 2) *event message*. In MUWS 1.0, event types are defined by providing a combination of: 1) a *topic* QName; the topic provides information about why the event was generated, and 2) a *message content* GED; the message content represents the information that is transmitted as a part of the *notification message*. Each element in an event type is not unique; however, an event is uniquely identified by the combination. The process of creating a new event involves the following steps:

1. The topic/message content combination.
2. Describing the semantics of both the topic and the message content element.
3. Including any relevant metadata.

3.1.5. Metadata

Generally speaking, metadata can be defined as data about data; therefore, metadata can be used as additional descriptive information about the content of MI in the context of management. Metadata can be applied to different aspects of MI, such as MC, properties, operations and events. In order to support real time analysis of the description documents, MUWS 0.5 expresses the metadata about MI as XML element attributes. MUWS 1.0 supports both design-time and run-time analysis of the description documents.

3.1.6. Addressing

An address, or reference, is a data structure that is used to reference a unique WS. This data structure must hold sufficient information in order to enable that WS to be located and receives requests. In addition, the data structure should provide the information required to locate the description of its WS in order to enable ManagingWS to understand the semantics of the WS. Obtaining references to MEP

use the same mechanism used in *WSRF*, which leverages the concept of *EPR* that was introduced by *WS-A*. In the case of lacking an *EPR* for the *MEP*, a *ManagingWS* may try to invoke some manageability operations on some *WS* without including the reference information; and if that invocation succeeds, the *ManagingWS* can assume that the *WS* is a *ManageableWS* that is accessible through some manageability provider (consolidated *MI*).

3.1.7. Notification

Notification is a mechanism that advertises the events to the interested recipients. At the beginning, an interest in receiving some notification must be established with the source by the interested recipient. The relation between source and recipient can be also established through a third-party broker. Regardless of the method used in registering to notifications, the address (es) of recipients should be provided so that notification messages are delivered to them. In the context of management, *ManageableWS* need to deliver information reflecting, for example, current state to *ManagingWS* so that the latter is able to provide proper management to the former. As it is described by *WS-T*, if a certain manageability capability offers events to manageability consumers, the definition of the capability should include a topic space that contains a set of topics where each topic is associated with an event that is offered by the manageability capability. As it is described in the text, an event is a combination between a topic and message content. And the topic included in the event is mapped to a topic in the topic space.

3.1.8. Versioning

Version is attribute that is used to identify a sequence of modifications applied to the *ManageableWS*. Version keeps the *ManagingWS* updated whether the *MI* of *ManageableWS* have changed since the last copy of those interfaces have been obtained. *MUWS 0.5* provides access to the Version attribute as one element of the *Identity* property.

3.1.9. Security

The same security requirements in the scope of *BWS* can be adhered in the scope of *ManageableWS*. For example, a manageability operation can be granted only to some manageability consumers that present some *ID* in their request messages. *WS-Security* can be used in this sense.

3.1.10. Registration and Discovery

Registration is the process of announcing the existence of some element so that it can be discovered. *Discovery* is the process of locating an existing element so that it can be used. A sub-process of Discovery is *Location* that retrieves the address of the element. Registration, Discovery and Location are implemented by a *Registry*.

3.2. WSDM List of Manageability Capabilities

The following table lists the common Manageability capabilities that were introduced by WSDM together with the URI of the describing document of each capability.

Table 4 WSDM Manageability Capabilities URIs

| MC | URI |
|------------------------------|--|
| Advertisement | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Advertisement |
| CalloutUser | http://www.ibm.com/xmlns/prod/autonomic/CalloutUser |
| Configuration | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Configuration |
| CorrelatableProperties | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/CorrelatableProperties |
| Description | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Description |
| Identity | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Identity |
| Identification | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Identification |
| ManageabilityCharacteristics | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/ManageabilityCharacteristics |
| ManageabilityReferences | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/ManageabilityReferences |
| Metrics | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Metrics http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Metrics |
| MetricsControl | http://www.ibm.com/xmlns/prod/autonomic/touchpoint/MetricsControl |
| OperationalState | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/OperationalState |

| | |
|------------------------|--|
| | open.org/wsdm/2004/12/mows/capabilities/Operational State |
| OperationalStatus | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Operational Status http://docs.oasis-open.org/wsdm/2004/12/mows/capabilities/Operational Status |
| Relationships | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Relationships |
| RelationshipAccess | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Relationship Access |
| RelationshipResource | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/Relationship Resource |
| RequestProcessingState | http://docs.oasis-open.org/wsdm/2004/12/mows/capabilities/Request Processing State |
| ResourceType | http://www.ibm.com/xmlns/prod/autonomic/touchpoint /ResourceType |
| State | http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/State |

3.3. WSDM List of Manageability Capabilities Elements

Table 5 WSDM Manageability Elements

| Element | Description |
|--|--|
| mows-events:Digest | MOWS Manageability Capabilities Topics |
| mows-events:IdentificationCapability | |
| mows-events:MetricsCapability | |
| mows-events:OperationalStateCapability | |
| mows-events:OperationalStatusCapability | |
| mows-events:RequestCompleted | |
| mows-events:RequestFailed | |
| mows-events:RequestProcessing | |
| Mows-events:RequestProcessingObservations | |
| Mows-events:RequestProcssingObservationsWith Attachments | |
| Mows-events:RequestProcessingStateCapability | |
| mows-events:RequestReceived | |
| mows-xs:OpertationalStateType | |

| | |
|--|---|
| | substate. |
| mows-xs:RequestProcessingStateType | Used for declaring elements which designate a request processing state: top-level or substate. |
| muws-events:ConfigurationCapability | MUWS Manageability Capabilities Topics |
| muws-events:DescriptionCapability | |
| muws-events:IdentityCapability | |
| Muws-events:ManageabilityEndpointCreation | |
| Muws-events:ManageabilityEndpointDestruction | |
| Muws-events:ManageableResourceCreation | |
| Muws-events:ManageableResourceDestruction | |
| muws-events:ManageableRelationships | |
| | |
| muws-events:MetricsCapability | |
| muws-events:OperationalStatusCapability | |
| Muws-events:RelationshipAccessCapability | |
| muws-events:RelationshipCreated | |
| muws-events:RelationshipDeleted | |
| Muws-events:RelationshipResourceCapability | |
| muws-events:StateCapability | |
| | Element contained within the Source Component and the Reporter Component that are contained with the ManagementEvent. |
| muws-p1-xs:ComponentAddress | The CorrelatableProperties capability |
| muws-p1-xs:CorrelatableProperties | Element contained within the ManagementEvent |
| muws-p1-xs:EventId | The Identity manageability capability as defined by MUWS 1.0 Part 1 |
| muws-p1-xs:Identity | An instance of the ManageabilityCapability property of the ManageabilityCharacteristics capability. |
| muws-p1-xs:ManageabilityCapability | The ManageabilityCharacteristics capability |
| muws-p1-xs:ManageabilityCharacteristics | The element that contains the a reference to a MEP |
| muws-p1-xs:ManageabilityEndpointReference | WSDM MUWS event-payload container |
| muws-p1-xs:ManagementEvent | Element contained within the ManagementEvent if the source is different from the reporter. |
| muws-p1-xs:ReporterComponent | The sole property of the Identity capability that is defined by MUWS 1.0 |
| muws-p1-xs:ResourceId | |

| | |
|------------------------------------|--|
| | Part1. An instance of this property is contained within contained within the Source Component and the Reporter Component that are contained with the Management Event. |
| muws-p1-xs:SourceComponent | Element contained within the ManagementEvent |
| muws-p2-xs:AccessEndpointReference | An element contained within the Relationship GED. |
| muws-p2-xs:CalculationInterval | A property of the Description capability |
| muws-p2-xs:Caption | A property of the Description capability |
| muws-p2-xs:Capability | A metadata element that can be applied to all the aspects of an MI. it contains the URI of the manageability capabilities |
| muws-p2-xs:CategoryType | The top level element of event classifications. |
| muws-p2-xs:ChangeType | A metadata element of a metrics property |
| muws-p2-xs:CreationNotification | Notification on the creation of either MEPs of MablWSs. |
| muws-p2-xs:CurrentTime | The sole property of the Metrics capability |
| muws-p2-xs:Description | The Description Manageability Capability |
| muws-p2-xs:Description | A property of the Description capability |
| muws-p2-xs:DestructionNotification | Notification on the destruction of either MEPs of MablWSs. |
| muws-p2-xs:EnteredState | A child element of the StateTransition element. |
| muws-p2-xs:GatheringTime | A metadata element of a metrics property |
| muws-p2-xs:Idempotency | A metadata element that is applied to operations and indicates whether invoking an operation twice is equivalent to invoking it once. |
| muws-p2-xs:Message | A child element that is contained within the Situation element |
| muws-p2-xs:MetricAttributes | An attribute group that has any of the metric attributes defined within it. |
| muws-p2-xs:MetricGroup | A metadata element of a metrics property |
| muws-p2-xs:Modifiability | A metadata element that is applied to properties. |
| muws-p2-xs:Mutability | A metadata element that is applied to properties. |
| muws-p2-xs:Name | An element contained within the Relationship GED. |
| muws-p2-xs:Notifiability | A metadata element that is applied to properties. |
| muws-p2-xs:OperationalStatus | It is an instance of the OperationalStatus property representing the OperationalStatus capability. |

| | |
|---|---|
| muws-p2-xs:Participant | An element contained within the Relationship GED. |
| muws-p2-xs:PostCondition | A metadata element that is applied to operations and contains a statement that asserts true if an operation has completed successfully. |
| muws-p2-xs:PreviousState | A child element of the StateTransition element. |
| muws-p2-xs:Priority | A metadata element that is applied to properties. |
| muws-p2-xs:QueryRelationshipsByType | The request of the QueryRelationshipsByType operation that is provided by the Relationships capability |
| muws-p2-xs:QueryRelationshipsByTypeResponse | The response of the QueryRelationshipsByType operation that is provided by the Relationships capability |
| muws-p2-xs:Relationship | The element that represents an instance of the Relationships capability |
| muws-p2-xs:RelationCreatedNotification | The notification message in the case of creating a new relationship. |
| muws-p2-xs:RelationDeletedNotification | The notification message in the case of deleting a relationship. |
| muws-p2-xs:RequestedType | A QName contained with the request of the QueryRelationshipsByType operation in order to identify the requested types of relationships |
| muws-p2-xs:Role | An element contained within the Relationship GED. |
| muws-p2-xs:Situation | Child element of the ManagementEvent that should exist in the notifications |
| muws-p2-xs:SituationCategory | Child elements contained within the Situation element |
| muws-p2-xs:SituationTime | |
| muws-p2-xs:Severity | |
| muws-p2-xs:State | The State Manageability Capability as defined by MUWS 1.0 P2 |
| muws-p2-xs:StateTransition | An XML element that represents the change of some state in the state model |
| muws-p2-xs:StaticValues | A metadata element that is applied to properties. |
| muws-p2-xs:SubstitutableMsg | Child elements contained within the Situation element |
| muws-p2-xs:SuccessDisposition | |
| muws-p2-xs:TimeScope | A metadata element of a metrics property |
| muws-p2-xs>Type | |
| muws-p2-xs:Units | A metadata element that is applied to properties and specifies the default unit for some property |
| muws-p2-xs:ValidRange | A metadata element that is applied to properties. |

| | |
|------------------------|---|
| muws-p2-xs:ValidValues | A metadata element that is applied to properties. |
| muws-p2-xs:ValidWhile | A metadata element that is applied to all the aspects of an MI and contains a statement that when is evaluated to true indicates that some manageability aspect, to which, the metadata element is related is true. |
| muws-p2-xs:Version | A property of the Identity capability that is defined by MUWS 0.5 |
| muws-xs:Identity | The Identity manageability capability as defined by MUWS 0.5 |
| muws-xs:Name | A property of the Identity capability that is defined by MUWS 0.5 |
| muws-xs:ResourceId | A property of the Identity capability that is defined by MUWS 0.5 |
| muws-xs:ResourceState | The sole property of the State capability as defined by MUWS 0.5. |
| muws-xs:State | The State manageability capability as defined by MUWS 0.5 |
| muws-xs:Version | A property of the Identity capability that is defined by MUWS 0.5 |

4. Autonomic Computing Manageability Capabilities

In the following subsections, the set of Autonomic Computing Manageability Capabilities that are defined by *Autonomic Computing Touchpoint (ACTP)* is discussed:

4.1 ResourceType Capability

It is used to classify Manageable Resources in order to enable Autonomic Managers to provide multiple levels of management functions based on the knowledge that an Autonomic Manager has about the resource being managed. For example, a Microsoft Windows Operating System (OS) might have the following classifications: “OS”, “Windows”, “Win32 OS”, and “Windows XP”. By using the *ResourceType*, an Autonomic Manager that is specialized in managing a “Windows XP” may be able to perform a more reliable management than another Autonomic Manager that can manage only “Win32 OS”; however, both Autonomic Managers should manage the same resource to some acceptable extent. In the context of WS, a WS can be considered as a composite Manageable Resource that consists of HTTP server, database server, application server, etc. Each type of those servers can be

separately managed by separate Managing Web Service. Moreover, *Relationships* Manageability Capability can be used to tie those lower-level resources together.

4.2 MetricsControl Capability

Because the continuous collection of some metrics may be costly, Autonomic Computing Touchpoint introduced the *MetricsControl* capability that defines properties and operations in order to control the collection of some metrics. A metric, to which, this capability is applied is called a *controllable metric*. A controllable metric is a metric, whose collecting can be controlled, such that the collection can be enabled or disabled by the Managing Web Service based on the cost of collection.

4.3 CalloutUser

A *callout* is a request made by some Manageable Web Service to fulfill some requests, such as requesting information from a third party before completing some request. The *CalloutUser* capability provides a mechanism, by which, Managing Web Services can determine what callout capabilities a Manageable Web Service may require from Managing Web Services; and then, assign providers to fulfill those requirements. It defines the *Callout* property that identifies the callout capabilities in terms of what Managing Web Services can be used with what resources. Managing Web Services can use a process similar to subscriptions in order to bind their callout capabilities to the Manageable Web Services by assigning a callout provider to each required callout capability by the Manageable Web Services; and then, inform the Manageable Web Services which callouts it can use. However, a Managing Web Service may be either unable or not willing to handle some callouts.

4.4 List of Manageability Capabilities

Table 6 Autonomic Computing Manageability Capabilities URIs

| MC | URI |
|----------------|---|
| CalloutUser | http://www.ibm.com/xmlns/prod/autonomic/CalloutUser |
| MetricsControl | http://www.ibm.com/xmlns/prod/autonomic/touchpoint/MetricsControl |
| ResourceType | http://www.ibm.com/xmlns/prod/autonomic/touchpoint/ResourceType |

4.5 List of Manageability Elements

Table 7 Autonomic Computing Manageability Elements

| Element | Description |
|------------------------------|---|
| actp:ActiveMetric | A property defined for the MetricControl MC that determines the metrics that are currently active and being collected |
| actp:ActiveMetricList | It is a list attribute that contains all the active metrics. |
| actp:ControlMetricCollection | An operation defined for the MetricControl MC that controls which metrics are collected. |
| actp:MetricControl | A MC that is used by AC to control the collection of metrics. |
| actp:Name | A property defined for the ResourceType MC that contains a name that is unique per an instance of some resource. |
| actp:ResourceType | A MC that is used to identify the type of a resource being managed. |
| actp:ResourceType | A property defined for the ResourceType MC that is used to contain the URI of types of the resource being managed. |

5. Summary of Namespaces and Standards

5.1 Namespaces

Table 8 List of Namespaces

| Namespace | Description Document |
|------------------------|---|
| mows-events | http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part2-events.xml |
| mows-wsdl | http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part2.wsdl |
| mows-xs | http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-mows.xsd |
| muws-xs | http://docs.oasis-open.org/wsdm/2004/04/muws-0.5/schema |
| muws-wsdl | http://docs.oasis-open.org/wsdm/2004/04/muws-0.5/wsdl |
| muws-p1-xs or muws-xs1 | http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part1.xsd |
| muws-p2-xs or muws-xs2 | http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part2.xsd |
| muws-p2-wsdl | http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-mows.wsdl |
| muws-events | http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-mows-events.xml |
| soap | http://schemas.xmlsoap.org/wsdl/soap/ |
| wsa | http://schemas.xmlsoap.org/ws/2004/08/addressing |
| wsdl | http://www.w3.org/2002/07/wsdl |

| | |
|-------|---|
| wsnt | http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.xsd |
| wsrp | http://docs.oasis-open.org/wsrp/2004/06/wsrp-WS-ResourceProperties-1.2-draft-01.xsd |
| wstop | http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.xsd |
| xs | http://www.w3.org/2001/XMLSchema |

5.2 List of Standards

Table 9 List of Standards

| Standard | Version | Organization |
|---|---------|--------------|
| ACTP | Draft | IBM |
| MOWS | 1.0 | OASIS |
| MUWS | 0.5 | OASIS |
| MUWS –Part1 | 1.0 | OASIS |
| MUWS –Part2 | 1.0 | OASIS |
| SOAP | 1.1 | W3C |
| UDDI | 3.0 | UDDI |
| Web Service Management: Service Lifecycle (WSLC) | | W3C |
| Web Services Resources Metadata (WS-ResourceMetadataDescriptor) | 1.0 | OASIS |
| Web Services Security | 1.0 | OASIS |
| WS-Addressing (WS-A) | | W3C |
| WS-Architecture (WS-Arch) | | W3C |
| WS-Base Notification | 1.2 | OASIS |
| WSDL | 1.1 | W3C |
| WSDM Event Format (WEF) | | OASIS |
| Web Services Distributed Management (WSMD) | | |
| WS-ServiceGroup (WS-SG) | 1.2 | OASIS |
| WS-Resource Lifetime (WS-RL) | 1.2 | OASIS |
| WS-Resource | 1.2 | OASIS |
| WS-Resource Framework (WSRF) | 1.0 | hp & OASIS |
| WS-Resource Properties (WS-RP) | 1.1 | OASIS |
| WS-Topics | 1.2 | OASIS |
| XML | 1.0 | W3C |
| XML Schema (XS) | | W3C |
| XML Namespaces (XNS) | | W3C |
| XML Path Language (XPath) | 1.0 | W3C |

Appendix C: Overview of Relevant Tools

The following table provides an overview of the tools used in the study. The tools are categorized into data collection, data analysis, and data management. The table includes the name of the tool, its purpose, and the version used.

| Tool | Purpose | Version |
|---------------------------|--|---------|
| Microsoft Word | Document creation and editing | 2010 |
| Microsoft Excel | Data analysis and visualization | 2010 |
| Microsoft PowerPoint | Presentations | 2010 |
| Microsoft Access | Data management | 2010 |
| Microsoft Outlook | Email and calendar management | 2010 |
| Microsoft OneNote | Note-taking and organization | 2010 |
| Microsoft SharePoint | Collaboration and document sharing | 2010 |
| Microsoft Dynamics | Business process automation | 2010 |
| Microsoft Lync | Instant messaging and video conferencing | 2010 |
| Microsoft Exchange | Email and calendar services | 2010 |
| Microsoft Office Web Apps | Web-based document editing | 2010 |
| Microsoft Office Live | Cloud-based document storage and sharing | 2010 |
| Microsoft Office 365 | Cloud-based productivity suite | 2010 |

Appendix C: Overview of Relevant Tools

1. IBM Autonomic Computing Toolkit

IBM Autonomic Computing Toolkit is a collection of technologies, tools and scenarios that is designed to be used in adopting and developing autonomic behavior in computing systems; for example, management capabilities, such as problem determination, can be enhanced by using products provided by the toolkit, such as Autonomic Management Engine, Generic Log Adapter, Log and Trace Analyzer and Common Base Events. In addition, management solutions can be created to meet specific requirements by using tools provided by the toolkit, such as Integrated Solutions Console, Resource Model Builder, Adapter Rule Editor, and other Eclipse plug-ins. Moreover, the toolkit provides a set of scenarios that show how the components of the toolkit can be integrated together to be used in realistic situations; for example, the toolkit includes the problem determination scenario that includes self-healing tasks and installation scenarios that perform self-configuration tasks. The toolkit is available for download at: <http://www-128.ibm.com/developerworks/autonomic> from IBM developerWorks Web site. The components of the toolkit are listed below:

1.1 Autonomic Management Engine (AME)

AME can be used in implementing autonomic managers and includes built-in representations of the autonomic control loop. AME uses resource models to monitor and manage different computing resources.

1.2 Resource Model Builder (RMB)

RMB is used to create resource models that contain specific metrics, events, thresholds, etc. together with specifications for corrective actions in the case of failure. A resource model specifies which data are accessed at run-time and how that data are processed; data are gathered at regular intervals of time, called cycles; the cycle associated with each resource model can be customized as required. At each cycle, collected data represent a snapshot of the status of the MR. One or more thresholds are defined per each resource model; the values of thresholds can be also customized as required. For example, the *Process Resource Model* obtains data related to the running processes; moreover, performance data are automatically collected and processed to determine whether the system is performing as expected.

1.3 Generic Log Adapter (GLA)

GLA is a facility that translates product-specific logs into CBE format to unify logs information

1.4 Adapter Configuration Editor (ACE)

Specific parsing rules that are used in translation is created with the aid of ACE, which is provided as an Eclipse plug-in. Adapter configuration files that are created by ACE can be run with the aid of GLA Runtime and Rule Sets bundle.

1.5 Log and Trace Analyzer

LTA consumes CBE messages and analyzes them in order to create a correlated view of the log events. Therefore, LTA contains a log analysis engine that implements an algorithm, which takes an incident from a log file as an input, matches this incident based on a set of predefined rules against a set of available symptoms provided by some symptom database, and returns a set of solutions and directives for the matched symptoms. A default analysis engine is provided by LTA; however, Java code-based parsers can be written to handle complex and long log messages. Moreover, LTA supports the facility of remotely importing any type of log file through the Remote Agent Controller (RAC) component.

1.6 Agent Controller

RAC component is provided by the Agent Controller bundle

1.7 Integrated Solutions Console (ISC)

The ISC component is a Web-based infrastructure that provides an environment for hosting different autonomic management capabilities. Autonomic management functions, such as setup, configuration, monitoring, and control can be administrated from ISC. ISC plug-ins can be created using the ISC toolkit, which includes ISC runtime and sample components.

2. Eclipse SDK

Eclipse is an integrated development environment (IDE) for writing Java code. It is open-source and free for anyone to download. It provides everything a powerful IDE should have: an advanced Java source editor with incremental compilation, application debugging support, and support for version control systems, and so on. Specifically, The Eclipse SDK includes the Eclipse Platform, Java development tools, and Plug-in Development Environment, including source and both user and

programmer documentation. Eclipse does not include a Java runtime environment (JRE).

Even if you are an Eclipse user, what you may not realize is that it is a lot more than just a Java IDE. Initially, Eclipse started as a tool integration platform. The original Eclipse developers recognized that there is a common set of infrastructure with a potential to be useful not just for a Java IDE, but other integrated development tools as well, regardless of their underlying programming languages. Indeed, today there are Eclipse-based IDEs for developing applications in C/C++, COBOL, PHP, Eiffel, Ruby, and others.

However, after a few initial releases, the developer community quickly realized the potential of using Eclipse technology in contexts other than software development. Its modular architecture and solid extension and customization strategy allowed anyone to create applications not originally anticipated by Eclipse developers. Thus, over the last few releases, Eclipse expanded its focus from being a tool integration platform to a full-fledged rich client platform (RCP). Its various components and frameworks have been refactored in a way that makes it possible to use Eclipse as the foundation for both IDE and non-IDE applications. Today, the list of companies building their products on Eclipse RCP is rapidly growing. Soon, you may find yourself using an Eclipse RCP application, whether you actually realize it or not.

3. Eclipse WTP Project

The Eclipse Web Tools Platform (WTP) project extends the Eclipse platform with tools for developing J2EE Web applications. The WTP project includes the following tools: source editors for HTML, JavaScript, CSS, JSP, SQL, XML, DTD, XSD, and WSDL; graphical editors for XSD and WSDL; J2EE project natures, builders, and models and a J2EE navigator; a Web service wizard and explorer, and WS-I Test Tools; and database access and query tools and models.

4. Apache Tomcat Application Server

Apache Tomcat is the Servlet container that is used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies. The Java Servlet and Java Server Pages specifications are developed by Sun under the Java Community Process. Apache Tomcat is developed in an open and participatory environment and released under the Apache Software License. Apache Tomcat

powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations.

5. Java API for XML-Based RPC (JAX-RPC)

The Java API for XML based RPC (JAX-RPC) enables Java technology developers to build Web applications and Web Services incorporating XML based RPC functionality according to the SOAP (Simple Object Access Protocol) 1.1 specification. By using JAX-RPC, developers can rapidly achieve Web Services interoperability based on widely adopted standards and protocols.

6. NSClient

NSClient, or *Netsaint Windows Client*, is a plug-in that is used by Netsaint to monitor all kind of information on a Windows NT, 2000 or XP server. The following information can be checked using NSClient: CPU load (single or multi-processors machines), memory load, disk space, service state, process state, system uptime, file date & time, etc. In addition, any performance counter can be measured, such as information related to Exchange server, SQL server, etc.

7. NSClient4j

NSClient4j is a Java API for accessing Windows Performance Monitor counters. It uses the NSClient Windows Service.

8. Command Line Process Viewer/Killer/Suspender for Windows NT/2000/XP

It is a command line utility to view, kill, suspend, or set the priority of processes, perhaps from a batch file.

Appendix D: WSAC Proof-of-Concept Prototype Implementation & Use Cases

The first section of the report describes the development of the prototype, which is designed to demonstrate the feasibility of the proposed system. The second section describes the implementation of the prototype, which is based on the WSAC system. The third section describes the use cases for the prototype, which are based on the WSAC system. The fourth section describes the results of the prototype, which are based on the WSAC system. The fifth section describes the conclusions of the prototype, which are based on the WSAC system.

The first use case is to demonstrate the feasibility of the proposed system. The second use case is to demonstrate the implementation of the prototype. The third use case is to demonstrate the use cases for the prototype. The fourth use case is to demonstrate the results of the prototype. The fifth use case is to demonstrate the conclusions of the prototype.

| Use Case | Description |
|----------|---|
| 1 | Demonstrate the feasibility of the proposed system. |
| 2 | Demonstrate the implementation of the prototype. |
| 3 | Demonstrate the use cases for the prototype. |
| 4 | Demonstrate the results of the prototype. |
| 5 | Demonstrate the conclusions of the prototype. |

Appendix D: WSAC Proof-of-Concept Prototype Implementation and Use Cases

1. The Prototype Implementation

For simplicity, each WS that was discussed in the prototype architecture in chapter 6 is implemented as a separate class, which implements a set of methods. Some of the methods of each WS are published to be consumed by other WS in the prototype, such methods are *public* methods; however, some other methods are not published as they are used by its WS in processing the received requests, such methods are *private* methods. When a BWS is published into a public service registry, the published interface includes the description of only the public methods. Each of the following points lists the methods included in the implementation of each WS of the prototype as well as the class structure of the WS:

1.1 Sensor

The sensor class extends *Thread* in order to implement the infinite AC loop, as after starting each AC round, the active sensor thread waits until the HWS finishes the processing of the submitted AC request; and after that, another AC round is initiated. In addition, the class has a set of attributes that hold the information that will be published in the private UDDI database; this information includes *portType*, *targetNamespace*, *endpoint*, and *resourceType*. The following table lists the methods implemented by sensor WS together with the type and the purpose of each method:

Table 10 List of sensor WS methods

| Method | Type | Purpose |
|------------------|---------|--|
| publishSensor() | Private | Publishing the interface of sensor WS into the private UDDI database. |
| startAutonomic() | Private | Starting the AC infinite loop for monitoring and managing the OS continuously. |
| Sense() | Public | This method is invoked by HWS in order to measure CPU utilization. Therefore, the method communicates with NSClient service. |
| getLogsFile() | Public | This method is invoked by HWS in order to retrieve the path, in which, the OS stores its logs. |
| getAllServices() | Public | This method is invoked by HWS in order to retrieve a list of the running services. |

| | | |
|-------------------------|---------|---|
| | | Therefore, the method communicates with the OS native APIs. |
| getAllProcesses() | Public | This method is invoked by HWS in order to retrieve a list of all the active processes together with the type and CPU utilization of each process. Therefore, this method communicates with the process utility. |
| getEffectorInfo() | Public | This method is invoked by HWS to retrieve the information of the effector WS, through which, management actions can be applied to the OS. |
| getMetrics() | Public | This method is invoked by HWS in order to retrieve a list of the OS metrics that should be monitored. In our prototype, only CPU is made available to be consumed by HWS. |
| getAutonomicBehaviour() | Private | This method is invoked by sense() method during each AC round. The method is responsible for locating and establishing a management relationship with HWS. In addition, it checks the health of the located HWS indirectly. As mentioned in the previous chapter, the method established the management relationship with HWS by invoking the entry method in the latter. |
| checkHWSHealth() | Private | This method periodically checks the health of located HWS, such that at the beginning of each AC round getAutonomicBehaviour() method consults the method whether the located HWS is still alive and responsive or not. |

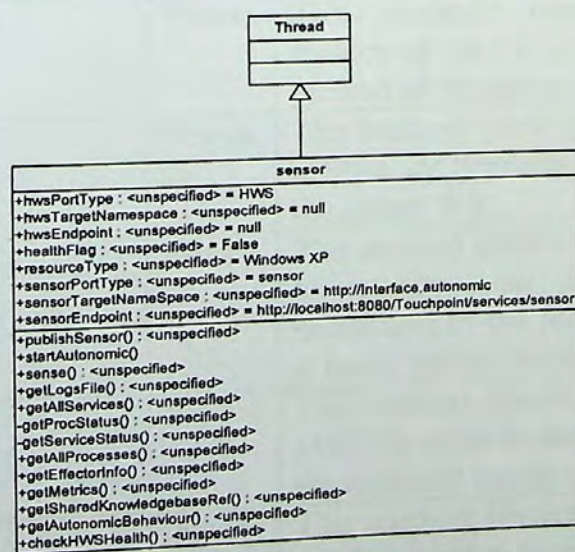


Figure 43 Sensor WS Class Structure

1.2 Healing Web Service

The HWS class extends *Thread* in order to check the health of each of the MAPE-WS, such that after invoking each of those WS separately, the active thread of HWS waits for a specific period of time; if HWS receives the expected response with that period, it will invoke the next MAPE-WS in the sequence; otherwise, HWS will locate another MAPE-WS. In addition, the class has a set of attributes that hold the information that will be published in the private UDDI database; this information includes *portType*, *targetNamespace*, *endpoint*, and *resourceType*, which is supported by HWS. The following table lists the methods implemented by HWS together with the type and the purpose of each method:

Table 11 List of HWS methods

| Method | Type | Purpose |
|-----------------------|---------|---|
| publishHWS() | Private | Publishing the interface of HWS into the private UDDI database. |
| heal() | Public | This method is the entry method that is invoked by <code>getAutonomicBehaviour()</code> method of the sensor WS. When the method is invoked it receives significant information from the sensor, such <code>portType</code> , <code>targetNamespace</code> , <code>endpoint</code> and reference to the knowledge base of the OS. |
| findEffector() | Private | This methods retrieves the information about the effector WS from the sensor WS by invoking the <code>getEffectorInfo()</code> method of the sensor WS. |
| getAvailableMetrics() | Private | This methods retrieves the significant metrics of OS by invoking the <code>getMetrics()</code> method of the sensor WS. |
| collectMetrics() | Private | This methods retrieves the value of received metrics by invoking the <code>sense()</code> method of the sensor WS. |
| orchestrate MAPE() | Private | This method locates the MAPE-WS in order to monitor and manage the OS. As mentioned in the previous chapter, locating is based on the <code>resourceType</code> criterion. |
| invokeMWS() | Private | This method invokes the entry method of MWS in order to assess the status based on the retrieved values of metrics. |
| invokeAWS() | Private | This method invokes the entry method of AWS in order to analyze the response of MWS. |
| invokePWS() | Private | This method invokes the entry method of PWS in order validate the actions |

| | | |
|----------------------|---------|---|
| invokeEWS() | Private | recommended by AWS. This method invokes the entry method of EWS in order specify the commands to be executed on the OS based on the approved action by PWS. |
| getActiveProcesses() | Public | This method is invoked by AWS in order to retrieve information of the active processes. Therefore, when this method is invoked it invokes the getAllProcesses () method of the sensor WS. |
| getRunningServices() | Public | This method is invoked by AWS in order to retrieve information of the running services. Therefore, when this method is invoked it invokes the getAllServices() method of the sensor WS. |
| dispatch() | Public | This method is invoked by EWS to execute some command on the OS. |
| invokeEffector() | Private | This method is invoked by dispatch() method to pass the command to the effector WS. |
| getLogsFile() | Public | This method is invoked by AWS in order to gain access to the system logs of the OS to perform a root-cause analysis. Therefore, when this method is invoked it invokes the getLogsFile() method of the sensor WS. |
| checkHealth() | Public | This method is invoked by the sensor WS in order to check the health of HWS. |

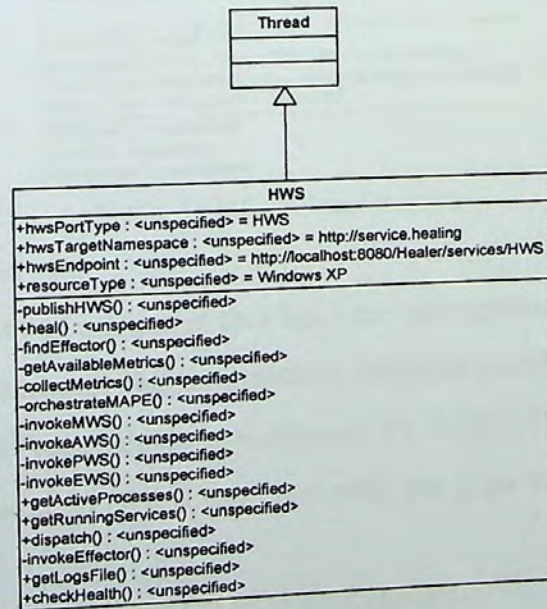


Figure 44 HWS Class Structure

1.3 Monitoring Web Service

The MWS class has a set of attributes that hold the information that will be published in the private UDDI database; this information includes *portType*, *targetNamespace*, *endpoint*, and *resourceType*, which is supported by MWS. The following table lists the methods implemented by MWS together with the type and the purpose of each method:

Table 12 List of MWS Methods

| Method | Type | Purpose |
|----------------|---------|--|
| publishMWS() | Private | Publishing the interface of MWS into the private UDDI database. |
| monitor() | Public | This method is the entry method of MWS that is invoked by invokeMWS() method of the HWS. When the method is invoked it receives the values of the metrics collected by HWS from the OS together with the reference to the knowledge base of the OS. The response of this method is reporting the behaviour of the OS as either normal or abnormal in terms of the CPU utilization. |
| checkMetrics() | Private | This method accesses the knowledge base of the OS in order to compare the reported values of metrics against the recorded thresholds in the knowledge base. |

| MWS |
|---|
| +mwsPortType : <unspecified> = MWS |
| +mwsTargetNamespace : <unspecified> = http://service.monitoring |
| +mwsEndpoint : <unspecified> = http://localhost:8080/Monitor/services/MWS |
| +resourceType : <unspecified> = Windows XP |
| -publishMWS() : <unspecified> |
| +monitor() : <unspecified> |
| -checkMetrics() : <unspecified> |

Figure 45 MWS Class Structure

1.4 Analyzing Web Service

The AWS class has a set of attributes that hold the information that will be published in the private UDDI database; this information includes *portType*, *targetNamespace*, *endpoint*, and *resourceType*, which is supported by AWS. The following table lists the methods implemented by AWS together with the type and the purpose of each method:

Table 13 List of AWS Methods

| Method | Type | Purpose |
|--------------|---------|---|
| publishAWS() | Private | Publishing the interface of AWS into the private UDDI database. |
| analyze() | Public | This method is the entry method of AWS |

| | | |
|-----------------------|---------|--|
| | | that is invoked by invokeAWS() method of the HWS. When the method is invoked it receives the response of MWS together with the reference to the knowledge base of the OS. The response of this method is one of the following: (1) no problem was found, (2) recommended healing action for the detected problem (e.g. a process to kill or a service to start) or (3) no healing action can be recommended to solve the detected problem. |
| getLogs() | Private | This method accesses the system logs of the OS by invoking the getLogsFile() method of HWS. |
| checkCapability() | Private | This method checks the symptoms database to check whether the reported situation is common or not. |
| getProcesses() | Private | This method retrieves the information of all the active processes by invoking the getActiveProcesses() method of HWS. |
| getServices() | Private | This method retrieves a list of all the running services by invoking the getRunningServices() method of HWS. |
| analyzeProcesses() | Private | This method analyzes the retrieved processes and reports the name and type of the process that utilizes the CPU the most. |
| analyzeServices() | Private | This method analyzes the retrieved services to find if there is a missing service. |
| analyzeLogs() | Private | This method locates the most recent record of the log file to discover the cause of the problem. |
| findSolution() | Private | This method locates the directive for solving the reported problem. |
| checkSymptomsDB() | Private | This method returns the directive for a reported problem by the findSolution() method. |
| solve() | Private | |
| checkKnowledgeBases() | Private | This method checks the knowledge base of the OS to know the purpose of the important services. |
| getTime() | Private | This method retrieves the current time from the local environment of AWS. |
| convertTime() | Private | This method formats the retrieved time to match the format, by which, time is recorded in the log file. |

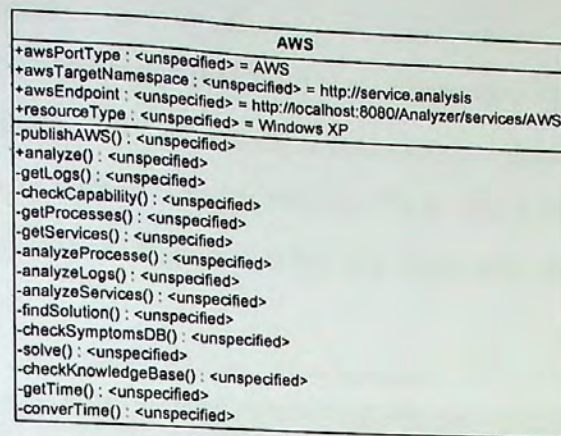


Figure 46 AWS Class Structure

1.5 Planning Web Service

The PWS class has a set of attributes that hold the information that will be published in the private UDDI database; this information includes *portType*, *targetNamespace*, *endpoint*, and *resourceType*, which is supported by PWS. The following table lists the methods implemented by PWS together with the type and the purpose of each method:

Table 14 List of PWS Methods

| Method | Type | Purpose |
|------------------------|---------|---|
| publishPWS() | Private | Publishing the interface of PWS into the private UDDI database. |
| Plan() | Public | This method is the entry method of PWS that is invoked by invokePWS() method of the HWS. When the method is invoked it receives the response of AWS together with the reference to the knowledge base of the OS. The response of this method is either validating the recommended healing action by AWS or requesting another action. |
| checkSuggestedAction() | Public | This method accesses the knowledge base of the OS to validate the recommended action by AWS against the policies of the OS. |

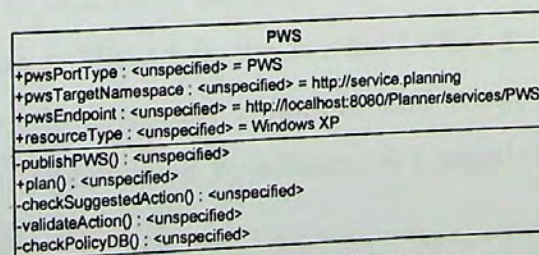


Figure 47 PWS Class Structure

1.6 Executing Web Service

The EWS class has a set of attributes that hold the information that will be published in the private UDDI database; this information includes *portType*, *targetNamespace*, *endpoint*, and *resourceType*, which is supported by EWS. The following table lists the methods implemented by EWS together with the type and the purpose of each method:

Table 15 List of EWS Methods

| Method | Type | Purpose |
|--------------|---------|--|
| publishEWS() | Private | Publishing the interface of EWS into the private UDDI database. |
| Execute() | Public | This method is the entry method of EWS that is invoked by invokeEWS() method of the HWS. When the method is invoked it receives the response of PWS together with the reference to the knowledge base of the OS. The response of this method is the command to be executed on the OS by HWS. |
| dispatch() | Private | This method translates the reported action into a command that can be executed on the OS. Moreover, the method schedule the execution of the command based on the execution policies of the OS. |

| EWS |
|--|
| +ewsPortType : <unspecified> = EWS |
| +ewsTargetNamespace : <unspecified> = http://service.execution |
| +ewsEndpoint : <unspecified> = http://localhost:8080/Executor/services/EWS |
| +resourceType : <unspecified> = Windows XP |
| -publishEWS() : <unspecified> |
| +execute() : <unspecified> |
| -dispatch() : <unspecified> |

Figure 48 EWS Class Structure

1.7 Effector

The effector class extends *Thread* in order to schedule the execution of commands as received from HWS. In addition, the class has a set of attributes that hold the information that will be published in the private UDDI database; this information includes *portType*, *targetNamespace*, *endpoint*, and *resourceType*. The following table lists the methods implemented by effector WS together with the type and the purpose of each method:

Table 16 List of effector WS Methods

| Method | Type | Purpose |
|-------------------|---------|---|
| publishEffector() | Private | Publishing the interface of the effector WS |

| | | |
|--------------|--------|--|
| runCommand() | Public | This method is invoked by the invokeEffector() method of HWS in order to execute a specific command on the OS. |
|--------------|--------|--|

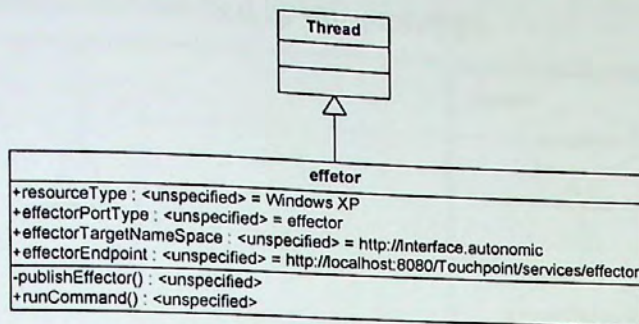


Figure 49 Effector WS Class Structure

2. The Prototype Use Cases

2.1 Publishing WS

Any WS of our prototype is published into the private UDDI by manually invoking the publishing method in each WS.

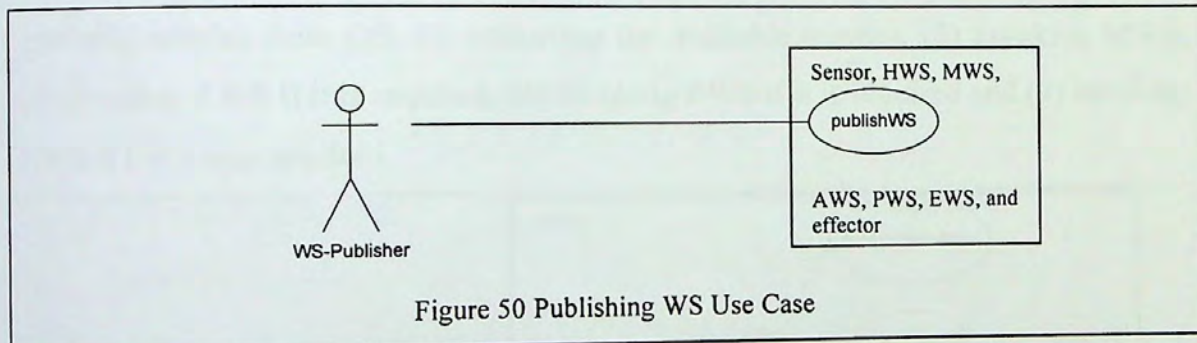


Figure 50 Publishing WS Use Case

2.2 Starting AC infinite loop

The AC infinite loop is started by the operator of the OS (e.g. OS administrator). Once the loop is started it keeps active as the OS is running.

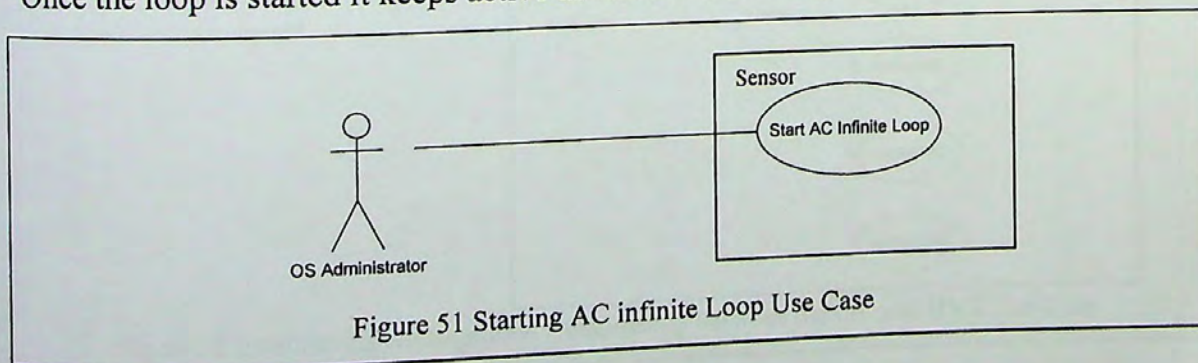


Figure 51 Starting AC infinite Loop Use Case

2.3 Locating HWS

The sensor WS of the OS ATP is responsible for locating an HWS that is specialized in healing the Windows XP resources; however, the sensor might fail to locate such HWS. This exception is not handled by our prototype.

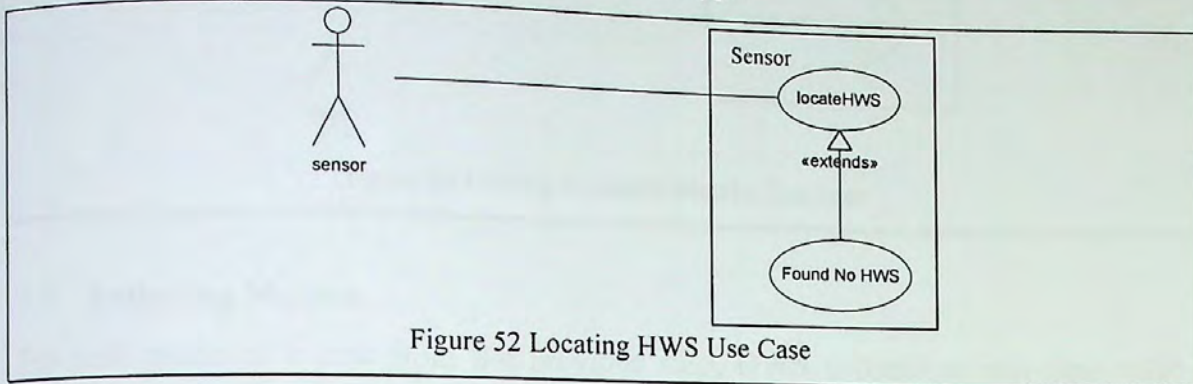


Figure 52 Locating HWS Use Case

2.4 Establishing a Management Relationship between the OS and HWS

Management relationship between the OS and HWS is established through the sensor WS, which invokes the heal entry method of HWS. As a result, of invoking the heal method, the HWS takes the following actions: (1) locating MAPE-WS, (2) getting the available metrics from OS, (3) collecting the available metrics, (4) invoking MWS, (5) invoking AWS if it is required, (6) invoking PWS if it is required and (7) invoking EWS if PWS was required.

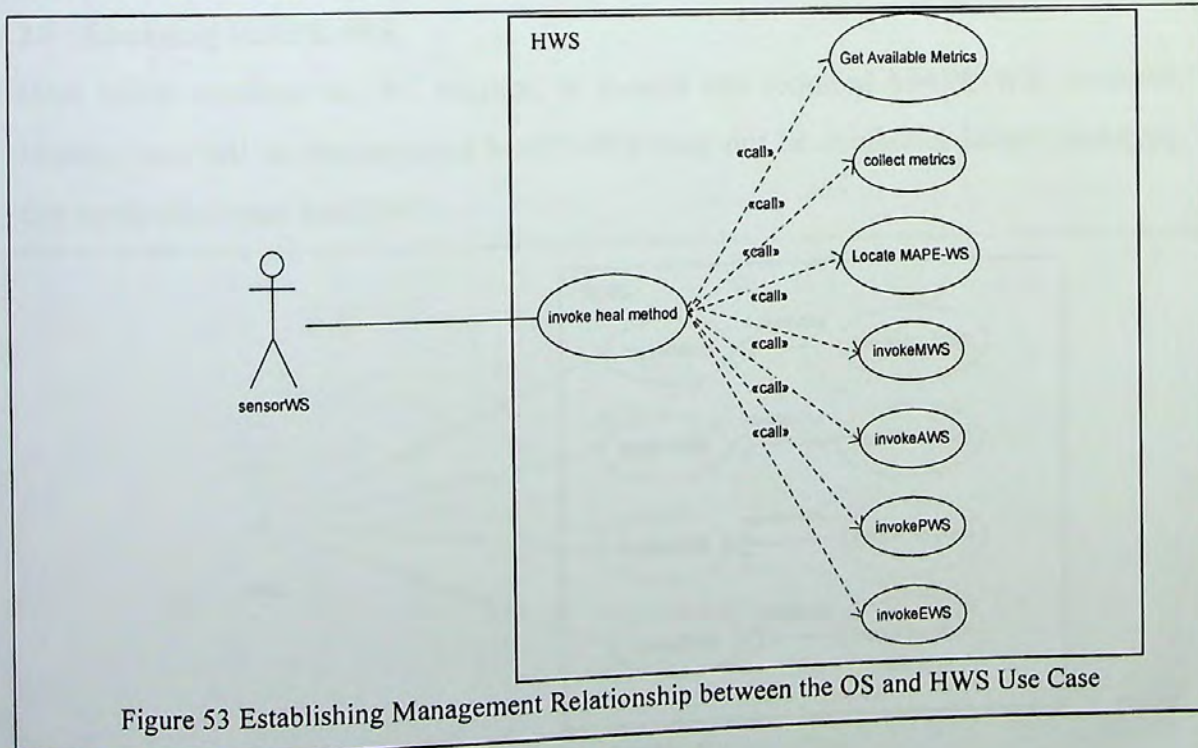


Figure 53 Establishing Management Relationship between the OS and HWS Use Case

2.5 Getting Available Metrics

HWS gets the available metrics in order to know the significant metrics of OS.

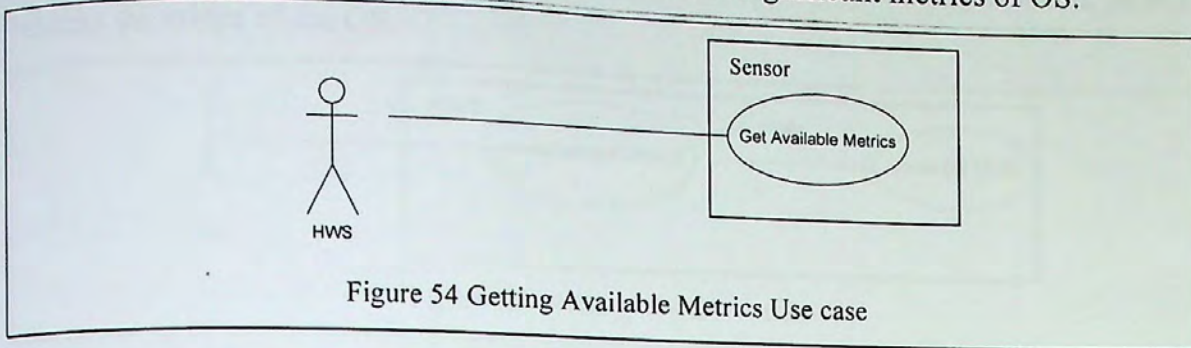


Figure 54 Getting Available Metrics Use case

2.6 Collecting Metrics

For each retrieved metric from the previous step, HWS collects its real-time value during each AC-round.

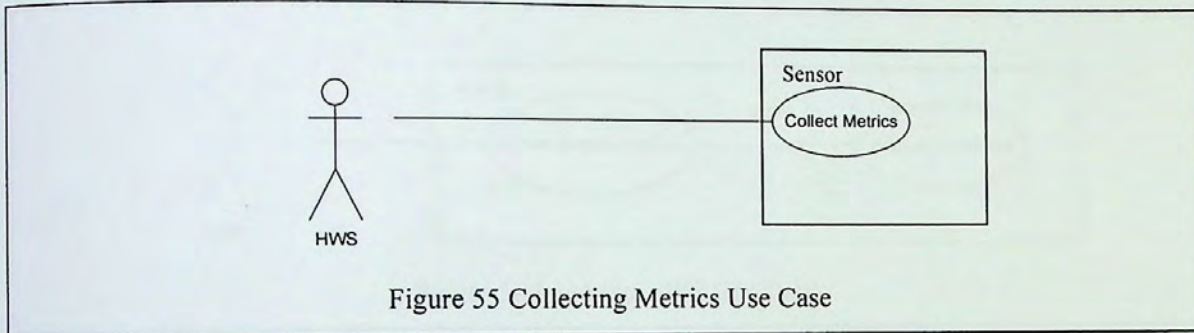


Figure 55 Collecting Metrics Use Case

2.7 Locating MAPE-WS

Once HWS receives an AC request, it locates the required MAPE-WS; however, locating may fail as the required MAPE-WS may not be available. In our prototype, this exception is not handled.

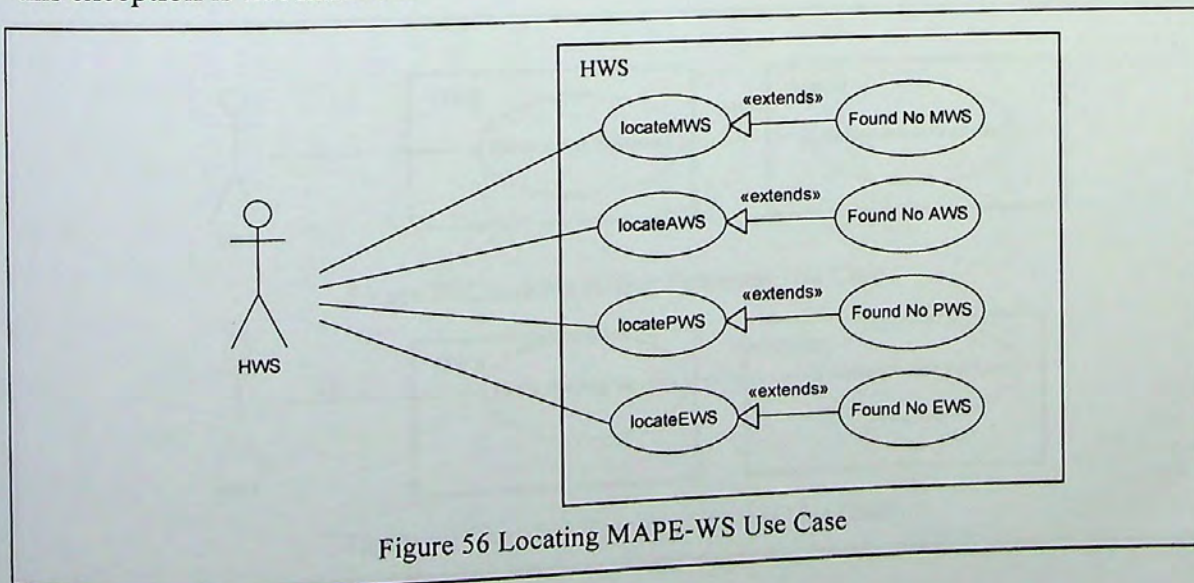


Figure 56 Locating MAPE-WS Use Case

2.8 Invoking MWS

MWS is invoked when HWS invokes the entry method of MWS; as a result, MWS checks the status of the OS.

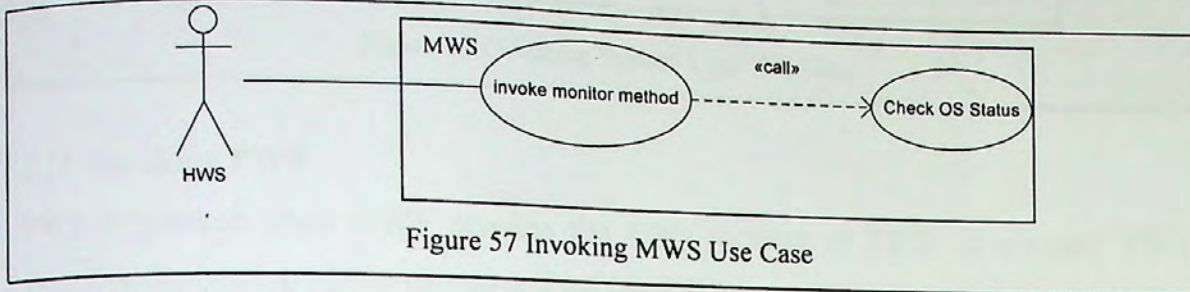


Figure 57 Invoking MWS Use Case

2.9 Invoking AWS

AWS is invoked when HWS invokes the entry method of MWS; as a result, AWS analyzes the situation of the OS.

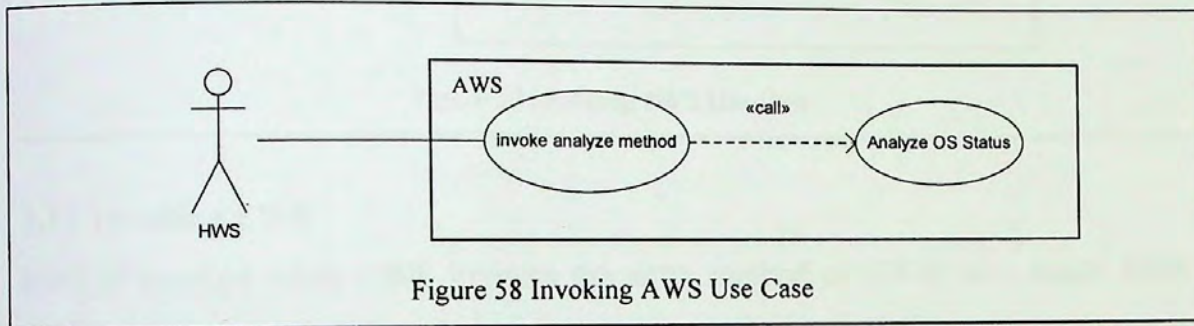


Figure 58 Invoking AWS Use Case

2.10 Analyzing OS Status

AWS analyzes the status of OS by one of the following approaches: (1) analyzing active processes, (2) analyzing starting services or (3) analyzing system logs. Choosing the analysis approach is based on the reported situation by HWS to AWS.

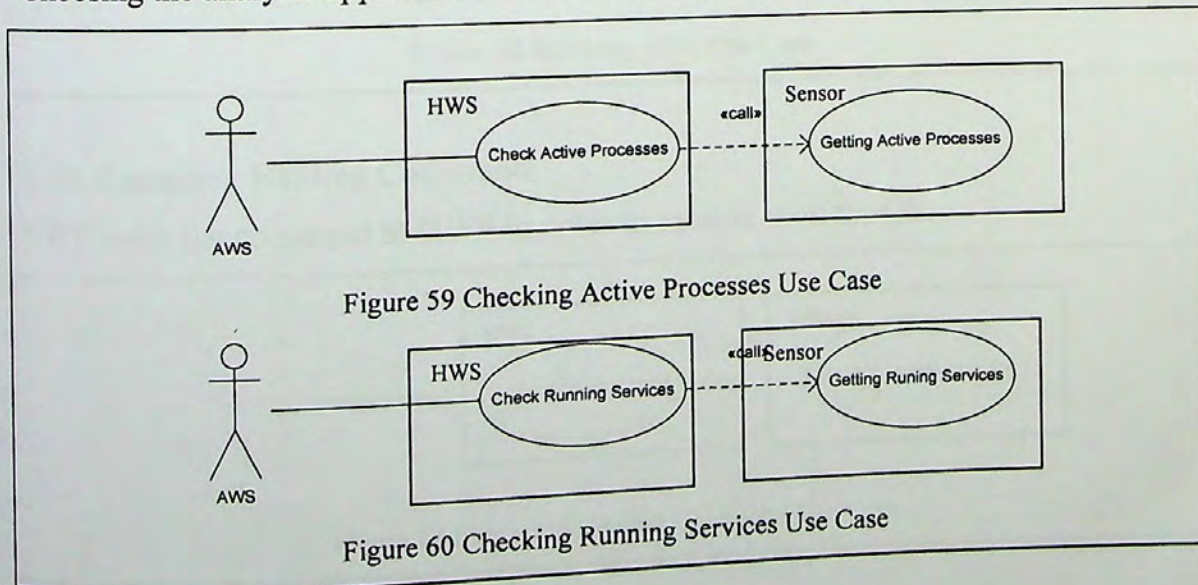


Figure 59 Checking Active Processes Use Case

Figure 60 Checking Running Services Use Case

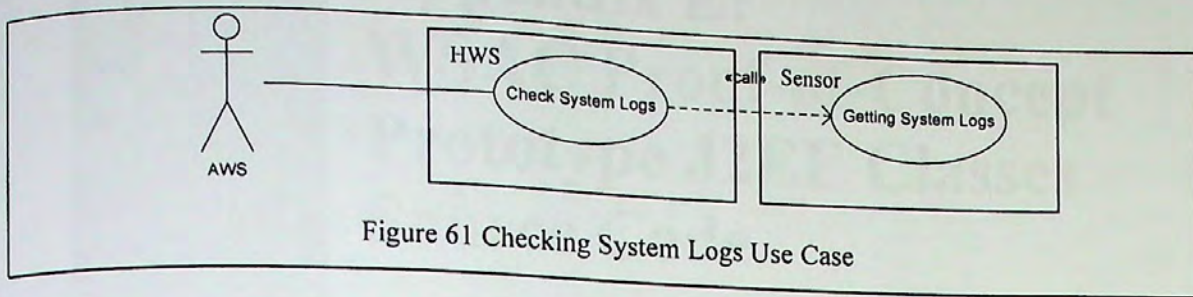


Figure 61 Checking System Logs Use Case

2.11 Invoking PWS

PWS is invoked when HWS invokes the entry method of PWS; as a result, PWS approves the healing action recommended by AWS.

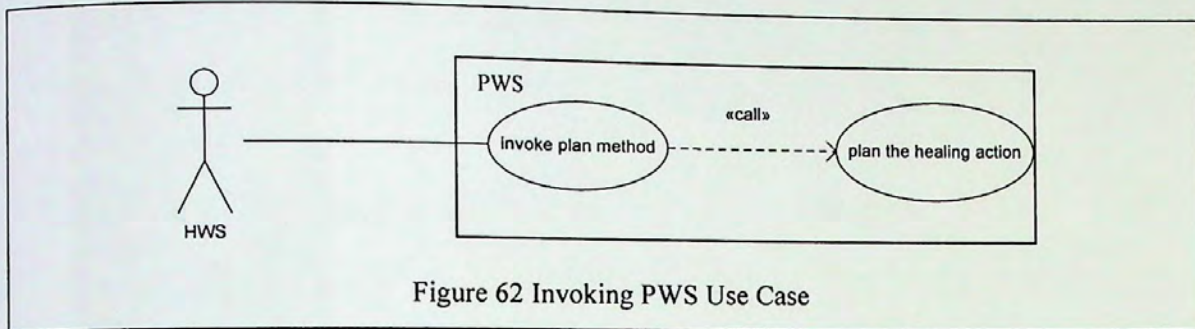


Figure 62 Invoking PWS Use Case

2.12 Invoking EWS

EWS is invoked when HWS invokes the entry method of EWS; as a result, EWS specifies the healing commands.

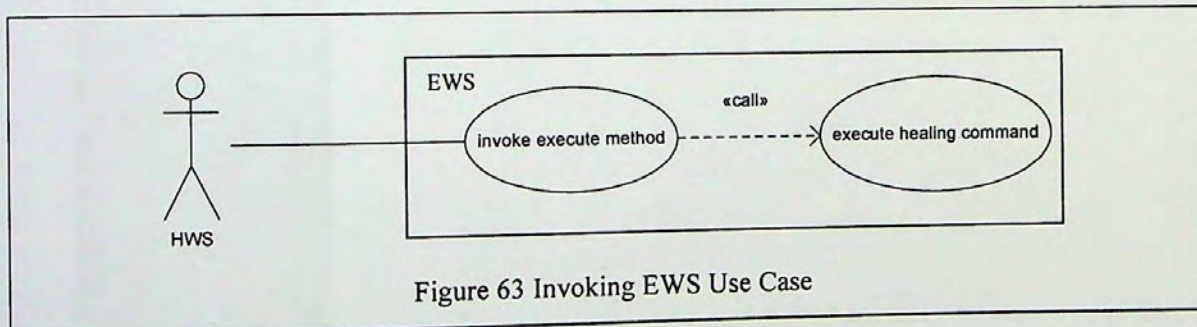


Figure 63 Invoking EWS Use Case

2.13 Executing Healing Command

EWS sends the command to HWS in order to execute it on the OS.

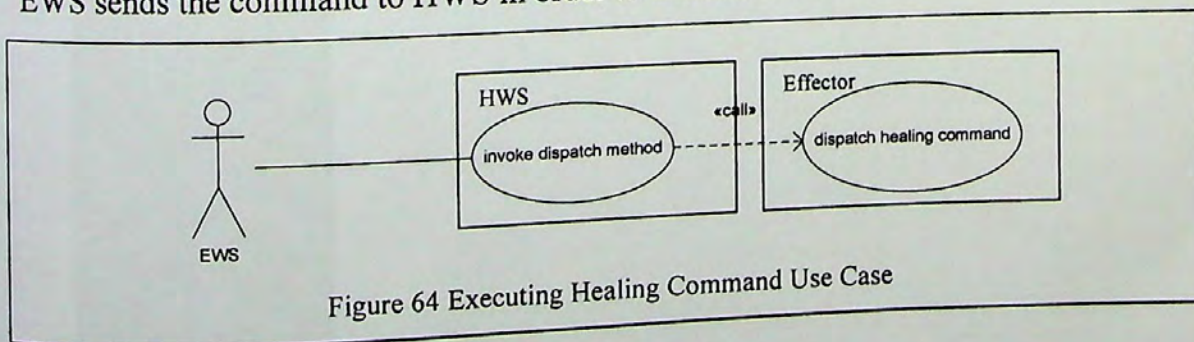


Figure 64 Executing Healing Command Use Case

**Appendix E:
WSAC Proof-of-Concept
Prototype J2EE Classes
Source Code**

Appendix E: WSAC Proof-of-Concept Prototype J2EE Classes Source Code

1. Sensor Web Service J2EE Class

```

package autonomic.Interface;

import org.nsclient4j.*;
import java.lang.String;
import java.lang.Exception;
import java.sql.*;
import java.io.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

//-----Sensor Web Service Class-----//
public class sensor extends Thread
{
    //Variables Declaration//
    //HWS Web Service UDDI Information
    public String hwsPortType = "HWS";
    public String hwsTargetNamespace = null;
    public String hwsEndpoint = null;
    //Sensor Web Service UDDI Information
    public String resourceType = "Windows XP";
    public String sensorPortType = "sensor";
    public String sensorTargetNamespace =
"http://Interface.autonomic";
    public String sensorEndpoint =
"http://localhost:8080/Touchpoint/services/sensor";
    //Health Flag of HWS Service
    public boolean healthFlag = false;
    //Publishing Sensor WS into Private UDDI
    public String publishSensor() throws Exception
    {
        String publishingResult = null;
        String dataSourceName = "PrivateUDDI_DNS";
        String dbURL = "jdbc:odbc:" + dataSourceName;
        try
        {
            String sql = null;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
dbURL, "", "");
            Statement s = con.createStatement();
            sql = "INSERT INTO UDDI_Table " + "VALUES
('"+sensorTargetNamespace+"',
 '"+sensorPortType+"', '"+sensorEndpoint+"', '"+resourceType+'')";
            s.executeUpdate(sql);
            s.close();

```

```

        con.close();
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    publishingResult = "Sensor Has Been Published
Successfully";
    return publishingResult;
}
////////////////////////////////////
//Starting Autonomic Infinite Loop
public void startAutonomic()throws Exception
{
    int turn = 1;
    System.out.println("Starting Autonomic Infinite
Loop...");
    try
    {synchronized(this){wait(2000);}}
    catch (InterruptedException e)
    {System.out.println("Error");}
    while (true)
    {
        System.out.println("Autonomic Round Number: " +
String.valueOf(turn));
        getAutonomicBehaviour();
        turn++;
        //Waiting 10 Seconds between Consecutive Autonomic
Rounds
        try
        {synchronized(this){wait(10000);}}
        catch (InterruptedException e)
        {System.out.println("Error");}
    }
}
////////////////////////////////////
//Starting Autonomic Round X
public String getAutonomicBehaviour()throws Exception
{
    String sharedKnowledgebaseRef = null;
    String healingAction = null;
    String targetNameSpace = null;
    String endpoint = null;
    String UDDI = null;
    System.out.println("Finding a Healing Web Service...");
    String dataSourceName = "PrivateUDDI_DNS";
    String portTypeName = "HWS";
    String patternStr1 = " ";
    String dbURL = "jdbc:odbc:" + dataSourceName;
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    sharedKnowledgebaseRef = getSharedKnowledgebaseRef ();
    UDDI = "sensor" + " " + "http://Interface.autonomic" + "
" + "http://localhost:8080/Touchpoint/services/sensor";
    try
    {
        if (checkHWSHealth()==null&&healthFlag==false)
        {

```



```

        System.out.println("Getting Info from UDDI");
        //Locating HWS From UDDI-----
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection(
dbURL, "", "");

        Statement st = con.createStatement();
        //Selection Statement-----
        st.executeQuery("SELECT * FROM UDDI_Table
WHERE portTypeName = '"+portTypeName+"' AND Specialization =
'+resourceType+'");

        ResultSet rs = st.getResultSet();
        if (rs != null)
            while( rs.next())
            {
                targetNamespace =
                endpoint = rs.getString(3);
            }
        st.close();
        con.close();
        hwsTargetNamespace = targetNamespace;
        hwsEndpoint = endpoint;
        healthFlag = true;
    }
    //Invoking Entry Method on HWS-----
    ServiceFactory serviceFactory =
ServiceFactory.newInstance();
    Service service = serviceFactory.createService(new
QName(hwsTargetNamespace, portTypeName));
    Call call = (Call) service.createCall();
    call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
    "");
    call.setProperty(call.OPERATION_STYLE_PROPERTY,
    "wrapped");
    call.setTargetEndpointAddress(hwsEndpoint);
    call.setPortTypeName(new QName(hwsTargetNamespace,
portTypeName));
    call.setOperationName(new QName(hwsTargetNamespace,
"heal"));
    call.addParameter("DB_Ref", stringQName,
ParameterMode.IN);
    call.addParameter("UDDI", stringQName,
ParameterMode.IN);
    call.addParameter("Parser", stringQName,
ParameterMode.IN);
    call.setReturnType(stringQName);
    //Service Invocation
    Object[] inParams = new Object[3];
    inParams [0] = sharedKnowledgebaseRef;
    inParams [1] = UDDI;
    inParams [2] = patternStr1;
    healingAction = (String) call.invoke(inParams);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    System.out.println(healingAction);

```

```

//Result of Autonomic Round X-----
return healingAction;
}
////////////////////////////////////
//Retrieving Values of OS Counters
public String sense (String metricType) throws Exception
{
    String sensedMetric = null;
    try
    {
        NSClient4j c = new NSClient4j("127.0.0.1", 1248);
        if (metricType.equals("CPU"))
            sensedMetric = c.getCPUUsage();
        if (metricType.equals("Memory"))
            sensedMetric =
c.getPerfMonCounter("\\Memory\\Available MBytes");
    }
    catch (Exception e)
    {sensedMetric = "Metric Could Not Be Collected!";}
    return sensedMetric;
}
////////////////////////////////////
//Retrieving a List of the Active Processes
public String getAllProcesses() throws Exception
{
    String allProcesses = null;
    String s = null;
    String arrangedProcesses = null;
    PrintStream outFile = null;
    int i = 0;
    String newline = System.getProperty("line.separator");
    try
    {
        Process p = Runtime.getRuntime().exec("process");
        BufferedReader stdInput = new BufferedReader(new
InputStreamReader(p.getInputStream()));
        while ((s = stdInput.readLine()) != null)
        {allProcesses = allProcesses + s + newline;}
        i = allProcesses.indexOf("Idle");
        arrangedProcesses = allProcesses.substring(i);
        outFile = new PrintStream(new
FileOutputStream("F:\\processes.txt"));
        outFile.println(arrangedProcesses);
        outFile.close();
    }
    catch (Exception e)
    {System.out.print("Active Processes Could Not Be
Retreived!");}
    return "F:\\processes.txt";
}
////////////////////////////////////
//Retrieving a List of the Running Services
public String getAllServices() throws Exception
{
    String runningServices = null;
    String allServices = null;
    String arrangedServicesTemp = null;
    String arrangedServices = null;
    String s = null;
}

```

```

PrintStream outFile = null;
String newline = System.getProperty("line.separator");
int i = 0;
int l = 0;
try
{
    Process p = Runtime.getRuntime().exec("net start");
    BufferedReader stdInput = new BufferedReader(new
InputStreamReader(p.getInputStream()));
    while ((s = stdInput.readLine()) != null)
    {allServices = allServices + s + newline;}
    i = allServices.indexOf('\n');
    arrangedServicesTemp = allServices.substring(i+2);
    l = arrangedServicesTemp.lastIndexOf("The");
    arrangedServices =
arrangedServicesTemp.substring(0,l-4);
    outFile = new PrintStream(new
FileOutputStream("F:\\services.txt"));
    outFile.println(arrangedServices);
    outFile.close();
}
catch(Exception e)
{System.out.print("Running Services Could Not Be
Retreived!");}
runningServices = "F:\\services.txt";
return runningServices;
}
////////////////////////////////////
//Checking the Status of Process X
public String getProcStatus(String Process) throws Exception
{
    boolean procStatus = false;
    String status = null;
    try
    {
        NSClient4j c = new NSClient4j("127.0.0.1", 1248);
        procStatus = c.isProcessUp(Process);
        if (procStatus == true) status = "Process " +
Process + " Is Running";
        else status = "Process " + Process + " Isn't
Running";
    }
    catch (Exception e)
    {System.err.println("Exception: " + e);}
    return status;
}
////////////////////////////////////
//Checking the Status of Service X
public String getServiceStatus(String Service) throws Exception
{
    boolean servStatus = false;
    String status = null;
    try
    {
        NSClient4j c = new NSClient4j("127.0.0.1", 1248);
        servStatus = c.isServiceUp(Service);
        if (servStatus == true)
            status = "Service " + Service + " Is
Started";
    }
}

```

```

else
    status = "Service " + Service + " Is
Stopped";
}
catch (Exception e)
{System.err.println("Exception: " + e);}
return status;
}
////////////////////////////////////
//Exposing Reference Path of the OS Log File
public String getLogsFile()
{
    String logsFileName = null;
    logsFileName = "F:\\myLogs\\log1.txt";
    return logsFileName;
}
////////////////////////////////////
//Exposing UDDI Information of the Effector Web Service
public String getEffectorInfo()
{
    String EffectorInfo = null;
    EffectorInfo = "effector http://Interface.autonomic
http://localhost:8080/Touchpoint/services/effector";
    return EffectorInfo;
}
////////////////////////////////////
//Exposing a List of all the Available Metrics
public String getMetrics() throws Exception
{
    String availableMetricsTemp = null;
    String availableMetrics = null;
    String dataSourceName = getSharedKnowledgebaseRef ();
    String dbURL = "jdbc:odbc:" + dataSourceName;
    try
    {
        //Getting Information from Shared Knowledge Base---
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection(
dbURL, "", "");
        Statement st = con.createStatement();
        //Selection Statement
        st.executeQuery("SELECT MetricID FROM Metrics");
        ResultSet rs = st.getResultSet();
        if (rs != null)
            while( rs.next())
                {availableMetricsTemp = availableMetricsTemp
+ " "+ rs.getString(1);}
        st.close();
        con.close();
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    int index = 0;
    index = availableMetricsTemp.indexOf(" ");
    availableMetrics =
availableMetricsTemp.substring(index+1);
}

```

```

return availableMetrics;
//return null;
}
////////////////////////////////////
//Exposing Reference Path of the OS Shared Knowledge Base
public String getSharedKnowledgebaseRef ()
{return ("SharedKnowledgeBase_DNS");}
////////////////////////////////////
//Checking the Health of HWS
public String checkHWSHealth() throws Exception
{
String hwsHealth = null;
QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
try
{
ServiceFactory serviceFactory =
ServiceFactory.newInstance();
Service service = serviceFactory.createService(new
QName(hwsTargetNamespace, hwsPortType));
Call call = (Call) service.createCall();
call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
");
call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
call.setTargetEndpointAddress(hwsEndpoint);
call.setPortTypeName(new QName(hwsTargetNamespace,
hwsPortType));
call.setOperationName(new QName(hwsTargetNamespace,
"checkHealth"));
call.removeAllParameters();
call.setReturnType(stringQName);
//Service Invocation
hwsHealth = (String) call.invoke(null);
}
catch(Exception e)
{hwsHealth = null;}
return hwsHealth;
}
}

```

Listing 7 Sensor Web Service J2EE Class

2. HWS J2EE Class

```

package healing.service;

import java.lang.String;
import java.lang.Exception;
import java.sql.*;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

//-----HWS Class-----//
public class HWS extends Thread

```

```

{
    //Variables Declaration//
    public String hwsPortType = "HWS";
    public String hwsTargetNamespace = "http://service.healing";
    public String hwsEndpoint =
"http://localhost:8080/Healer/services/HWS";
    public String resourceType = "Windows XP";
    //Publishing HWS into Private UDDI
    public String publishHWS() throws Exception
    {
        String publishingResult = null;
        String dataSourceName = "PrivateUDDI_DNS";
        String dbURL = "jdbc:odbc:" + dataSourceName;
        try
        {
            String sql = null;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con= DriverManager.getConnection(
dbURL, "", "");
            Statement s = con.createStatement();
            sql = "INSERT INTO UDDI_Table " + "VALUES
('"+hwsTargetNamespace+"',
 '"+hwsPortType+"', '"+hwsEndpoint+"', '"+resourceType+"')";
            s.executeUpdate(sql);
            s.close();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println("Error: " + e);
            e.printStackTrace();
        }
        publishingResult = "HWS Has Been Published Successfully";
        return publishingResult;
    }
    //Entry heal Method
    public String heal(String knowledgeBaseRef, String resUDDI,
String Parser) throws Exception
    {
        System.out.println("Healing Has Been Started...");
        String hAction = "";
        String mwsResult = null;
        String ewsResult = null;
        String mwsRecord = null;
        String awsRecord = null;
        String pwsRecord = null;
        String ewsRecord = null;
        String res_portTypeName = null;
        String res_targetNameSpace = null;
        String res_endpoint = null;
        String collectedMetrics = null;
        String availableMetrics = null;
        String action = null;
        String hwsUDDI = "HWS http://service.healing
http://localhost:8080/Healer/services/HWS";
        String patternStr2 = ": ";
        String [] resourceUDDI = resUDDI.split(Parser);
    }
}

```

```

res_portTypeName = resourceUDDI[0];
res_targetNameSpace = resourceUDDI[1];
res_endpoint = resourceUDDI[2];
String[] UDDI_Records = orchestrateMAPE().split(Parser);
mwsRecord = UDDI_Records[0] + " " + UDDI_Records[1];
awsRecord = UDDI_Records[2] + " " + UDDI_Records[3];
pwsRecord = UDDI_Records[4] + " " + UDDI_Records[5];
ewsRecord = UDDI_Records[6] + " " + UDDI_Records[7];
String[] mwsInfo = mwsRecord.split(Parser);
String[] awsInfo = awsRecord.split(Parser);
String[] pwsInfo = pwsRecord.split(Parser);
String[] ewsInfo = ewsRecord.split(Parser);
availableMetrics = getAvailableMetrics(res_portTypeName,
res_targetNameSpace, res_endpoint);
try
{
    synchronized(this)
    {wait(1000);}
    if (availableMetrics == null)
    {
        System.out.println("No Management Information
Has Been Received From the Resource!");
        System.out.println("Switching to Poll-
Mode....");
        QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service =
serviceFactory.createService(new QName(res_targetNameSpace,
res_portTypeName));
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
        call.setTargetEndpointAddress(res_endpoint);
        call.setPortTypeName(new
QName(res_targetNameSpace, res_portTypeName));
        call.setOperationName(new
QName(res_targetNameSpace, "startAutonomic"));
        call.removeAllParameters();
        call.invoke(null);
    }
    else
    {
        System.out.println("Starting With Push-
Mode....");
        System.out.println("Management Information
Has Been Received From the Resource");
        String[] res_metrics =
availableMetrics.split(Parser);
        int metricsListLength = res_metrics.length;
        System.out.println("Available Metrics on the
Resource are: ");
        for (int i=0; i<metricsListLength; i++)
            System.out.println(res_metrics[i]);
        for (int i=0; i<metricsListLength-1; i++)
        {

```

```

String status = null;
String s = "initial";
boolean flag = false;
collectedMetrics = collectMetrics
(res_metrics[i] , res_portTypeName, res_targetNameSpace,
res_endpoint);
Could Not Be Collected!"))
    if (collectedMetrics.equals("Metric
    {
        status = collectedMetrics;
        flag = true;
    }
    else
    {
        mwsResult =
        invokeMWS(collectedMetrics , res_metrics[i], knowledgeBaseRef,
        mwsInfo[0], mwsInfo[1]);

        synchronized(this)
        {wait(2000);}
        if (mwsResult == null)
        {
            System.out.println("Failed
Monitoring...Locating Another Monitor....");
        }
        else
        {
            System.out.println("Monitoring
Has Been Perfomed Correctly");
            String[] afterMWS =
            mwsResult.split(patternStr2);
            status = afterMWS[0];
            s = afterMWS[1];
            System.out.println("Result
of Monitoring: " + status);
        }
    }
    if (flag == true || s.equals("Analyze"))
    {
        System.out.println("An Analyzer
is Being Located to Analyze: " + status + " Situation...");
        String check = "start";
        while (check!=null)
        {
            String awsResult = null;
            String pwsResult = null;
            awsResult =
            invokeAWS(status, awsInfo[0], awsInfo[1], hwsUDDI, resUDDI,
            knowledgeBaseRef);
            synchronized(this)
            {wait(2000);}
            if (awsResult == null)
            {
                System.out.println("Failed
Analysis...Locating Another Analyzer....");
                check = null;
            }
            else
            {

```



```

Been Perfomed Correctly");
Analysis: " + awsResult);
(awsResult.equals("No Action Found!"))
Action Suggested By The Located AWS!";
"Another AWS Is Being Located";
(awsResult.equals("No Problem Was Found"))
= "Analysis Has Found No Problems";
null;

afterAWS = awsResult.split(" ");
afterAWS[0] + " " + afterAWS[1] + " " + afterAWS[2];;
System.out.println("A Planner is Being Located to Plan The
Execution of: " + action + " Solution");
= invokePWS(awsResult, pwsInfo[0], pwsInfo[1]);
synchronized(this){wait(2000);}
(pwsResult == null)
System.out.println("Failed Planning...Locating Another
Planner....");
check = null;
System.out.println("Planning Has Been Perfomed Correctly");
System.out.println("Result of Planning: " + pwsResult);
(pwsResult.equals(action))
System.out.println("The Suggested " + action + " Action by AWS
Will Be Executed...");
System.out.println("An Executor is Being Located to Execute The
Action");
System.out.println("Action Has Been Passed to EWS");
ewsResult = invokeEWS(pwsResult,ewsInfo[0], ewsInfo[1],
hwsUDDI, resUDDI);
synchronized(this){wait(2000);}
if (ewsResult == null)
{
System.out.println("Analysis Has
Result of
if
{
System.out.println("No
ewsResult =
check = null;
}
else
{
if
{
ewsResult
check =
}
else
{
String[]
action =
pwsResult
if
{
}
else
{
if
{
System.out.println("The Suggested " + action + " Action by AWS
Will Be Executed...");
System.out.println("An Executor is Being Located to Execute The
Action");
System.out.println("Action Has Been Passed to EWS");
ewsResult = invokeEWS(pwsResult,ewsInfo[0], ewsInfo[1],
hwsUDDI, resUDDI);
synchronized(this){wait(2000);}
if (ewsResult == null)
{

```

```

        System.out.println("Failed Execution...Locating Another
Executor....");
        check = null;}
    else
    {
        System.out.println("Execution Has Been Perfomed
Correctly");
        check = null;
    }
}

(pwsResult.equals("Find Another Process"))                                if
                                                                            {
        System.out.println("A Sysetm Process Cannot Be Killed....");
        System.out.println("Finding Another Process To Kill....");
                                                                            }
                                                                            }
                                                                            }
                                                                            }
                                                                            }
        else ewsResult = s;
    }
    hAction = ewsResult;
}
catch (InterruptedException e)
{
    System.out.println("Error");
}
return hAction;
}
//////////////////////////////////////
//Entry Getting Information about the Effector Web Service
public String findEffector(String portTypeName, String
targetNameSpace, String endpoint)
{
    String effector = null;
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(targetNameSpace, portTypeName));
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
        "");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
        "wrapped");
        call.setTargetEndpointAddress(endpoint);
        call.setPortTypeName(new QName(targetNameSpace,
portTypeName));
        call.setOperationName(new QName(targetNameSpace,
"getEffectorInfo"));
        call.removeAllParameters();
        call.setReturnType(stringQName);
    }
}

```

```

        effector = (String) call.invoke(null);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    return effector;
}
////////////////////////////////////
//Getting A List of the Available Metrics
public String getAvailableMetrics(String portTypeName, String
targetNameSpace, String endpoint)
{
    String availableMetrics = null;
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(targetNameSpace, portTypeName));
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
        "");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
        "wrapped");
        call.setTargetEndpointAddress(endpoint);
        call.setPortTypeName(new QName(targetNameSpace,
portTypeName));
        call.setOperationName(new QName(targetNameSpace,
"getMetrics"));
        call.removeAllParameters();
        call.setReturnType(stringQName);
        //Service Invocation
        availableMetrics = (String) call.invoke(null);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    return availableMetrics;
}
////////////////////////////////////
//Collecting Values of the Available Metrics
public String collectMetrics(String metricToCollect, String
portTypeName, String targetNameSpace, String endpoint)
{
    String Comment1 = null;
    if (metricToCollect.equals("CPU"))
        Comment1 = "CPU Utilization: ";
    if (metricToCollect.equals("Memory"))
        Comment1 = "Available MBytes in Memory: ";
    String collectedMetrics = null;
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try

```

```

    {
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        QName targetNameSpace, portTypeName);
        Service service = serviceFactory.createService(new
Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
        call.setTargetEndpointAddress(endpoint);
        call.setPortTypeName(new QName(targetNameSpace,
portTypeName));
        call.setOperationName(new QName(targetNameSpace,
"sense"));
        call.addParameter("metric", stringQName,
ParameterMode.IN);
        call.setReturnType(stringQName);
        //Service Invocation
        Object[] inParams = new Object[1];
        inParams [0] = metricToCollect;
        collectedMetrics = (String) call.invoke(inParams);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    if (collectedMetrics.equals("Metric Could Not Be
Collected!"))
        System.out.println("Metrics Could Not Be
Collected!");
    else
        System.out.println("Current Status of " + Comment1
+ " " + collectedMetrics);
    return collectedMetrics;
}
////////////////////////////////////
//Orchestrating MAPE Cycle
public String orchestrateMAPE() throws Exception
{
    System.out.println("Orchestrating MAPE Cycle...");
    String dataSourceName = "PrivateUDDI_DNS";
    String portTypeName1 = "MWS";
    String portTypeName2 = "AWS";
    String portTypeName3 = "PWS";
    String portTypeName4 = "EWS";
    String targetNameSpace1 = null;
    String endpoint1 = null;
    String targetNameSpace2 = null;
    String endpoint2 = null;
    String targetNameSpace3 = null;
    String endpoint3 = null;
    String targetNameSpace4 = null;
    String endpoint4 = null;
    String UDDI = null;
    String resourceType = "Windows XP";
    String dbURL = "jdbc:odbc:" + dataSourceName;
    try

```

```

    {
        //Getting Information from Private UDDI-----
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con1 = DriverManager.getConnection(
dbURL, "", "");
        Statement st1 = con1.createStatement();
        //Selection Statement
        st1.executeQuery("SELECT * FROM UDDI_Table WHERE
portTypeName = '"+portTypeName1+"' AND Specialization =
 '"+resourceType+"'");
        ResultSet rs1 = st1.getResultSet();
        if (rs1 != null)
            while( rs1.next())
            {
                targetNameSpace1 = rs1.getString(1);
                endpoint1 = rs1.getString(3);
            }
        UDDI = targetNameSpace1 + " " + endpoint1;
        st1.close();
        con1.close();
        Connection con2 = DriverManager.getConnection(
dbURL, "", "");
        Statement st2 = con2.createStatement();
        //Selection Statement
        st2.executeQuery("SELECT * FROM UDDI_Table WHERE
portTypeName = '"+portTypeName2+"' AND Specialization =
 '"+resourceType+"'");
        ResultSet rs2 = st2.getResultSet();
        if (rs2 != null)
            while( rs2.next())
            {
                targetNameSpace2 = rs2.getString(1);
                endpoint2 = rs2.getString(3);
            }
        UDDI = UDDI + " " + targetNameSpace2 + " "
+endpoint2;
        st2.close();
        con2.close();
        Connection con3 = DriverManager.getConnection(
dbURL, "", "");
        Statement st3 = con3.createStatement();
        //Selection Statement
        st3.executeQuery("SELECT * FROM UDDI_Table WHERE
portTypeName = '"+portTypeName3+"' AND Specialization =
 '"+resourceType+"'");
        ResultSet rs3 = st3.getResultSet();
        if (rs3 != null)
            while( rs3.next())
            {
                targetNameSpace3 = rs3.getString(1);
                endpoint3 = rs3.getString(3);
            }
        UDDI = UDDI + " " + targetNameSpace3 + " "
+endpoint3;
        st3.close();
        con3.close();
        Connection con4 = DriverManager.getConnection(
dbURL, "", "");
        Statement st4 = con4.createStatement();

```

```

        st4.executeQuery("SELECT * FROM UDDI_Table WHERE
portTypeName = '"+portTypeName4+"' AND Specialization =
 '"+resourceType+"'");
        ResultSet rs4 = st4.getResultSet();
        if (rs4 != null)
            while( rs4.next())
            {
                targetNameSpace4 = rs4.getString(1);
                endpoint4 = rs4.getString(3);
            }
        UDDI = UDDI + " " + targetNameSpace4 + " "
+endpoint4;
        st4.close();
        con4.close();
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    return UDDI;
}
////////////////////////////////////
//Invoking the Entry Metod of MWS
public String invokeMWS(String metricVal, String metricType,
String knowledgeBaseRef, String targetNameSpace, String endpoint)
{
    String mwsResponse = null;
    String portTypeName = "MWS";
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        //Remote Invocation-----
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(targetNameSpace, portTypeName));
        //Service Access-----
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
        call.setTargetEndpointAddress(endpoint);
        call.setPortTypeName(new QName(targetNameSpace,
portTypeName));
        call.setOperationName(new QName(targetNameSpace,
"monitor"));
        call.addParameter("MetricValue", stringQName,
ParameterMode.IN);
        call.addParameter("MetricType", stringQName,
ParameterMode.IN);
        call.addParameter("DB_Ref", stringQName,
ParameterMode.IN);
        call.setReturnType(stringQName);
        //Service Invocation
        Object[] inParams = new Object[3];
        inParams [0] = metricVal;

```

```

        inParams [1] = metricType;
        inParams [2] = knowledgeBaseRef;
        mwsResponse = (String) call.invoke(inParams);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    return mwsResponse;
}
//////////////////////////////////////
//Invoking the Entry Method of AWS
public String invokeAWS(String MWS_Output, String
targetNameSpace, String endpoint, String UDDI1, String UDDI2, String
knowledgeBase)
{
    String portTypeName = "AWS";
    String awsResponse = null;
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        //Remote Invocation-----
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(targetNameSpace, portTypeName));
        //Service Access
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
        "");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
        "wrapped");
        call.setTargetEndpointAddress(endpoint);
        call.setPortTypeName(new QName(targetNameSpace,
portTypeName));
        call.setOperationName(new QName(targetNameSpace,
"analyze"));
        call.addParameter("Status", stringQName,
ParameterMode.IN);
        call.addParameter("hwsUDDI", stringQName,
ParameterMode.IN);
        call.addParameter("resUDDI", stringQName,
ParameterMode.IN);
        call.addParameter("KB", stringQName,
ParameterMode.IN);
        call.setReturnType(stringQName);
        //Service Invocation
        Object[] inParams = new Object[4];
        inParams [0] = MWS_Output;
        inParams [1] = UDDI1;
        inParams [2] = UDDI2;
        inParams [3] = knowledgeBase;
        awsResponse = (String) call.invoke(inParams);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
    }
}

```

```

        e.printStackTrace();
    }
    return awsResponse;
}
//Invoking the Entry Method of PWS
public String invokePWS(String AWS_Output, String
targetNamespace, String endpoint)
{
    String portTypeName = "PWS";
    String pwsResponse = null;
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        //Remote Invocation-----
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(targetNamespace, portTypeName));
        //Service Access
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
        "");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
        "wrapped");
        call.setTargetEndpointAddress(endpoint);
        call.setPortTypeName(new QName(targetNamespace,
portTypeName));
        call.setOperationName(new QName(targetNamespace,
"plan"));
        call.addParameter("Action", stringQName,
ParameterMode.IN);
        call.setReturnType(stringQName);
        //Service Invocation
        Object[] inParams = new Object[1];
        inParams [0] = AWS_Output;
        pwsResponse = (String) call.invoke(inParams);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    return pwsResponse;
}
//Invoking the Entry Method of PWS
public String invokeEWS(String PWS_Output, String
targetNamespace, String endpoint, String HWS, String Resource)
{
    String portTypeName = "EWS";
    String ewsResponse = null;
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {

```



```

//Remote Invocation-----
ServiceFactory serviceFactory =
ServiceFactory.newInstance();
Service service = serviceFactory.createService(new
QName(targetNameSpace, portTypeName));
//Service Access
Call call = (Call) service.createCall();
call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
call.setTargetEndpointAddress(endpoint);
call.setPortTypeName(new QName(targetNameSpace,
portTypeName));
call.setOperationName(new QName(targetNameSpace,
"execute"));
call.addParameter("finalAction", stringQName,
ParameterMode.IN);
call.addParameter("hws", stringQName,
ParameterMode.IN);
call.addParameter("resource", stringQName,
ParameterMode.IN);
call.setReturnType(stringQName);
//Service Invocation
Object[] inParams = new Object[3];
inParams [0] = PWS_Output;
inParams [1] = HWS;
inParams [2] = Resource;
ewsResponse = (String) call.invoke(inParams);
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
return ewsResponse ;
}
}
////////////////////////////////////
//Getting a List of Active Processes
public String getActiveProcesses(String resUDDI) throws
Exception
{
    String activeProcesses = null;
    String res_portTypeName = null;
    String res_targetNameSpace = null;
    String res_endpoint = null;
    String [] resourceUDDI = resUDDI.split(" ");
    res_portTypeName = resourceUDDI[0];
    res_targetNameSpace = resourceUDDI[1];
    res_endpoint = resourceUDDI[2];
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        //Remote Invocation-----
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(res_targetNameSpace, res_portTypeName));

```

```

//Service Access
Call call = (Call) service.createCall();
call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
""");
call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
call.setTargetEndpointAddress(res_endpoint);
call.setPortTypeName(new QName(res_targetNameSpace,
res_portTypeName));
call.setOperationName(new
QName(res_targetNameSpace, "getAllProcesses"));
call.removeAllParameters();
call.setReturnType(stringQName);
activeProcesses = (String) call.invoke(null);
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
return activeProcesses;
}
////////////////////////////////////
//Getting a List of Running Services
public String getRunningServices(String resUDDI)
{
    String runningServices = null;
    String res_portTypeName = null;
    String res_targetNameSpace = null;
    String res_endpoint = null;
    String [] resourceUDDI = resUDDI.split(" ");
    res_portTypeName = resourceUDDI[0];
    res_targetNameSpace = resourceUDDI[1];
    res_endpoint = resourceUDDI[2];
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        //Remote Invocation-----
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(res_targetNameSpace, res_portTypeName));
        //Service Access
        Call call = (Call) service.createCall();
call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY, "");
call.setProperty(call.OPERATION_STYLE_PROPERTY, "wrapped");
call.setTargetEndpointAddress(res_endpoint);
call.setPortTypeName(new QName(res_targetNameSpace,
res_portTypeName));
call.setOperationName(new
QName(res_targetNameSpace, "getAllServices"));
call.removeAllParameters();
call.setReturnType(stringQName);
        //Service Invocation
        runningServices = (String) call.invoke(null);
    }
    catch(Exception e)
    {

```

```

        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    return runningServices;
}
}
//Dispatching Commands on the OS
public String dispatch (String command, String resource, long
schedule)
{
    String dispatchingStatus = null;
    String effector = null;
    String res_portTypeName = null;
    String res_targetNameSpace = null;
    String res_endpoint = null;
    String effector_portTypeName = null;
    String effector_targetNameSpace = null;
    String effector_endpoint = null;
    String[] resourceUDDI = resource.split(" ");
    res_portTypeName = resourceUDDI[0] ;
    res_targetNameSpace = resourceUDDI[1] ;
    res_endpoint = resourceUDDI[2] ;
    effector = findEffector(res_portTypeName,
res_targetNameSpace, res_endpoint);
    String[] effectorUDDI = effector.split(" ");
    effector_portTypeName = effectorUDDI[0] ;
    effector_targetNameSpace = effectorUDDI[1] ;
    effector_endpoint = effectorUDDI[2] ;
    if
(command.startsWith("Start") || command.startsWith("Stop"))
    {
        String KnowledgeBaseRef = null;
        String[] toRun = command.split(" ");
        String serviceName = null;
        QName stringQName = new
QName ("http://www.w3.org/2001/XMLSchema", "string");
        try
        {
            ServiceFactory serviceFactory =
ServiceFactory.newInstance();
            Service service =
serviceFactory.createService(new QName(res_targetNameSpace,
res_portTypeName));
            Call call = (Call) service.createCall();
            call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
            call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
            call.setTargetEndpointAddress(res_endpoint);
            call.setPortTypeName(new QName(res_targetNameSpace,
res_portTypeName));
            call.setOperationName(new
QName(res_targetNameSpace, "getEffectorInfo"));
            call.removeAllParameters();
            call.setReturnType(stringQName);
            //Service Invocation
            KnowledgeBaseRef = (String) call.invoke(null);
            //Getting Information from Private UDDI-----
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

```

```

String dbURL = "jdbc:odbc:" + KnowledgeBaseRef;
Connection con1 = DriverManager.getConnection(
dbURL, "", "");

Statement st1 = con1.createStatement();
//Selection Statement
st1.executeQuery("SELECT ServiceToCheck FROM
Services Where ServiceName = '"+toRun[2]+"' "+toRun[3]+"' ");
ResultSet rs1 = st1.getResultSet();
if (rs1 != null)
    while( rs1.next())
        {serviceName = rs1.getString(1);}
    }
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
command = toRun[0] + toRun[1] + serviceName;
}
System.out.println("Actual Command Being Executed: " +
command);
dispatchingStatus = invokeEffector(command,
effector_portTypeName, effector_targetNameSpace, effector_endpoint,
schedule);
return dispatchingStatus;
}
////////////////////////////////////
//Invoking the Entry Method of the Effector Web Service
public String invokeEffector(String Command, String
portTypeName, String targetNameSpace, String endpoint, long schedule)
{
    String executeStatus = null;
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    QName longQName = new
QName("http://www.w3.org/2001/XMLSchema", "long");
    try
    {
        //Remote Invocation-----
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(targetNameSpace, portTypeName));
        //Service Access
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
        "");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
        "wrapped");
        call.setTargetEndpointAddress(endpoint);
        call.setPortTypeName(new QName(targetNameSpace,
portTypeName));
        call.setOperationName(new QName(targetNameSpace,
"runCommand"));
        call.addParameter("Command", stringQName,
ParameterMode.IN);
        call.addParameter("schedule", longQName,
ParameterMode.IN);
        call.setReturnType(stringQName);
    }
}

```

```

//Service Invocation
Object[] inParams = new Object[2];
inParams [0] = Command;
inParams [1] = new Long(schedule);
executeStatus = (String) call.invoke(inParams);
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
return executeStatus;
}
}
////////////////////////////////////
//Getting a Pointer to the OS Log File
public String getLogsFile(String UDDI) throws Exception
{
    String logFile = null;
    String portTypeName = null;
    String targetNameSpace = null;
    String endpoint = null;
    String[] hwsUDDI = UDDI.split(" ");
    portTypeName = hwsUDDI[0];
    targetNameSpace = hwsUDDI[1];
    endpoint = hwsUDDI[2];
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        //Remote Invocation-----
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(targetNameSpace, portTypeName));
        //Service Access
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
        call.setTargetEndpointAddress(endpoint);
        call.setPortTypeName(new QName(targetNameSpace,
portTypeName));
        call.setOperationName(new QName(targetNameSpace,
"getLogsFile"));
        call.removeAllParameters();
        call.setReturnType(stringQName);
        //Service Invocation
        logFile = (String) call.invoke(null);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    return logFile;
}
}
////////////////////////////////////
//Checking The Health of HWS

```

```

public String checkHealth()
{
    String myHealth = null;
    myHealth = "HWS Is Alive!";
    return myHealth;
}
}

```

Listing 8 HWS J2EE Class

3. MWS J2EE Class

```

package monitoring.service;

import java.lang.String;
import java.lang.Exception;
import java.sql.*;
import java.util.*;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

//-----MWS Class-----//
public class MWS
{
    //////////////////////////////////////////////////Variables Declaration//////////////////////////////////////
    public String mwsPortType = "MWS";
    public String mwsTargetNamespace = "http://service.monitoring";
    public String mwsEndpoint =
"http://localhost:8080/Monitor/services/MWS";
    public String resourceType = "Windows XP";
    //////////////////////////////////////////////////
    //Publishing MWS into Private UDDI
    public String publishMWS() throws Exception
    {
        String publishingResult = null;
        String dataSourceName = "PrivateUDDI_DNS";
        String dbURL = "jdbc:odbc:" + dataSourceName;
        try
        {
            String sql = null;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
dbURL, "", "");
            Statement s = con.createStatement();
            sql = "INSERT INTO UDDI_Table " + "VALUES
('"+mwsTargetNamespace+"',
 '"+mwsPortType+"', '"+mwsEndpoint+"', '"+resourceType+"')";
            s.executeUpdate(sql);
            s.close();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println("Error: " + e);
            e.printStackTrace();
        }
    }
}

```

```

        publishingResult = "MWS Has Been Published Successfully";
        return publishingResult;
    }
    ////////////////////////////////////////////////////
    //MWS heal Method
    public String monitor (String metricValue, String metricType,
String knowledgeBaseRefrence) throws Exception
    {
        System.out.println("Monitoring Has Been Launched...");
        String mwsResult = null;
        String Comment2 = null;
        if (metricType.equals("CPU"))
        {
            System.out.println("Measuring CPU Utilization...");
            Comment2 = "CPU Utilization: ";
        }
        if (metricType.equals("Memory"))
        {
            System.out.println("Measuring Memory
Consumption...");
            Comment2 = "Memory Consumption: ";
        }
        String dataSourceName = knowledgeBaseRefrence;
        mwsResult = checkMetrics(metricValue, metricType,
dataSourceName) + Comment2;
        if (checkMetrics(metricValue,
metricType, dataSourceName).equals("Normal "))
        {
            mwsResult = mwsResult + "No Need For Analysis";
        }
        if (checkMetrics(metricValue,
metricType, dataSourceName).equals("Abnormal "))
        {
            mwsResult = mwsResult + "Analyze";
        }
        return mwsResult;
    }
    ////////////////////////////////////////////////////
    //Evaluating the Health of the OS
    public String checkMetrics(String sMetric, String Metric, String
DB_DNS) throws Exception
    {
        String result = null;
        String dataSourceName = DB_DNS;
        String dbURL = "jdbc:odbc:" + dataSourceName;
        double thresholdValue = 0.0;
        try{
            //Getting Thresholds-----
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
dbURL, "", "");
            Statement st = con.createStatement();
            //Selection Statement
            st.executeQuery("SELECT MetricThreshold FROM
Metrics Where MetricID = '"+Metric+"'");
            ResultSet rs = st.getResultSet();
            if (rs != null)
                while( rs.next())
            {

```

```

        }
        st.close();
        con.close();
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    if (Metric.equals("CPU"))
    {
        if ( Double.valueOf(sMetric).doubleValue() >
thresholdValue)
            result = "Abnormal ";
        else result = "Normal ";
    }
    else if (Metric.equals("Memory"))
    {
        if
(Double.valueOf(sMetric).doubleValue() < thresholdValue)
            result = "Abnormal ";
        else result = "Normal ";
    }
    return result;
}
}

```

Listing 9 MWS J2EE Class

4. AWS J2EE Class

```

package analysis.service;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.lang.String;
import java.lang.Exception;
import java.sql.*;
import java.util.Calendar;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

//-----AWS Web Service Class-----//
public class AWS
{
    //////////////////////////////////////////////////Variables Declaration//////////////////////////////////////
    //AWS Web Service UDDI Information
    public String awsPortType = "AWS";
    public String awsTargetNamespace = "http://service.analysis";
    public String awsEndpoint =
"http://localhost:8080/Analyzer/services/AWS";
    public String resourceType = "Windows XP";
    ////////////////////////////////////////////////////
    //Publishing AWS into Private UDDI
    public String publishAWS() throws Exception
    {

```



```

String publishingResult = null;
String dataSourceName = "PrivateUDDI_DNS";
String dbURL = "jdbc:odbc:" + dataSourceName;
try
{
    String sql = null;
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection(
dbURL, "", "");
    Statement s = con.createStatement();
    sql = "INSERT INTO UDDI_Table " + "VALUES
('"+awsTargetNamespace+"',
 '"+awsPortType+"', '"+awsEndpoint+"', '"+resourceType+"')";
    s.executeUpdate(sql);
    s.close();
    con.close();
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
publishingResult = "AWS Has Been Published Successfully";
return publishingResult;
}
////////////////////////////////////
//AWS Entry Method
public String analyze ( String reportedStatus, String hwsUDDI
,String resUDDI, String KB) throws Exception
{
    String analyzer1Result = null;
    String analyzer2Result = null;
    String awsResponse = null;
    String Capability = null;
    String dataSourceName = "SymptomsDB_DNS";
    System.out.println("Analysis Has Been Started...");
    Capability = checkCapability(reportedStatus);
    if (Capability.equals("ok"))
    {
        System.out.println("Analyzing A known
Situation...");
        if (reportedStatus.equals("Abnormal CPU
Utilization"))
        {
            System.out.println("Analyzing CPU
Utilization...");
            String processesFile = null;
            System.out.println("Getting Active
Processes...");
            processesFile = getProcesses(hwsUDDI,
resUDDI);
            analyzer1Result =
analyzeProcesses(processesFile);
            if (analyzer1Result.equals("Normal Status"))
            {
                awsResponse = "No Problem Was Found";
            }
            else
            {

```

```

        findSolution(reportedStatus) + " " + analyzer1Result;
    }
    else if (reportedStatus.equals("Metric Could Not Be
Collected!"))
    {
        System.out.println("Analyzing NSClient
Service...");
        analyzer2Result = solve(reportedStatus,
"NSClient", KB, hwsUDDI, resUDDI);
        awsResponse = analyzer2Result;
    }
    else if (Capability.equals("Perfroming Analysis From
Scratch!"))
    {
        System.out.println("Analyzing Unknown
Situation...");
        System.out.println("Analyzing Logs To Determine The
Cause of the Reported Situation...");
        String logsFile = null;
        String logs = null;
        logsFile = getLogs(hwsUDDI, resUDDI);
        logs = analyzeLogs(logsFile);
        System.out.println("Log Analysis Result: " + logs);
        int k = 0;
        k = logs.lastIndexOf(' ');
        String logMessage = logs.substring(0, k);
        String logReporter = logs.substring(k+1);
        if (logMessage.equals("Stop situation data"))
        {
            System.out.println("Retreiving Running
Services....");
            System.out.println("Checking " + logReporter
+ " Service....");
            analyzer2Result = solve(reportedStatus,
logReporter, KB, hwsUDDI, resUDDI);
            String dbURL = "jdbc:odbc:" + dataSourceName;
            try
            {
                String sql = null;
                String action = "Start NSClient
Service";
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                Connection con =
DriverManager.getConnection( dbURL, "", "");
                Statement s = con.createStatement();
                sql = "INSERT INTO Table1 " + "VALUES
('"+reportedStatus+"', '"+action+"')";
                s.executeUpdate(sql);
                s.close();
                con.close();
            }
            catch(Exception e)
            {
                System.out.println("Error: " + e);
                e.printStackTrace();
            }
        }
    }
}

```

```

    }
    awsResponse = analyzer2Result;
}
System.out.println(awsResponse);
return awsResponse;
}
////////////////////////////////////
//Retrieving the OS Log File
public String getLogs(String UDDI1, String UDDI2) throws
Exception
{
    String logsFile = null;
    String hws_portTypeName = null;
    String hws_targetNameSpace = null;
    String hws_endpoint = null;
    String[] hwsUDDI = UDDI1.split(" ");
    hws_portTypeName = hwsUDDI[0];
    hws_targetNameSpace = hwsUDDI[1];
    hws_endpoint = hwsUDDI[2];
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    try
    {
        //Remote Invocation-----
        ServiceFactory serviceFactory =
ServiceFactory.newInstance();
        Service service = serviceFactory.createService(new
QName(hws_targetNameSpace, hws_portTypeName));
        //Service Access
        Call call = (Call) service.createCall();
        call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
        call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
        call.setTargetEndpointAddress(hws_endpoint);
        call.setPortTypeName(new QName(hws_targetNameSpace,
hws_portTypeName));
        call.setOperationName(new
QName(hws_targetNameSpace, "getLogsFile"));
        call.addParameter("resource", stringQName,
ParameterMode.IN);
        call.setReturnType(stringQName);
        Object[] inParams = new Object[1];
        inParams [0] = UDDI2;
        logsFile = (String) call.invoke(inParams);
    }
    catch(Exception e)
    {
        System.out.println("Error: " + e);
        e.printStackTrace();
    }
    return logsFile;
}
////////////////////////////////////
//Checking the Familiarity of AWS with the Reported Situation
public String checkCapability(String statusToAnalyze)
{
    String capability = null;
    String currentCapabilitiesTemp = null;

```

```

String currentCapabilities = null;
String dataSourceName = "SymptomsDB_DNS";
int index = 0 ;
String dbURL = "jdbc:odbc:" + dataSourceName;
try
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection(
dbURL, "", "");
    Statement s = con.createStatement();
    s.executeQuery("SELECT OperationalStatus FROM
Table1");
    ResultSet rs = s.getResultSet();
    if (rs != null)
        while( rs.next())
        {
            currentCapabilitiesTemp =
currentCapabilitiesTemp + "-" + rs.getString(1);
        }
    s.close();
    con.close();
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
index = currentCapabilitiesTemp.indexOf('-');
currentCapabilities =
currentCapabilitiesTemp.substring(index+1);
String[] Capabilities = currentCapabilities.split("-");
int u = 0;
boolean flag = false;
while (u<Capabilities.length&&flag == false)
{
    if (Capabilities[u].equals(statusToAnalyze))
    {
        System.out.println(Capabilities[u]);
        flag =true;
    }
    u++;
}
if (flag==true)
    capability = "ok";
else
    capability = "Perfroming Analysis From Scratch!";
return capability;
}
////////////////////////////////////
//Retrieving Active Processes
public String getProcesses(String UDDI1, String UDDI2)
{
    String activeProcesses = null;
    String hws_portTypeName = null;
    String hws_targetNameSpace = null;
    String hws_endpoint = null;
    String [] hwsUDDI = UDDI1.split(" ");
    hws_portTypeName = hwsUDDI[0];
    hws_targetNameSpace = hwsUDDI[1];
}

```

```

        hws_endpoint = hwsUDDI[2];
        QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
        try
        {
            //Remote Invocation-----
            ServiceFactory serviceFactory =
ServiceFactory.newInstance();
            Service service = serviceFactory.createService(new
QName(hws_targetNameSpace, hws_portTypeName));
            //Service Access
            Call call = (Call) service.createCall();
            call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
            call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
            call.setTargetEndpointAddress(hws_endpoint);
            call.setPortTypeName(new QName(hws_targetNameSpace,
hws_portTypeName));
            call.setOperationName(new
QName(hws_targetNameSpace, "getActiveProcesses"));
            call.addParameter("resUDDI", stringQName,
ParameterMode.IN);
            call.setReturnType(stringQName);
            //Service Invocation
            Object[] inParams = new Object[1];
            inParams [0] = UDDI2;
            activeProcesses = (String) call.invoke(inParams);
        }
        catch(Exception e)
        {
            System.out.println("Error: " + e);
            e.printStackTrace();
        }
        return activeProcesses;
    }
    //Retrieving Running Services
    public String getServices(String hws, String res)
    {
        String runningServices = null;
        String hws_portTypeName = null;
        String hws_targetNameSpace = null;
        String hws_endpoint = null;
        String [] hwsUDDI = hws.split(" ");
        hws_portTypeName = hwsUDDI[0];
        hws_targetNameSpace = hwsUDDI[1];
        hws_endpoint = hwsUDDI[2];
        QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
        try
        {
            //Remote Invocation-----
            ServiceFactory serviceFactory =
ServiceFactory.newInstance();
            Service service = serviceFactory.createService(new
QName(hws_targetNameSpace, hws_portTypeName));
            //Service Access
            Call call = (Call) service.createCall();

```

```

");
call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"wrapped");
call.setProperty(call.OPERATION_STYLE_PROPERTY,
call.setTargetEndpointAddress(hws_endpoint);
call.setPortTypeName(new QName(hws_targetNameSpace,
hws_portTypeName));
call.setOperationName(new
QName(hws_targetNameSpace, "getRunningServices"));
call.addParameter("resUDDI", stringQName,
ParameterMode.IN);
call.setReturnType(stringQName);
//Service Invocation
Object[] inParams = new Object[1];
inParams [0] = res;
runningServices = (String) call.invoke(inParams);
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
return runningServices;
}
////////////////////////////////////
//Analyzing Active Processes
public String analyzeProcesses (String processesFile) throws
Exception
{
    System.out.println("Analyzing Active Processes...");
    int length = 0;
    String analysisResult= null;
    String processes = null;
    String arrangedProcesses = null;
    String s = null;
    int i = 0;
    String newline = System.getProperty("line.separator");
    try
    {
        File FileName = new File(processesFile);
        FileReader fr = new FileReader(FileName);
        BufferedReader br = new BufferedReader (fr);
        while ((s = br.readLine()) !=null)
        {
            if
(s.endsWith("invalid.") || s.endsWith("denied."))
                processes = processes + s;
            else
                processes = processes + s + newline;
        }
        i = processes.indexOf("Idle");
        arrangedProcesses = "
processes.substring(i);
        fr.close();
    }
    catch(Exception e)
    {
        System.out.print("Active Processes Cannot Be
Retrieved!");
    }
}

```

```

    }
    String[] tempProcesses =
arrangedProcesses.split(newline);
    length = tempProcesses.length - 2;
    String[] Processes = new String[length];
    String[] processName = new String[length];
    String[] processNameTemp = new String[length];
    String[] processID = new String[length];
    String[] processIDTemp = new String[length];
    String[] processThreads = new String[length];
    String[] processThreadsTemp = new String[length];
    String[] processPriority = new String[length];
    String[] processPriorityTemp = new String[length];
    String[] processUtilization = new String[length];
    String[] processUtilizationTemp = new String[length];
    String[] processOwner = new String[length];
    String[] processOwnerTemp = new String[length];
    int[] procID = new int[length];
    int[] procPriority = new int [length];
    int[] procUtilization = new int[length];
    for (int x=0; x<length; x++)
    {
        Processes[x] = new String(tempProcesses[x+2]);
        processNameTemp[x] = new String
(Processes[x].substring(0,16));
        processIDTemp[x] = new
String(Processes[x].substring(17,22));
        processThreadsTemp[x] = new
String(Processes[x].substring(23,30));
        processPriorityTemp[x] = new
String(Processes[x].substring(31,39));
        processUtilizationTemp [x] = new
String(Processes[x].substring(40,43));
        processOwnerTemp[x] = new String
(Processes[x].substring(44));
    }
    for (int x=0; x<length; x++)
    {
        int index1 = 0;
        index1 = processNameTemp[x].lastIndexOf(" ");
        processName[x] = new String
(processNameTemp[x].substring(index1+1));
        int index2 = 0;
        index2 = processIDTemp[x].lastIndexOf(" ");
        processID[x] = new String
(processIDTemp[x].substring(index2+2));
        procID[x] =
Integer.valueOf(processID[x]).intValue();
        int index3 = 0;
        index3 = processThreadsTemp[x].lastIndexOf(" ");
        processThreads[x] = new String
(processThreadsTemp[x].substring(index3+1));
        int index4 = 0;
        index4 = processPriorityTemp[x].lastIndexOf(" ");
        processPriority[x] = new String
(processPriorityTemp[x].substring(index4+1));
        procPriority[x] =
Integer.valueOf(processPriority[x]).intValue();
        int index5 = 0;

```

```

");
        index5 = processUtilizationTemp[x].lastIndexOf("
        processUtilization[x] = new String
(processUtilizationTemp[x].substring(index5+1));
        procUtilization[x] =
Integer.valueOf(processUtilization[x]).intValue();
        int index6 = 0;
        index6 = processOwnerTemp[x].lastIndexOf(" ");
        processOwner[x] = new String
(processOwnerTemp[x].substring(index6+1));
    }
    String maxUtilizationProcName = null;
    int maxUtilizationProcID = 0;
    String maxUtilizationProcOwner = null;
    int maxUtilizationProcPriority = 0;
    int maxUtilization = 0;
    for (int x=0; x<length; x++)
    {
        if (procUtilization[x]>maxUtilization)
        {
            int index = 0;
            maxUtilizationProcName = processName[x];
            maxUtilizationProcID = procID[x];
            index=processOwner[x].indexOf('\\');
            maxUtilizationProcOwner =
processOwner[x].substring(index+1);
            maxUtilizationProcPriority = procPriority[x];
            maxUtilization = procUtilization[x];
        }
    }
    System.out.println("Process of Maximum CPU Utilization: "
+ maxUtilizationProcName);
    System.out.println("ID of the Process of Maximum CPU
Utilization: " + String.valueOf(maxUtilizationProcID));
    System.out.println("Owner of the Process of Maximum CPU
Utilization: " + maxUtilizationProcOwner);
    System.out.println("Priority of the Process of Maximum
CPU Utilization: " + String.valueOf(maxUtilizationProcPriority));
    System.out.println("Maximum CPU Utilization: " +
String.valueOf(maxUtilization));
    if (maxUtilization == 0 ||
maxUtilizationProcName.equals("null"))
    {
        analysisResult = "Normal Status";
    }
    else
    {
        analysisResult = maxUtilizationProcName + " " +
String.valueOf(maxUtilizationProcID) + " " + maxUtilizationProcOwner
+ " " + String.valueOf(maxUtilizationProcPriority) + " " +
String.valueOf(maxUtilization);
    }
    return analysisResult;
}
//////////////////////////////////////
//Root-Cause Analysis by Analyzing Logs
public String analyzeLogs(String logsFile)throws Exception
{
    System.out.println("Analyzing Available Logs...");
}

```



```

String s = null;
String logsAnalysisResult= null;
String logsTemp = null;
String logs = null;
String CurrentTime = null;
String dayName = null;
String Day = null;
String Month = null;
String timeZone = null;
String Zone = null;
String Year = null;
String clock = null;
String Hours = null;
String Minutes = null;
String Seconds = null;
String timeToConvert = null;
String convertedTime = null;
boolean flag =false;
int index = 0;
int index2 = 0;
int length = 0;
String patternStr1 = " ";
String patternStr2 = ":";
String newline = System.getProperty("line.separator");
try
{
    File FileName = new File(logsFile);
    FileReader fr = new FileReader(FileName);
    BufferedReader br = new BufferedReader (fr);
    while ((s = br.readLine()) !=null)
    {
        logsTemp = logsTemp + s + newline;
    }
    fr.close();
}
catch(Exception e)
{
    System.out.print("Active Processes Cannot Be
Retreived!");
}
index = logsTemp.indexOf('[');
logs = logsTemp.substring(index);
String[] allLogs = logs.split(newline);
length = allLogs.length;
String logDate = null;
String logClock = null;
String[] logTimeTemp = new String[length];
String[] logD = new String[length];
String[] logMo = new String[length];
String[] logY = new String[length];
String[] logH = new String[length];
String[] logM = new String[length];
String[] logS = new String[length];
String[] logReporter = new String[length];
String[] logID = new String[length];
String[] logMessage = new String[length];
for (int x=0; x<length; x++)
{
    String[] Logs = allLogs[x].split(",");

```

```

logTimeTemp[x] = new String (Logs[0]);
String[] logTime = logTimeTemp[x].split(" ");
logDate = logTime[0].substring(1);
logClock = logTime[1];
timeZone = logTime[2].substring(0,3);
String[] dateRecord = logDate.split("/");
logMo[x] = dateRecord[0];
logD[x] = dateRecord[1];
logY[x] = dateRecord[2];
String[] timeRecord = logClock.split(":");
logH[x] = timeRecord [0];
logM[x] = timeRecord [1];
logS[x] = timeRecord [2];
logReporter[x] = new String (Logs[2]);
logID[x] = new String (Logs[4]);
logMessage[x] = new String (Logs[5]);
}
CurrentTime = getTime();
String[] TimeFields = CurrentTime.split(patternStr1);
dayName = TimeFields[0];
Month = TimeFields[1];
Day = TimeFields[2];
clock = TimeFields[3];
Zone = TimeFields[4];
Year = TimeFields[5];
String[] hms = clock.split(patternStr2);
Hours = hms[0];
Minutes = hms[1];
Seconds = hms[2];
if (timeZone.equals(Zone))
{
    System.out.println("The Same Time Zone");
}
else
{
    System.out.println("Different Time Zones...Time
Conversion Is Being Performed...");
}
timeToConvert = Month + " " +Year;
convertedTime = convertTime(timeToConvert);
String[] convertedTimeRecord = convertedTime.split(" ");
int convertedMonth =
Integer.valueOf(convertedTimeRecord[0]).intValue();
int convertedYear = Integer.valueOf(convertedTimeRecord
[1]).intValue();
int day = Integer.valueOf(Day).intValue();
int H = Integer.valueOf(Hours).intValue();
int M = Integer.valueOf(Minutes).intValue();
int S = Integer.valueOf(Seconds).intValue();
int u = length-1;
int m = 0;
int[] years = new int[length];
int[] days = new int[length];
int[] months = new int[length];
int[] hours = new int[length];
int[] minutes = new int[length];
int[] seconds = new int[length];
for (int h=0; h<length; h++)
{

```

```

years[h] = Integer.valueOf(logY[h]).intValue();
days[h] = Integer.valueOf(logD[h]).intValue();
months[h] = Integer.valueOf(logMo[h]).intValue();
hours[h] = Integer.valueOf(logH[h]).intValue();
minutes[h] = Integer.valueOf(logM[h]).intValue();
seconds[h] = Integer.valueOf(logS[h]).intValue();
}
while (u >= 0 && flag == false)
{
    if (convertedYear - years[u] < convertedYear - years[u-1])
    {
        logsAnalysisResult = logMessage[u] + " " +
logReporter[u];
        m = u;
        flag = true;
    }
    else
    {
        if (convertedYear - years[u] == convertedYear -
years[u-1])
        {
            if (convertedMonth - months[u] < convertedMonth -
months[u-1])
            {
                logsAnalysisResult = logMessage[u] + "
" + logReporter[u];
                m = u;
                flag = true;
            }
        }
        if (convertedMonth - months[u] == convertedMonth -
months[u-1])
        {
            if (day - days[u] < day - days[u-1])
            {
                logsAnalysisResult = logMessage[u] +
" " + logReporter[u];
                m = u;
                flag = true;
            }
        }
        if (day - days[u] == day - days[u-1])
        {
            if (H - hours[u] < H - hours[u-1])
            {
                logsAnalysisResult = logMessage[u] +
" " + logReporter[u];
                m = u;
                flag = true;
            }
        }
        if (H - hours[u] == H - hours[u-1])
        {
            if (M - minutes[u] < M - minutes[u-1])
            {
                logsAnalysisResult = logMessage[u] +
" " + logReporter[u];
                m = u;
                flag = true;
            }
        }
    }
}

```

```

    }
    }
    else if (M-minutes[u]==M-minutes[u-1])
    {
        if (S-seconds[u]<S-seconds[u-1])
        {
            logsAnalysisResult = logMessage[u] +
            m = u;
            flag = true;
        }
    }
    }
    u--;
}
if (u==0)
    System.out.println("No Match!");
return logsAnalysisResult;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Analyzing Running Services
public String analyzeServices (String servicesFile, String
serviceToCheck) throws Exception
{
    int length = 0;
    String analysisResult= null;
    String services = null;
    String arrangedServices = null;
    String s = null;
    int i = 0;
    String newline = System.getProperty("line.separator");
    try
    {
        File FileName = new File(servicesFile);
        FileReader fr = new FileReader(FileName);
        BufferedReader br = new BufferedReader (fr);
        while ((s = br.readLine()) !=null)
        {
            services = services + s + newline;
        }
        fr.close();
    }
    catch(Exception e)
    {
        System.out.print("Running Services Cannot Be
Retreived!");
    }
    i = services.indexOf('A');
    arrangedServices = " " + services.substring(i);
    String[] ServicesTemp = arrangedServices.split(newline);
    length = ServicesTemp.length;
    String[] Services= new String[length];
    for (int t=0; t<length; t++)
    {
        Services[t] = new String
(ServicesTemp[t].substring(3));
    }
    int u = 0;
    boolean flag = false;
}

```

```

while (u<Services.length&&flag == false)
{
    if (Services[u].equals(serviceToCheck))
    {
        flag =true;
    }
    u++;
}
if (flag==true)
    analysisResult = "Service "+ serviceToCheck + " Is
Running";
else
    analysisResult = "Service "+ serviceToCheck + "
Isn't Running";
return analysisResult;
}
////////////////////////////////////
//Checking Symptoms Database
public String findSolution(String rMetric)
{
    return checkSymptomsDB("SymptomsDB_DNS",rMetric);
}
public String checkSymptomsDB(String dataSourceName, String
Metric)
{
    String opStatusTemp = null;
    String opStatus = null;
    String action = null;
    String dbURL = "jdbc:odbc:" + dataSourceName;
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection(
dbURL, "", "");
        Statement s = con.createStatement();
        s.executeQuery("SELECT OperationalStatus FROM
Table1");
        ResultSet rs = s.getResultSet();
        if (rs != null)
            while( rs.next())
            {
                opStatusTemp = opStatusTemp+ "-" +
rs.getString(1);
            }
        s.close();
        con.close();
        int index = 0;
        opStatus = opStatusTemp.substring(index+1);
        String[] status = opStatus.split("-");
        boolean flag = false;
        int u = 0;
        while (u<status.length&&flag == false)
        {
            if (Metric.equals(status[u]))
            {
                System.out.println("Status Can Be
Analyzed");
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

```

```

DriverManager.getConnection( dbURL, "", "");
Connection con1 =
Statement s1 = con1.createStatement();
s1.executeQuery("SELECT Action FROM
Table1 WHERE OperationalStatus = '"+Metric+"' ");
ResultSet rsl = s1.getResultSet();
if (rsl != null)
    while( rsl.next())
    {
        action = rsl.getString(1);
    }
s.close();
con.close();
flag =true;
    }
    u++;
}
if (flag==false)
{
    System.out.println("No Healing Action Can Be
Suggested");
    System.out.println("More Information Is
needed for Analysis");
    action = "More Information";
}
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
return action;
}
}
////////////////////////////////////
//Checking Services
public String solve(String rMetric, String
failedComponent,String KB, String hwsUDDI,String resUDDI) throws
Exception
{
    String serviceToCheck = null;
    String solution = null;
    String servicesFile = null;
    String result = null;
    servicesFile = getServices(hwsUDDI, resUDDI);
    serviceToCheck = checkKnolwedgeBase(KB, rMetric);
    if (serviceToCheck.equals(failedComponent))
        solution = analyzeServices(servicesFile,
serviceToCheck);
    else
    {
        solution = analyzeServices(servicesFile,
failedComponent);
    }
    System.out.println("solution: "+solution);
    if (solution.equals("Service "+ serviceToCheck + " Is
Running"))
        result = "No Solution Is Suggested";
    else
        result = "Start Service " + serviceToCheck;
}
}

```

```

        return result;
    }
    ///////////////////////////////////////////////////////////////////
    //Checking Services That Must Be Running in OS
    public String checkKnolwedgeBase(String dataSourceName, String
Metric)
    {
        String serviceToCheck = null;
        String dbURL = "jdbc:odbc:" + dataSourceName;
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
dbURL, "", "");
            Statement s = con.createStatement();
            s.executeQuery("SELECT ServiceToCheck FROM Services
WHERE Symptom = '"+Metric+"' ");
            ResultSet rs = s.getResultSet();
            if (rs != null)
                while( rs.next())
                {
                    serviceToCheck = rs.getString(1);
                }
            s.close();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println("Error: " + e);
            e.printStackTrace();
        }
        return serviceToCheck;
    }
    ///////////////////////////////////////////////////////////////////
    //Getting Current Time
    public String getTime()
    {
        String time = null;
        Calendar cal = Calendar.getInstance();
        time = String.valueOf(cal.getTime());
        return time;
    }
    ///////////////////////////////////////////////////////////////////
    //Converting Time into CBE Format
    public String convertTime(String Time)
    {
        String convertedTime = null;
        int tempTime = 0;
        String[] timeToConvert = Time.split(" ");
        if (timeToConvert[0].equals("Jan"))
            convertedTime = "01";
        else if (timeToConvert[0].equals("Feb"))
            convertedTime = "02";
        else if (timeToConvert[0].equals("Mar"))
            convertedTime = "03";
        else if (timeToConvert[0].equals("Apr"))
            convertedTime = "04";
        else if (timeToConvert[0].equals("May"))
            convertedTime = "05";
    }

```

```

else if (timeToConvert[0].equals("Jun"))
    convertedTime = "06";
else if (timeToConvert[0].equals("Jul"))
    convertedTime = "07";
else if (timeToConvert[0].equals("Aug"))
    convertedTime = "08";
else if (timeToConvert[0].equals("Sep"))
    convertedTime = "09";
else if (timeToConvert[0].equals("Oct"))
    convertedTime = "10";
else if (timeToConvert[0].equals("Nov"))
    convertedTime = "11";
else if (timeToConvert[0].equals("Dec"))
    convertedTime = "12";
tempTime = Integer.valueOf(timeToConvert[1]).intValue();
tempTime = tempTime-2000;
if (tempTime<10)
{
    convertedTime = convertedTime + " " +
"0"+String.valueOf(tempTime);
}
else
{
    convertedTime = convertedTime + " "
+String.valueOf(tempTime);
}
return convertedTime;
}
}

```

Listing 10 AWS J2EE Class

5. PWS J2EE Class

```

package planning.service;

import java.util.*;
import java.util.Date;
import java.lang.String;
import java.sql.*;
import java.lang.Exception;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

//-----PWS Class-----//
public class PWS
{
    //Variables Declaration//
    public String pwsPortType = "PWS";
    public String pwsTargetNamespace = "http://service.planning";
    public String pwsEndpoint =
"http://localhost:8080/Planner/services/PWS";
    public String resourceType = "Windows XP";
    //Publishing PWS into Private UDDI
    public String publishPWS() throws Exception
{

```



```

String publishingResult = null;
String dataSourceName = "PrivateUDDI_DNS";
String dbURL = "jdbc:odbc:" + dataSourceName;
try
{
    String sql = null;
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection(
dbURL, "", "");
    Statement s = con.createStatement();
    sql = "INSERT INTO UDDI_Table " + "VALUES
('"+pwsTargetNamespace+"',
 '"+pwsPortType+"', '"+pwsEndpoint+"', '"+resourceType+"')";
    s.executeUpdate(sql);
    s.close();
    con.close();
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
publishingResult = "PWS Has Been Published Successfully";
return publishingResult;
}
////////////////////////////////////
//PWS Entry Method
public String plan(String suggestedAction)throws Exception
{
    System.out.println("suggestedAction: "+ suggestedAction);
    String planningResult = null;
    String plannerResult = null;
    String[] actionToExecute = suggestedAction.split(" ");
    String actionToEWS = actionToExecute[0] + " " +
actionToExecute[1] + " " + actionToExecute[2];
    System.out.println("Planning Has Been Started...");
    planningResult = checkSuggestedAction(suggestedAction);
    if (planningResult.equals("ok"))
        plannerResult = actionToEWS;
    if (planningResult.equals("Find Another Process"))
        plannerResult = planningResult;
    return plannerResult;
}
////////////////////////////////////
//Validating Action Suggested by AWS
public String checkSuggestedAction(String action)
{
    String actionToExceute = null;
    String systemProcessesTemp = null;
    String systemProcesses = null;
    String[] command = action.split(" ");
    String dataSourceName = "SharedKnowledgeBase_DNS";
    String dbURL = "jdbc:odbc:" + dataSourceName;
    if
(command[0].equals("Start") || command[0].equals("Stop"))
    {
        actionToExceute = "ok";
    }
    else

```

```

    {
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
dbURL, "", "");
            Statement s = con.createStatement();
            //Selection Statement
            s.executeQuery("SELECT PID FROM Policies");
            ResultSet rs = s.getResultSet();
            if (rs != null)
                while( rs.next())
                {
                    systemProcessesTemp =
systemProcessesTemp + " " + rs.getString(1);
                }
            s.close();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println("Error: " + e);
            e.printStackTrace();
        }
        int index = 0;
        index = systemProcessesTemp.indexOf(" ");
        systemProcesses =
systemProcessesTemp.substring(index+1);
        String[] sysProcesses = systemProcesses.split(" ");
        String[] actionItems = action.split(" ");
        boolean flag = false;
        int u = 0;
        while (u<sysProcesses.length&&flag == false)
        {
            if
(actionItems[2].equals(sysProcesses[u]) || actionItems[4].equals("SYSTE
M"))
            {
                flag =true;
            }
            u++;
        }
        if (flag==true)
            actionToExceute = "Find Another Process";
        else
            actionToExceute = "ok";
        }
        return actionToExceute;
    }
}

```

Listing 11PWS J2EE Class

6. EWS J2EE Class

```

package execution.service;

import java.sql.*;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;

```

```

import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;

//-----EWS Class-----//
public class EWS
{
    ///////////////////////////////////////////////////Variables Declaration//////////////////////////////////////
    public String ewsPortType = "EWS";
    public String ewsTargetNamespace = "http://service.execution";
    public String ewsEndpoint =
"http://localhost:8080/Executor/services/EWS";
    public String resourceType = "Windows XP";
    ///////////////////////////////////////////////////
    //Publishing EWS into Private UDDI
    public String publishEWS() throws Exception
    {
        String publishingResult = null;
        String dataSourceName = "PrivateUDDI_DNS";
        String dbURL = "jdbc:odbc:" + dataSourceName;
        try
        {
            String sql = null;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
dbURL, "", "");

            Statement s = con.createStatement();
            sql = "INSERT INTO UDDI_Table " + "VALUES
('"+ewsTargetNamespace+"',
 '"+ewsPortType+"', '"+ewsEndpoint+"', '"+resourceType+"')";
            s.executeUpdate(sql);
            s.close();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println("Error: " + e);
            e.printStackTrace();
        }
        publishingResult = "EWS Has Been Published Successfully";
        return publishingResult;
    }
    ///////////////////////////////////////////////////
    //Translating Actions into Commands
    public String execute(String plan, String HWS, String resUDDI)
    {
        System.out.println("Execution Has Been Started...");
        String command = null;
        long schedule = 0;
        String executionStatus = null;
        String[] run = plan.split(" ");
        String key = run[0] + " " + run [1];
        String dataSourceName1 = "ExecutionDB_DNS";
        String dataSourceName2 = "SharedKnowledgeBase_DNS";
        String dbURL1 = "jdbc:odbc:" + dataSourceName1;
        String dbURL2 = "jdbc:odbc:" + dataSourceName2;
        try
        {
            //Getting Information-----//

```

```

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection(
dbURL1, "", "");
Statement st = con.createStatement();
//Selection Statement
st.executeQuery("SELECT Action FROM Dispatcher
Where Plan = '"+key+"' ");
ResultSet rs = st.getResultSet();
if (rs != null)
    while( rs.next())
        {command = rs.getString(1);}
st.close();
con.close();
Connection con2 = DriverManager.getConnection(
dbURL2, "", "");
Statement st2 = con2.createStatement();
//Selection Statement
st2.executeQuery("SELECT Delay FROM Delays Where
Action = '"+key+"' ");
ResultSet rs2 = st2.getResultSet();
if (rs2 != null)
    while( rs2.next())
        {schedule = rs2.getLong(1);}
st2.close();
con2.close();
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
System.out.println("Command: " + command);
System.out.println("Schedule: " +
String.valueOf(schedule));
executionStatus = dispatch (command + " " + run[2], HWS,
resUDDI, schedule);
return executionStatus;
}
////////////////////////////////////
//Reporting Commands and Execution Schedules
public String dispatch(String command, String hws, String
resource, long schedule)
{
    System.out.println(command + " Is Being Dispatched...");
    String hws_portTypeName = null;
    String hws_targetNameSpace = null;
    String hws_endpoint = null;
    String dispatchingStatus = null;
    String healingStatus = null;
    String [] hwsUDDI = hws.split(" ");
    hws_portTypeName = hwsUDDI[0];
    hws_targetNameSpace = hwsUDDI[1];
    hws_endpoint = hwsUDDI[2];
    QName stringQName = new
QName("http://www.w3.org/2001/XMLSchema", "string");
    QName longQName = new
QName("http://www.w3.org/2001/XMLSchema", "long");
    try
    {

```

```

//Remote Invocation-----
ServiceFactory serviceFactory =
ServiceFactory.newInstance();
Service service = serviceFactory.createService(new
QName(hws_targetNamespace, hws_portTypeName));
//Service Access
Call call = (Call) service.createCall();
call.setProperty(call.ENCODINGSTYLE_URI_PROPERTY,
"");
call.setProperty(call.OPERATION_STYLE_PROPERTY,
"wrapped");
call.setTargetEndpointAddress(hws_endpoint);
call.setPortTypeName(new QName(hws_targetNamespace,
hws_portTypeName));
call.setOperationName(new
QName(hws_targetNamespace, "dispatch"));
call.addParameter("command", stringQName,
ParameterMode.IN);
call.addParameter("resource", stringQName,
ParameterMode.IN);
call.addParameter("schedule", longQName,
ParameterMode.IN);
call.setReturnType(stringQName);
//Service Invocation
Object[] inParams = new Object[3];
inParams [0] = command;
inParams [1] = resource;
inParams [2] = new Long(schedule);
dispatchingStatus = (String) call.invoke(inParams);
if (dispatchingStatus.equals("ok"))
    healingStatus = (command + " Has Been
Executed Successfully..." + "Resource Status Has Been Healed...");
else
    healingStatus = (command + " Has Failed..." +
"Resource Status Has not Been Healed!");
}
catch(Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
}
return healingStatus;
}
}

```

Listing 12 EWS J2EE Class

7. Effector Web Service J2EE Class

```

package autonomic.Interface;

import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

//-----Effector Web Service Class-----//
public class effector extends Thread

```

```

{
    ////////////////////////////////////////////////////////////////////Variables Declaration//////////////////////////////////////////////////////////////////
    public String effectorPortType = "effector";
    public String effectorTargetNamespace =
"http://Interface.autonomic";
    public String effectorEndpoint =
"http://localhost:8080/Touchpoint/services/effector";
    public String resourceType = "Windows XP";
    ////////////////////////////////////////////////////////////////////
    //Publishing Effector into Private UDDI
    public String publishEffector() throws Exception
    {
        String publishingResult = null;
        String dataSourceName = "PrivateUDDI_DNS";
        String dbURL = "jdbc:odbc:" + dataSourceName;
        try
        {
            String sql = null;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
dbURL, "", "");
            Statement s = con.createStatement();
            sql = "INSERT INTO UDDI_Table " + "VALUES
('"+effectorTargetNamespace+"',
 '"+effectorPortType+"', '"+effectorEndpoint+"', '"+resourceType+"')";
            s.executeUpdate(sql);
            s.close();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println("Error: " + e);
            e.printStackTrace();
        }
        publishingResult = "Effector Has Been Published
Successfully";
        return publishingResult;
    }
    ////////////////////////////////////////////////////////////////////
    //Dispatching Commands
    public String runCommand(String command, long schedule)
    {
        String result = null;
        long seconds = 0;
        try
        {
            seconds = schedule * 1000;
            System.out.println("Command: " + command + " Will
Be Executed After: " + String.valueOf(seconds/1000) + " Seconds");
            synchronized(this)
            {
                wait(seconds);
            }
            Runtime.getRuntime().exec(command);
            /*BufferedReader stdInput = new BufferedReader(new
InputStreamReader(p.getInputStream()));
            BufferedReader stdError = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
            // read the output from the command

```

```
        while ((s = stdInput.readLine()) != null) {
            System.out.println(s);
        }
        // read any errors from the attempted command
        while ((s = stdError.readLine()) != null) {
            System.out.println(s);
        }*/
        result = "ok";
    }
    catch (IOException e)
    {
        System.out.println("exception happened!");
        e.printStackTrace();
        System.exit(-1);
    }
    catch (InterruptedException e)
    {
        System.out.println("Error");
    }
    return result;
}
}
```

Listing 13 Effector WS J2EE Class

AMERICAN UNIV. IN CAIRO LIBRARY
3 8534 01319 2632