IA-BASED FUALT DETECTION SYSTEM

"Mohammad Jehad" Baeth "Ahmad Fawzi"

UNIVERSITI UTARA MALAYSIA
2010

IA-BASED FAULT DETECTION SYSTEM

This thesis submitted to the Graduate School in partial fulfillment of the requirements for the degree Master of Science (Information Technology) University Utara Malaysia

By

"Mohammad Jehad" Baeth " Ahmad Fawzi" (802379)

KOLEJ SASTERA DAN SAINS
(College of Arts and Sciences)
Universiti Utara Malaysia

PERAKUAN KERJA KERTAS PROJEK
(Certificate of Project Paper)

Saya, yang bertandatangan, memperakukan bahawa
(I, the undersigned, certify that)

MOHAMED JEHAD BAETH "A. FAWZI"
(802789)

calon untuk Ijazah
(candidate for the degree of) **MSc. (Information Technology)**

telah mengemukakan kertas projek yang bertajuk
(has presented his/ her project paper of the following title)

INTELLIGENT AGENT BASED FUALT DETECTION SYSTEM

seperti yang tercatat di muka surat tajuk dan kulit kertas projek
(as it appears on the title page and front cover of project paper)

bahawa kertas projek tersebut boleh diterima dari segi bentuk serta kandungan dan meliputi bidang ilmu dengan memuaskan.
(that the project paper acceptable in form and content, and that a satisfactory knowledge of the field is covered by the project paper).

Nama Penyelia Utama
(Name of Main Supervisor): **PROF. DR. KU RUHANA KU MAHAMUD**

Tandatangan
(Signature) : _____

Tarikh
(Date) : 29 April 2010

PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a postgraduate degree from University Utara Malaysia, I agree that the University Library may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence by the Dean of the Graduate School.

It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to University Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part should be addressed to:

Dean of Graduate School

University Utara Malaysia

06010 UUM Sintok

Kedah Darul Aman.

ABSTRACT

The current IP-network management systems normally utilize the centralized (client–server) architecture. Researchers have stated that, those systems may cause serious efficiency defects, when the complexity and size of the network increases. Automated network monitoring systems have some limitation. They are known of making a huge overload on the network bandwidth due to their unnecessary message transaction between the server and the monitored hosts. Moreover, the lack of getting appropriate information that describes the malfunction makes it hard for the administrational team to identify the source of reported error. An innovative distributed intelligent agent based fault detection system that operates on Windows platform was presented, to capture abnormal and faulty behaviors on both application and system levels. The design process of the intelligent agent utilized the ability of reactive operating and independent decision taking. The system has a web based graphical user interface to facilitate the accessibility to such vital information. Several evaluation scenarios were conducted to evaluate the trustworthiness and performance criterion of the proposed system.

DEDICATION

In the name of Allah, the most merciful and compassionate.

All praise is due to Allah, the most Generous and loving, the source of all blessings.

To the symbol of wisdom, my mentor, and the source of my existence. To my father "Ahmad".

Heaven lies under your feet along with my happiness and success in this life and the afterlife, there are no words that can honor you enough. To my mother "Sahar".

To Eyad, only Allah can pay off my debt to you, thank you for turning me from a JGDAF into me.

To my sisters Bara'a, Tayma, Thara'a, Yousr.

To my friends Fadi, Alla, Ahmad, Hamzeh, Dia'a, Osama, Zyad, Hussam, Faisal, Chris, Hashem, Rony, Samer, Mohamad, Homam, Morhaf, Taha, AbdulGhani and All those I have not mentioned, thank you for your continuous support.

ACKNOWLEDGEMENT

At the beginning of my speaking, I thank Allah for helping me in my study and guiding me to continue what I have started in my educational life. I thank Allah in every day for giving me the ability and motivation to continue this work...

After thanking Allah, I would like to convey my regards to my supervisor Prof. Dr. Ku Ruhana for the benefit and precious information that she gave me as one of her students. I thank and honor her for helping me to complete my study in a good way...

Finally, I would like to say thankfulness word for the lecturers in the Information Technology Department at University Utara Malaysia (UUM)...

Thank you UUM...

"Mohammad Jehad" Baeth "Ahmad Fawai"

2010

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

The development and growth of computer networks concepts and technology have paved the way for developing new applications in this field of study. Considering the high demand of business organizations for improvements in network management, the wave of development was directed towards creating more advanced network management systems this is to simplify and speed up administrational responsibilities as well as observing network hosts in a real-time basis in order to protect the network from faults. Generally, network management system can handle problems concerning network configuration, tune reliability and efficiency issues, even more increase security standards (Cisco Systems, 2004). In other words, network management system is concerned with monitoring, analyzing and controlling a network, serving the purpose of smoother operation.

Network monitoring is the portion of network management, which is concerned with detecting symptoms of failure and analyzing the status and behavior of the managed network devices. It is getting vital for computer networks to perform in the best manner (Boutaba and Xiao, 2002). A typical network management system consists of two parts as shown in Figure1.1.

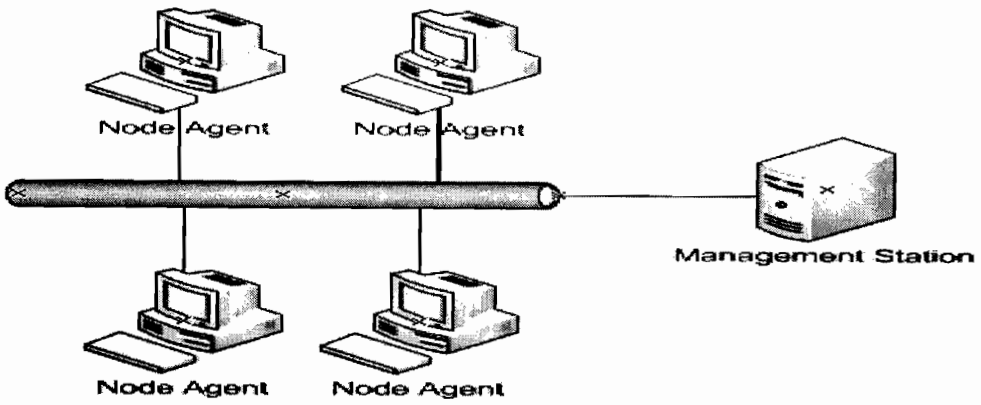


Figure 1. 1 : Conventional Network Monitoring System

One is the agent, a server process operating on every existing node in the network. The task of these agents is to gather and store data related to the status of the monitored device. Collected data is stored in the Management Information Bases. The other part of a network management system is the management station. It has graphical user interface to aid administrators to perform monitoring functions. Managers reclaim data acquired from network hosts agents by sending requests to every single entity. This process is either done on demand or depending on a periodic basis (Boutaba and Xiao, 2002). Depending on this architecture, many advanced network management application and development packages have been developed. The aim of this advancement was to introduce more sophisticated management systems, ones that can handle larger scale networks. However, the majority of these systems have some drawbacks and limitation that can affect the network performance (Gavalas, Greenwood, Ghanbari, and O'Mahony, 2000). These issues have been illustrated in chapter two in details. As a result, the idea of automating portions of administrational tasks by planting intelligence into these monitoring systems began to appear as the needed solutions. Consequently, intelligent agent was the suitable

choice to fulfill the requirements of intelligence (Liotta & Pavlou, 2002).The agent is a part of software system that interacts and cooperates with other agents (including both people and software). The agent interacts with other agents to hold the responsibility of taking actions and deciding if the action is appropriate or not. Agents do not invoke themselves during execution, instead they activate themselves automatically. An agent is an entity which is able to solve the problems automatically using the main features of the agent. These features are independence, social ability and reactivity. The independence is one of the most important characteristics of the agent, in other words, the agent work independently with no interaction with the human or external actors. Moreover agents have the ability to manipulate their status. The second important characteristic of agent software is the social ability, which means agents are considered to be interrelating with other agents (and possibly humans). However, the agent's communicate together using special protocols. Moreover, the reactivity of the agent system is the ability to observe their surrounding environment, and take actions considering the time factor of affecting their environment (Franklin and Graesser, 1996). Agents are classified generally according to their mobility, functionality, skills and location. However an agent if classified under one criterion, does not mean that it may not belong to another criteria (Wooldridge, 2002).

According to Zambonelli, Jennings and Wooldridge (2003) intelligent agents systems are "application software that are intended and developed in terms of self-directed software entities (agents) that can swiftly achieve their goals". An intelligent agent can provide the user with information helps with the decision making process. In order to achieve the

agent tasks, it has to be placed in a compatible environment, where an agent was developed to interact with. However, an agent is affected by the surrounding environment events, which these events may affect the time and nature of its actions. Furthermore, whenever an agent has performed a certain task to achieve a specified goal, a new objective will be set and pursued by the agent.

A distributed agent interacts between modules such as software, hardware or agent and concrete functionality to achieve the complex objectives. These agents are implemented in different machines to communicate together through a network to allow a distributed scheme.

Multi-agent system (MAS) is a system consists of multiple interacting intelligent agents. They can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to handle (Collinot and Drogoul, 2007). Another type of agent is a Mobile agent. The main idea of a Mobile agent is to perform a client server tasks by transmitting an executable program on demand. They are described to hold dynamic services, in which they have the ability to transmit themselves over the networks in order to perform specific tasks. Moreover, they are programmed using scripting languages. However, A mobile agent can be dispatched from a client computer and proceed to a remote server computer for execution (Collinot & Drogoul, 2007). In order for a mobile agent to emigrate over a network, the server has to allow such autonomous actions. Mobile agents can interact with each other in both synchronous and asynchronous ways, according to the server roaming and messaging support (Harvey, Iosif, Julian and Charles, 2001).

1.1 Problem Statement

According to Stallings (1999) the current IP-network management systems normally utilize the centralized (client–server) architecture. Researchers have stated that, those systems may cause serious efficiency defects, when the complexity and size of the network increases (Liotta, Pavlou & Knight, 2002).

Numerous studies have been conducted in the area of automated network monitoring. As a result, many systems that can handle such heterogeneous systems in a rapid change of the network structure have been introduced. However, those systems have some limitation. They are known of making a huge overload on the network due to their unnecessary message transaction between the server and the monitored hosts (Gavalas, Greenwood, Ghanbari, and O'Mahony, 2000). Moreover, as the malfunction reports are being given by users who may lack the experience of knowing exactly what is wrong, makes it hard for the maintenance team to identify what exactly went wrong and how serious it is (Bianchini, 2003). The lack of getting appropriate information describing the malfunction when an error is reported makes it hard for the maintenance team to make a correct assessment of the potential risk. Nevertheless, the delay of fixing some problems or errors may have catastrophic effects, if not handled immediately. Therefore, an efficient model, especially an automated monitoring to perform on heterogeneous networks is needed.

1.2 Research Objectives

The purpose of this study is to:

- Identify the suitable agent taxonomy for the network monitoring and fault detection model.
- Construct a model of an automated fault diagnosis and reporting system.
- Design the diagnosis agent for the system.

1.3 Significance of the Study

The IA fault detection system saves time, cost and effort of the extensive maintenance operation tasks, as well as the prototype ensures a more secured and more functional network. This prototype leads into an improved quality of the network service, and getting better security standards over the network from outsiders' intrusion attempts as well.

1.4 Scope and Limitations

The scope of this study focuses on supporting the task of the UUM maintenance operational team, to get more efficient information of the malfunction of over the intranet, even without human involvement.

The prototype was developed by using JAVA programming language, which is considered to be an independent platform language that can be executed under any operating system, as well as JAVA support for multithreading in distributed

programming is one more advantage to be taken into consideration. The security provided by JAVA for developing networking application is needed in such systems.

Another programming language that is used in developing the prototype is C language. It will hold responsibility of extracting diagnosis information from the operating system by utilizing an existing API "windows.h". The C subsystem will be integrated with JAVA using JAVA NATIVE INTERFACE "JNI", which is a programming framework that allows JAVA code running in a Java Virtual Machine to call and to be called by native applications and libraries written in other languages, such as C, C++.

The prototype can diagnose and report a sample of errors and malfunctions, such as programs halts, compatibility issues, system unplanned restarts and hardware drivers related issues.

The process itself however will have some challenges and limitations concerning the heterogeneous architecture of network in which workstation maybe running under different platforms, operating systems, and even a different set of hardware, moreover the rapidly increasing number of bugs and errors requires a huge effort to classify each one of them.

1.5 Organization of Report

This section provides a general overview for each chapter. This study falls into six chapters; Introduction, Literature Review, Methodology, System Analysis and Design, Results and Discussion and Conclusion.

Starting with the introduction, an overall idea of this study will be gathered in the readers' mind. Explaining some terminologies that have been used in this study will make it easy to understand this work.

From the literature chapter two, the main concept of network management and monitoring, and their applications have been clarified. As well as showing the different agent models that has been utilized in the network monitoring systems. Chapter three presents the methodology that has been applied in this study, which has been adopted from the Rapid Application Development approach.

Chapter four includes the system analysis and design to clarify the system specification.

The experiments applied in this study can be found in Chapter five, there are two main experiments; one is measuring the system impact on the network efficiency, and the other one is to measure the system impact on the host performance. A conclusion and future work are presented in Chapter six.

CHAPTER TWO

LITERATURE REVIEW

This chapter presents an inspection of the different approaches used for fault detection and analysis. It also shows intelligent agent different taxonomies and characteristics along with the applications of intelligent agent technology and the contribution of intelligent agent technology in the Fault Detection systems as well.

2.1 Intelligent Agents

The agent is a part of software system that interacts and cooperates with other agents (including both people and software). The agent interacts with other agents to hold the responsibility of taking actions and deciding if the action is appropriate or not. Agents do not invoke themselves during execution, instead they activate themselves automatically. An agent is an entity which is able to solve the problems automatically using the main features of the agent; these features are independence, social ability and reactivity. The independence is one of the most important characteristics of the agent, in other words, the agent work independently with no interaction with the human or external actors and may the agents have the ability to manipulate their status. The second important characteristic of agent software is the social ability, which means agents are considered to be interrelating with other agents (and possibly humans). However, the agent's communicate

with each other using special protocols. Moreover, the reactivity of the agent system is the ability to observe their surrounding environment, and take actions considering the time factor of affecting their environment (Franklin and Graesser, 1996).

Agents are classified generally according to their mobility, functionality, skills and location. However an agent if classified under one criterion, does not mean that it may not belong to another criteria as well, depending on the agent anatomy (Wooldridge, 2002).

2.1.1 Distributed Intelligent Agents

A distributed agent interacts between modules such as software, hardware or agent and concrete functionality to achieve the complex objectives. These agents are implemented in different machines to communicate through a network to allow a distributed scheme. According to Zambonelli, Jennings and Wooldridge (2003), Intelligent Agents Systems are "application software that are intended and developed in terms of self-directed software entities (agents) that can swiftly achieve their goals". An intelligent agent can provide the user with information helps about the decision making process. In order to achieve the agent tasks, it has to be placed in a suitable environment, where an agent was developed to interact with. However, an agent is affected by the surrounding environment events, which these events may affect the time and nature of its actions. Furthermore, whenever an agent has performed a certain task to achieve a specified goal, a new objective will be set and pursued by the agent.

A distributed IA-based network monitoring tool was proposed by Wu, Zhaol (2008) In their research they used the cross-platform language Python to develop the system, the significance of the system is that it uses both of the automatic and manual monitoring modes, the combination of the monitoring modes were used to improve the efficiency of the system, so it wouldn't have recognizable effect on the network reliability. After evaluating the system they found that methods applied in their system can be used in different network topologies. Moreover, they found that the more the anticipated condition close to the original state, the smaller the communication failure probability gets, and the higher the communication are reliable. However they have stated that the experiment subject was a network that consists of 16 nodes, which means that in large scale networks their results would be different and the system may cause an overflow on the network. They suggested utilizing of Multi-IA in order to improve the monitoring capabilities for more complex networks.

2.1.2 Multi-Agents

Multi-agent system (MAS) is a system consists of multiple interacted intelligent agents. They can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to handle (Collinot and Drogoul, 2007).

Gavalas et al. (2000) have presented a research project studying a new a strategy to monitor devices and a tool, called "MoDPAI", which uses computation and intelligent software agents. The strategy divided the network's management into well-defined steps, encompassing stages from the survey of the devices' requirements to the generation of

standardized network monitoring reports. The main characteristic of the tool presented herein is the use of software agents that help the administrator's decision-making activities, speeding up the monitoring and control of the monitored devices. The administrator and agents can exchange information directly through the tool or through pervasive devices connected to the Internet, allowing queries and responses for guidance of the monitoring activities to be carried out at any point. The advantages of using mobile agent technology is that it facilitate the network management tasks, by getting more efficient information faster than the traditional way of diagnosing and management which leads to a more efficient use of computational resources. Moreover agents if given a perceive knowledge, the overload on the network will not be large, and will not affect the network performance. However such agents will not perform efficiently under networks with large diversity of heterogeneously. Empirical results confirmed a significant improvement on traffic overhead when testing their proposed application in realistic management scenarios.

2.1.3 Mobile Agents

According to Gavalas, Tsekouras and Anagnostopoulos (2009) Mobile agents (MAs) are programs that can be dispatched by a client for execution on a remote server. Mobile agents are not an implementation or protocol. It is an infrastructure or framework. To implement mobile agents you can use any protocol and any programming language, the choice is up to the programmer. There are a couple of implementations of the mobile agents framework, the framework consists of four major components. The manager application has a user interface of some kind that shows the status of different network

devices. It can monitor the progress of any MA and can dispatch another. From this application the user has control over the whole network management process .On every managed device there is a Mobile Agent Server. The MAS takes care of MAs it receives from the network and executes them. The framework does not depend on security and can do without it. If security is to be used, the MAS should have the biggest responsibility, because it executes programs (MAs) that can come from anyone. For MA framework to be secure, the MAS need to be able to authenticate users and decide whether the code should be executed or not. The MAS can itself collect management data, but a better idea is to let the MAS only deal with security and execution of the MA and let the real data collection to another protocol like SNMP. There must be someone or something that creates these MAs. MAs can be written by users that are interested in monitoring some important status or they can be generated by other programs. These MAs are stored in a repository where the application can use the one it needs. Because of this flexibility MA have the ability of running new programs on any network device, and the functionality to accomplish management tasks of a constantly changing network environment can by dynamically extended. Mobile Agents are programs, but they also contain their state. When an MA gets back to the manager, after being on the network, the manager wants the information that the MA retrieved and the MA therefore needs state information. An MA can do one half of its work on one device and do the other half on another device. Probably it wants to save the state from the first half to be able to know where to begin the work on the other device. This is another reason for MAs to also contain state.

2.2 Fault Detection and Diagnosis

According to Sakai, Matsuba and Ishikawa (2007) a failure is an incident that arise when the pitched service turn from the flow of an acceptable behavior. Where, an error happen when an element provides the ability or a service does not supply an anticipated reaction. And a fault is the reason that an error arises. Faults lead into errors. Errors lead to succeeding errors, and then a failure take place. Fault detection techniques for network systems observe the condition of procedures and hardware components running on the operating system platform periodically. In these fault detection system, associations among monitoring processes and monitored entities are set via the system configuration. The fault detection system will monitor the defined target using these relationships, and consequently finds the faults.

Fault management is part of network management term, whenever a service or network device fails, the management system shall detect the fault, find the cause and report the failure. In some cases the management system can also restore the service automatically, but most often a network operator has to fix the fault manually. The goal of fault management is to increase the network reliability, discover failures as quickly as possible so a network operator can fix the problem, hopefully even before the network's users notices there is a problem .

There has been many researched conducted in the area of Fault detection and analysis in distributed systems and applications. Fault detection techniques are classified based on

their criteria such as hardware and software techniques and the type of detection schemes such as statistical methods, distance-based .

2.2.1 Hardware/Software Based Fault Detection

Hardware replication and lockstepping to detect hardware faults in microprocessors is very commonly used. However, this fixed division of hardware resources among replicated components has a high expectancy of causing efficiency related problems . They proposed using concurrent and Redundant Threading approach to provide momentary fault reporting, they utilized multiple hardware frameworks of Simultaneous Multithreading. This approach provides better efficiency by using active scheduling of its hardware. Their proposed approach had an improved ability to detect faults. To achieve that first they initiated the field of replication, in order to get an abstraction of both the physical redundancy of a lockstepped system and the logical redundancy of an SRT processor. This helps in discovering the scale of fault exposure and the output and input which requires handling.

Another framework that an employs new technique to enhance fault detection proposed by Ray, Hoe and Falsafi (2001). They proposed a random superscalar data approach by adjusting recursively verifying the outputs of the active paralleled execution threads. Furthermore, they suggested using a branch rewind mechanism for recovery. Their proposed technique called PROFiT, Their proposed approach mainly depended on three key processes. Firstly, they embedded a dynamic instruction injection, which can produce

redundant execution threads. Secondly, they utilized value synchronization to examine the generated instruction results to discover faults. Finally, they periodically saved the state of execution that is identified to be fault free in order to be used as recovery points. This approach finds out faults and performance points of strength and weakness of every running application, and then determines when to activate redundant execution using its profile.

2.2.2 Detection Scheme Based Fault Detection

Fault detection and analysis has been classified into several categories, statistical approach, profiling, and distance-based methods. Statistical approach works by tracing the behavior of the system through approximating a set of defined varying objects periodically. Like tracking the messages transactions of events among components, the system metrics and sessions log. It keeps statistical averages of these variables, and by comparing the standard deviation obtained from the previous reading with the ongoing reading coming from the monitored target it can detect abnormal behavior by identifying if the thresholds have outmatched their limits or not (Kim et al., 2008).

Distance-based methods solve this inadequacy. It has the ability of identifying outliers according to the computing distances between nodes (Cohen, Zhang, Goldszmidt, Symons, Kelly & Fox, 2005). Instead of constantly storing the readings of the system metrics, which gives a fixed way of indexing and retrieving faults diagnosis data, they proposed a better method to generate and store fault reports. Their selective data

collection approach recorded the provenance and significance of these metrics. Their technique has shown how clustering and retrieval techniques assist the diagnosis entity to enhance the outcome of previous work and classify similar frequent problem occurrences. Moreover, their proposed technique has shown improvements in correcting misdiagnosis, even in distributed environments.

2.3 Applications of IA-based Fault detection and diagnosis

Network management systems are characterized by distributed data processing and decision making, which would create a bottleneck if brought to be managed centrally. However, distributed intelligent agents can be delegated the task of network management to perform administrative tasks such as applying network policy, monitor quality of service, etc. In addition to the flexibility, intelligent agents offer considerable advantages by conserving network bandwidth, decentralizing computation load, quickly responding and scaling easily (Boutaba, 2002).

Utton and Scharf (2004) have proposed a system for diagnosing errors over a network of home convenience accessories, such as network links, nodes, devices, and software's Systems. The systems suppose to observe devices, diagnose automatically for faults and errors and then generate reports describes the error which has happened or it may occur. Their article shows a model of the application, and the observations taken after deployment. The experiment shows how the technology of distributed intelligent multiagent can help us solve problems in such heterogeneous distributed environment, where different set of hardware's and software's are the environment of the intelligent

multiagent. The idea of the model utilized the partial maintenance approach where when a fault is diagnosed and reported the agent will diagnose the cause of the problem and then will locate the smallest part in the device, in which if replaced it can solve the problem. However because of the great diversity of the network devices, the multiagents were embedded with different sets of intelligence. In other words the system efficiency and reliability worked fine according to the aimed standards but the computational load of the system was very expensive.

According to Dong-Liang and Sheng-Yuan. (2009) there has been lots of network management application, which effectively can assist with the task of managing and monitoring a network. However, those application shows lack for flexibility in the mechanisms of monitoring a network, and the diagnosis of network status. Consequently, they proposed a new approach that supports the network monitoring, and utilizing the multi distributed intelligent agents along with the SNMP protocol. They developed a prototype the implement their proposed approach using JAVA along with Knowledge query manipulation Language to handle the communication among the agents. They used a MySQL database to save data being collected by the agents. The system was deployed in centralized network architecture. Their prototype has shown an increased robustness due to the usage of only open source code. Moreover, the prototype showed reeducation of both computational workload and network workload. However, the system needed further expansion regarding the report generation methods.

The development of Mobile Agent-based applications is fraught with risks. Due to the mobile agent ability of immigrating across network nodes and perform different task on

different hosts efficiently, security problems may occur. These problems become more serious when mobile agent is used for network management purpose. Satoh (2002) have addressed these problems along with other defect that may occur when developing a mobile agent-based network management system. As a response they proposed a new approach for developing such applications. His model consisted of two types of mobile agents. One is the task agent, which has a specific knowledge that enables it to handle a specific type of tasks. The other one is the navigator agent which is more familiar with the topology of its target sub network, by that it enables the navigator agent to handle issues that is related to a certain domain in the network. This dual mobile agent mechanism was called Agent pool. It explicitly selects the appropriate agent for the different sub networks located in the managed networks domain. He has designed and implemented a prototype based on His proposed approach. The designed prototype was implemented using Mobile spaces, taking in consideration the value can be added by using Mobile spaces, which resembles with the strengthening the conservation of the mobile agents state. Moreover mobile spaces being developed under JAVA infrastructure, it holds the same values of JAVA programming. The agents used a TCP-based agent migration protocol to perform their movement on the network environment. The prototype showed great improvements regarding the reusability and performance. Moreover agents were more network independent, in other words agents were able to detect changes in the network topology, and preserve to be used during their next dispatch in a heuristic manner. However the prototype showed quite a limitation when dealing with their migration routes.

According to Bianchini (2003), the needs for decentralize and heterogeneous system management and the rapid decentralization of computational resources have involved the development of strategies and tools to assist the network administrator in great part of the tasks through artificial management system. The MoDPAI come as a result of the strategy to monitor devices and tool, Intelligent system agent and pervasive computation was the mainly used to produce the prototype. Based on the prototype developed, the system for monitoring network devices using computing and intelligent software agent (MoDPAI), modeled based on the catalysis method techniques and programmed with JAVA programming. The layers that had been used in this prototype are offers the Graphical User interface or Wireless User Interface, shows the communication through the SNMP protocol and the function of the ISA, the third layer is the file system and the Database to storage the all information related to the monitoring such as knowledge bases of the agents, devices information properties, and configuration; that for increase the reusability and reduce the code redundancy and facilitating maintenance. The results of their prototype are allowed for more accurate analysis and detection of problems in the network. However, the developed prototype had one disadvantage, which is the need for end users to be familiar with the syntax of KB language.

Ibrahim (2006) have stated in his article that Mobile agent based Network Monitoring applications are still facing a lot of defects and major problems, especially considering the miss coordination among multiple mobile agents when visiting the same location on a network. Moreover, he mentioned the arising problems in coding the mobile agents when dealing with security issues, where security bleaches may change the agent code. He

proposed a new architecture that addresses the risks of code mobility. The key idea was to provide a graphical user interface to view information of managed nodes visiting plan, and then obtain input from users. In other words users had the credentials to request creating mobile agents. The architecture proposed a server agent in which plays the role of providing resources and communication support to the mobile agents. He implemented a prototype based on the proposed architecture using both Java and Aglets. Agents used Aglet transfer protocol to perform migrations over the network. The model shows better security standards especially regarding the agent migrations, and better protection against malicious agents' attacks. However, due the huge load of information the agent collect in every visited node the impact on the network bandwidth was impeccable. He recommended the integration of mobile agents with SNMP architecture.

Pugazendi and Duraiswamy (2009) have acknowledged the role of mobile agents to reduce network traffic and providing more effective approach of network monitoring. They have utilized the asynchronous operating, independent execution of a mobile agent to develop a monitoring system. Their aim was to develop a robust and fault tolerance system that can perform the tasks of network monitoring. They developed their tool using the extreme programming approach by identifying objective of the system and applying them immediately in programming code. The system contained the both static and mobile agent. The mobile agent, when generated will hold a list that contains names of the node which the agent will migrate to, using Agent Transfer Protocol for dispatching. They migrate to the specified node to collect required data, and then returns to the manager agent to dump these data for processing and analyzing operation. The static agent part

was meant to be located on every monitored node on the network. Their basic mission is to wait for the arrival of mobile agents, and hand over the required data to the mobile agent, after filtering these data. The manager agent mission is to dispatch mobile agents to collect data, and process data collected by these agents. After processing data the manager could present two types of views. A graphical and tabular view, and an event record for every node as well. By using this architecture they were able to avoid several problems that were faced in previous agent based-monitoring system. Problems like agents return wrong results of the agent call, and the incorrect execution of the migrated code. The architecture highly improved the security standards.

The direction of developing computer system is driven into the development of more dependable and fault tolerance applications. However, the mechanisms used to detect failures in those applications are not capable of detecting faults caused by factors from outside their specified area on both of the hardware and software levels. In which, such undetectable faults can have catastrophic effect in critical computer-driven systems (Baldini, Benso, Chiusano & Prinetto, 2000). They have stated in their paper that, there have been many studies conducted for the purpose of investigating the dependability validation under the UNIX operating system, not many have been conducted in the same field under windows operating system. Considering these reasons, they have proposed a fault diagnosis tool capable of operating under WIN32 platform, the tool called BOND. The objective of the proposed tool was to measure the flexibility and stability of a computer application by performing fault tolerant validation procedure after injecting the application with artificial errors. The tool design consisted of two major components.

One is the logger agent which is responsible of detecting faults caused by debugging, APIs calls or wrong memory access. The logger agent is responsible of synchronizing the fault injection towards the targeted application and then observing the behavior of this application. The other major component of BOND is the fault injection agent, which is responsible of injecting faults into the target application, according to the location of the fault, fault type and fault duration criteria's. The diagnosis reports results were classified into no effect, an application crash or a silent failure. They have preformed a simulation to evaluate the functionality of the program. The results have demonstrated the power of the tool by injecting faults on different application contexts and evaluating the immunity of the targeted application.

Large scale networks diversity of hardware and software component can make the detection of a failure source a very complicated task. In such distributed environments errors can propagate, where an error can cause consequent errors over the other connected hosts on the network, a fault monitoring system then will detect a numerous number of errors all over the network, where all of these report are considered to be worthless since it does not specify the origin of the fault. Moreover, the existing fault detection systems reconfiguration to handle such issues is hard work (Sakai, Matsuba and Ishikawa, 2007). They proposed a fault detection system that addresses these issues. Their proposed system is able to identify the source of an error using the relations mapping the sets of reported propagated errors using the error relationship tree. The error relationship tree was introduced to link propagated errors with the source error and the type of hardware reported with the fault. The system is connected to a database that

contains information about connected entities and software's, the network topology as well. The system terminology consisted of a database adapter responsible executing SQL queries to obtain information of the reported faulting entity, a simple command executer which lunches a tool that diagnose the fault and generate error reports. Lastly, results manager which handles the task of generating graphical reports. The prototype was implanted to perform under LINUX platform. The detection engine was implemented in C language, while the detection tools are implanted using Linux shell commands. The proposed system had the advantage of identifying the source of errors, and the reconfiguration of the error relation tree is far more simple task than conventional fault detection system reconfiguration, in which supports the system flexibility in handling a wide set of errors.

2.4 Summary

Based on the traditional diagnosis systems approaches, combined with the intelligent Agent technology, a distributed intellectualized have proven to add important new abilities to the fault detection approaches. Although many of the proposed system have shown enhanced effectiveness and flexibility, the trade-off between performance and precision is still an issue facing such systems.

CHAPTER THREE

METHODOLOGY

This chapter shows the process phases which are followed to obtain the intended results of the project. It describes the step by step methodology that we have followed starting from getting the tentative design followed by the development process of the system and the evaluation procedures conducted.

3.1 Research Phases

The research design methodology used in this thesis has been adopted from the general methodology for design and research proposed by Vaishnavi and Kuechler (2004). Figure 3.1 shows steps of the adopted framework, the output of each step, and how the flow of the framework achieves the project objectives.

Awareness of Problem is conducted during the preparation of the research proposal. Concept Design phase follows immediately behind the proposal. It outputs a tentative design. Tentative design is likely to make the key role player in affecting the performance of the prototype based on that design would be an integral part of the Proposal. Development phase takes the tentative design and implement it. The techniques for implementation will of course vary depending on the artifact to be that is being constructed to fulfill the requirement specified. Evaluation phase is conducted once the system is constructed. The system is evaluated according to a criterion that is commonly applied to fault detection systems. The evaluation phase contains an analytic sub-phase in which hypotheses are made about the behavior of the proposed system. Conclusion phase

is the finale of a specific research effort. Typically, it is the result of contribution made based on the obtained results.

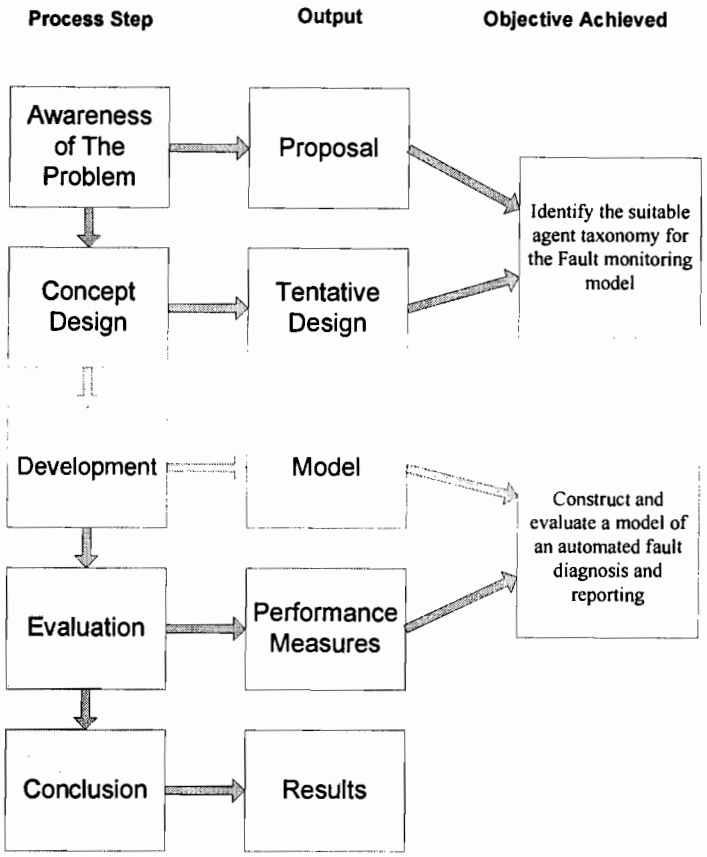


Figure 3. 1 : Research Design Methodology Adopted from (Vaishnavi & kuechler, 2004)

3.2 Concept Design

After conducting literature review a clear formulation of the aspects of the problem has been acquired, and the suitable design of the agent architecture as well. Moreover, addressing the operating system and the application running under Microsoft platform

faults, Microsoft support center "Microsoft TechNet" contains information that helps identifying the aspects and symptoms of each error and the risk an error may hold.

3.3 Development

Like most of the OO-SDLCs RAD works in sequential order, Rapid Application Development consists of only three phases in which are running sequentially, the phase of Planning Requirements, Design Workshop phase and the Implementation phase. Figure 3.2 shows the Methodology steps. The choice of Rapid application development was due to its simplicity, less time consumption, speed of execution and quality results .

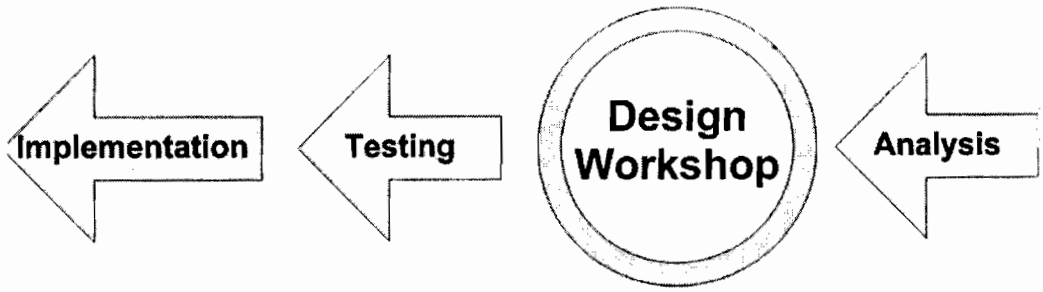


Figure 3. 2 : Rapid Application Methodology Life Cycle

The proposed model consists of a diagnosis intelligent agent and the server side. The diagnosis IA resides on every connected node on the network. It collects fault reports caused by different sources and transmit over to the server side. The server side listen to a specified port on the network, classify the hazard of the fault and store reports in a database for further future diagnosis of hosts state. It has a user interface connected on a

web server to insure accessibility over the network. Figure 3.3 shows the architecture of the system.

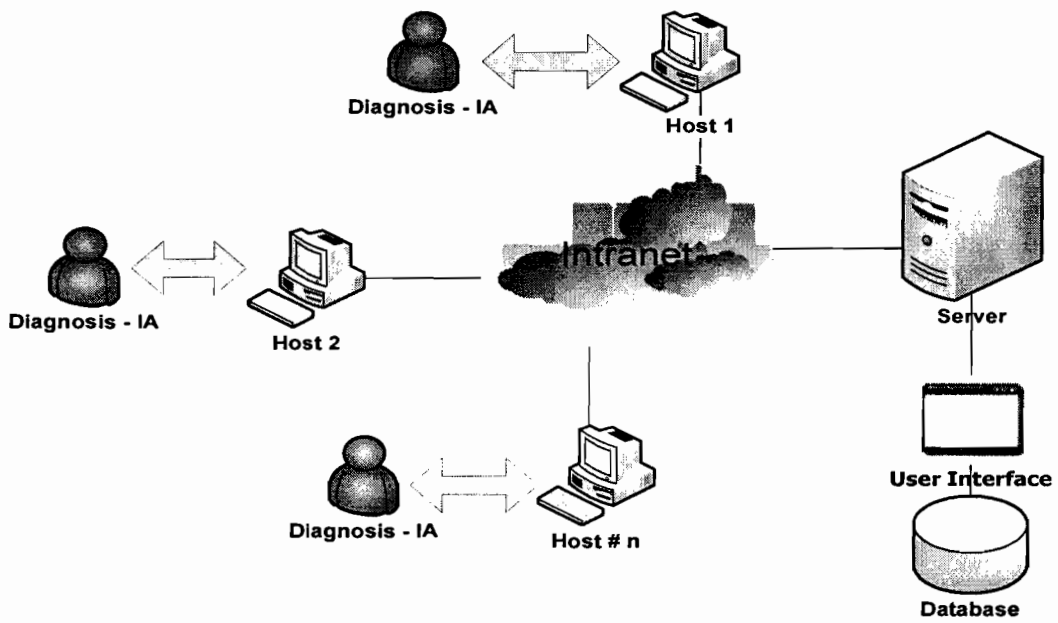


Figure 3. 3 : Intelligent agent-based fault detection system

3.3.1 Designing Agent Model

In order to get an optimal design of the agent model, a definition of the agent role model has to be set. Agent role model describes the characteristics of the agent functions and capabilities.

Defining the agent role model is done by the recognition of four aspects:

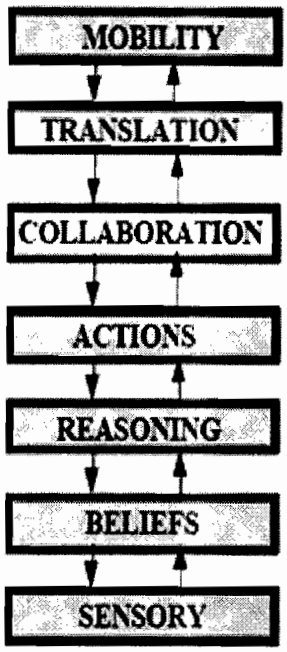
- ❖ Responsibilities of the agent: Which are translated into function during the implementation phase
- ❖ Permissions: Specifies the limitation on the resource that the agent can access
- ❖ Activities: It defines the tasks that can be performed by the agent without outsiders involvement
- ❖ Protocol: This defines the interaction method among agents

Furthermore, a definition of the interaction model has to be set also. Interaction model shows the protocol agents have to follow during the execution of their process. It is derived from the initial role model by giving a clear description of how an agent responds to an event, lunch a role or deal with captured information.

According to Buschmann (1996) Agents should be deconstructed into layers because of their higher level of behavior depends on their lower level capabilities, and because of their messaging architecture in which There is two way information stream between each two related levels.

As in their research they have shown the basic layered architecture of the distributed intelligent agent structure.

- Mobility Layer: responsible for bringing in the messages coming from distant agents
- Translation Layer: translate incoming transmissions.
- Collaboration Layer: determines whether an incoming message should be processed or not.
- Actions Layer: responsible for taking in pending action
- Reasoning Layer responsible for reading the selected action
- Beliefs Layer: responsible for updating beliefs according to reasoning
- Sensory Layer: responsible for gathering regular sensor updates



The suggested methodology of development refers that different prototypes will be produced by the end of every cycle according to the feedback of the administrators. However, the model of our diagnosis agent will be fixed as shown in Figure 3.4.

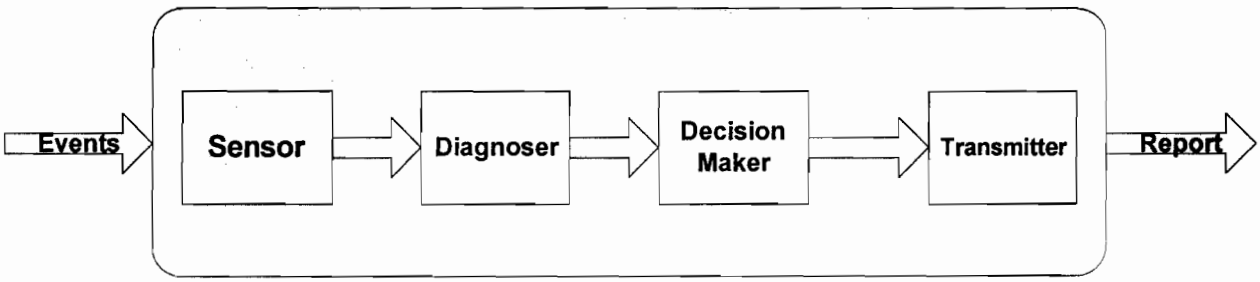


Figure 3. 4 : Agent Model

The model shows that the predictive agent will be divided into four sub agents to each its own specific task, in which the task of the other agents depends on its results. The Sensor sub Agent, responsible of observing the status of the computer the agent running on,

reading the computer performance meters and collecting information about errors rising, notifications coming from the system, in which may be reported to the user or only preserved in the system log. Diagnosis agent responsible of reading information coming from the sensor agent, its task is to translate these data, classify and purify it. Decision making Agent , responsible of taking hold of data coming the diagnosis agent for further analysis in order to either generate a report or wait for more information. Transmitting Agent which is holds the responsibility of communicating with the server agent. It will be activated in two cases either to report to the system that the agent is alive or to send reports coming from Decision making Agent.

Kendall, Krishna, Suresh and Pathak (2000) recommended a new framework that utilizes the of OOP methodologies in the development of an agent based system for network management, with an object oriented representation. The results of the research shows that it is possible to use an object oriented methodology for developing models and application such as Distributed Intelligent agent systems, by applying the following Design patterns through the Design phase of the development process each to its respected layer:

Table 3. 1 : List of Design Pattern to be used with their respective Agent Layer (Kendall et al., 2000)

Layer	Design Pattern
SENSORY	-Adapter pattern
Beliefs	-Composite pattern
REASONING	-The Interpreter and Strategy patterns
ACTION	- Command is used to make a plan into a command. - Abstract Factory creates a plan object based on a given class library. - Factory Method creates intention thread objects dynamically. - Decorator implements the Prioritizer. - Future with Observer
COLLABORATION	- Synchronized Singleton is used to manage the collaboration threads. - Decorator changes the behavior of the thread dynamically

3.3.2 Server Design

Figure 3.5 shows the design of the server side model, it consist of four major components, the listener opens a port on the network interface and listen on that network in order to collect incoming fault reports transmitted from the distributed diagnosis agents. Where the organizer takes charge of classifying incoming faults reports according to the report host source and the criticality of the report, it is also in charge of handling SQL queries performed on the reports database. The reports database is a repository of the previous reports, where every fault report will be stored in the database for the purpose of preserving these reports, using it for user requests. The final part of the server side is the

JSP user interface, which is a graphical user interface, implemented using JAVA SERVER PAGES and located on a web server for remote access, it displays information regarding the status of every connected host on the network and that's where the final form of the report is shown.

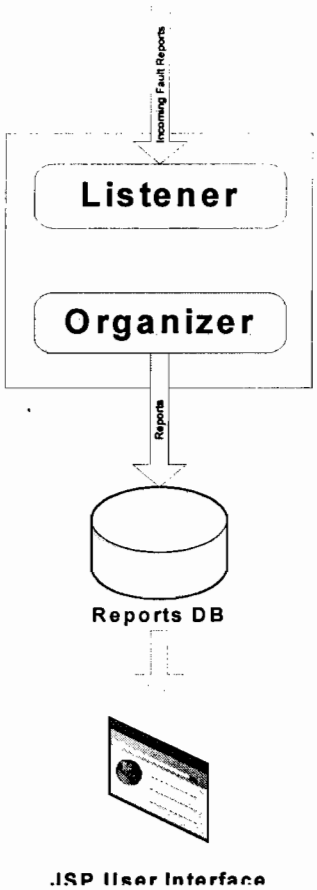


Figure 3. 5 : Server Side

3.3.3 Implementation

Conducting the implementation is based on the results gained from the design phase where every model is realized into a working prototype. However, the implementation of such system requires a programming language that can adapt with different operating

systems platform, a wide support of networking and the ability for a concurrent execution. Moreover, realizing the need for a fully object oriented support to facilitate the implementation of design patterns specified under the design of agent model section. JAVA programming language can fulfill all of these requirements and so is appropriate choice for implementing such system.

Since WIN32 is the agent environment of operation, the agent is required to fully interact with the operating system components in order to collect the needed data for diagnosis. However, JAVA does not have a direct support of WIN32 application.

The Java Native Interface (JNI) permits the combination of code written in the Java programming language with code written in other languages such as C and C++. It allows programmers to take full advantage of the Java platform and enabling JAVA of interacting with WINDOWS components.

JNI allows one to write native methods to handle situations when an application cannot be written entirely in the Java programming language. This allows all Java applications to access this functionality in a safe and platform-independent manner. Before resorting to using JNI, developers should make sure the functionality is not already provided in the standard libraries. The JNI framework lets a native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code.

Considering these features, the sensory layer of the agent has been implemented using JNI. The native part of the sensory is implemented using C language utilizing the windows.h API for the purpose of reading the operating system indicators, and then transforming these data into a JAVA integrated form to carry out the diagnosis tasks.

3.4 Evaluation

In this section, we illustrate various scenarios and evaluate the anomaly detection capabilities of the system using fault injection into the experiment subject. Faults selection was made on both application and system levels. Moreover, faults selection was extracted from real life situations. After conducting fault injection into the experiment subject, we will observe the proposed system response to these artificial errors, and the system effect on the network performance.

3.5 Summary

The agent based fault detection system architecture has been discussed in details in this chapter, the system consist of two parts formulating a client-server architecture, where the agent is embedded with intelligence in order to operate the operating system metrics collection and the identification of faulty events. The agent was designed under several desing patterns in order to divide it into reusable components which makes it easier to update its set of beliefs. The server had a web based graphical user interface connected to a database to store faults reports coming from agents distributed across the intranet.

CHAPTER FOUR

SYSTEM ANALYSIS AND DESIGN

The system analysis and design chapter contains Logical Design and Physical Designing. Logical designing describes the structure and characteristics or features. The physical design shows the actual software and a working system.

4.1 System Requirements

Listed below are the functional requirements and non-functional requirement of the system. In the priority column, the following short hands are used:

- M – mandatory requirements (something the system must do)
- D – desirable requirements (something the system preferably should do)
- O – optional requirements (something the system may do)

4.1.1 Functional Requirements

Table 4. 1 : list of Functional Requirements

	Requirement ID	Requirement Description	Priority
	IMS_01	View Latest Error	
1.	IMS _01_01	Admin can view the latest error for all the Hosts.	M
2.	IMS _01_02	Admin can view the latest error for each Host.	M
	IMS_02	View Network Error Statistics	
3.	IMS_02_01	Admin can view the host's errors.	
4.	IMS_02_02	Admin can view the number of error in all hosts.	
	IMS _03	View Host Error Statistics	
6.	IMS _03_01	Admin can view each host error.	M
7.	IMS _03_02	Admin can view the number of errors in each host.	M

8.	IMS _03_03	Admin can view the risk type for each error in each host.	D
9.	IMS _04	Collect and send Host Status	
10.	IMS _04_01	Agent will move between the hosts.	M
11.	IMS _04_02	Agent will collect the data about each host status.	M
12.	IMS_04_03	Agent will send the status for each host to the server.	D
	IMS _05	Identify Event	
13.	IMS _05_01	Agent can identify the source and problem for each error.	M
14.	IMS _05_02	Agent can determine the risk type for each error.	D
	IMS _06	Organize Collected Data	
15.	IMS _06_01	Agent will arrange the errors.	M
16.	IMS_06_02	Agent will organize the handled process.	M

4.1.2 Non-Functional Requirements

Table 4. 2 : List of Non-Functional Requirements

No.	Requirement ID	Requirement Description	Priority
	IMS _07	SECURITY	
17.	IMS _07_01	Only the Admin can see hosts records.	M
	IMS _8	PERFORMANCE	
18.	IMS _8_02	The system should be available all the time.	O
	IMS _9	OPERATION	
19.	IMS _9_01	The system will operate in Windows environment.	D
	IMS _10	RELIABILITY ISSUES	
20.	IMS _10_01	If the systems crash, it should behave perfectly normal when reloaded again.	M

4.2 Use Case Diagram & Specification

Figure 4.1 shows the use case diagram presenting a graphical overview of the functionality provided by the IA-based fault detection system.

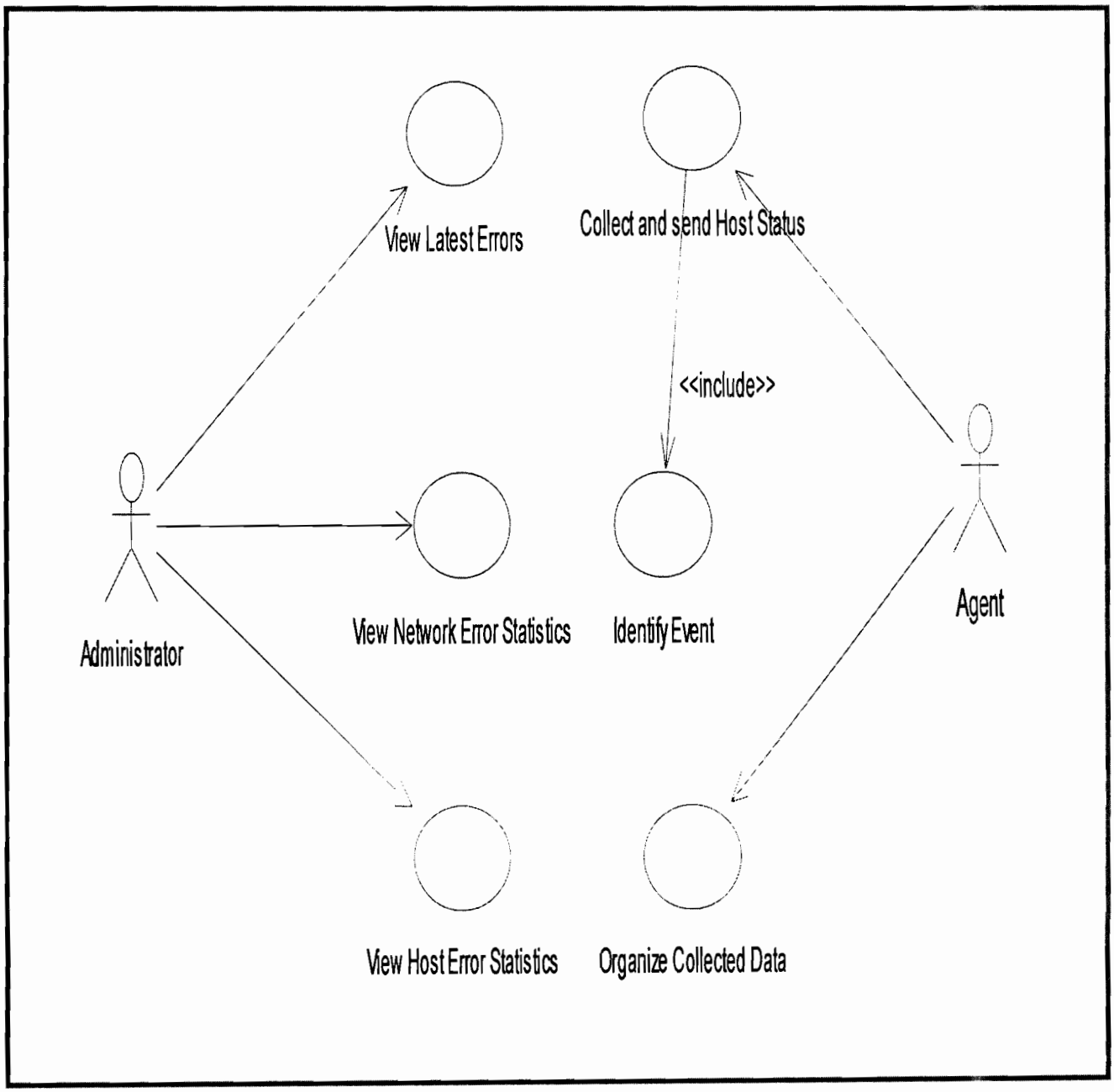


Figure 4. 1: Use Case Diagram

4.2.1 View Latest Errors

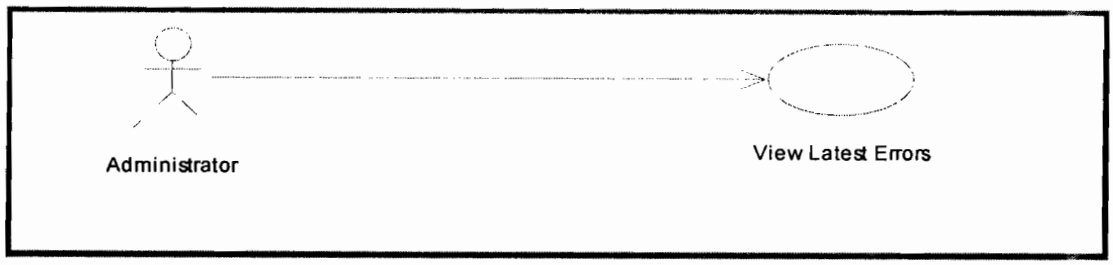


Figure 4. 2: View Latest Errors Use Case

Description:

- This use case allows Admin to view the latest errors for all the hosts order by the risk held in it.

Actor:

- Admin.

Requirements:

- IMS_01

Pre-Conditions:-

- Not Applicable.

Post-Conditions: -

- Not Applicable.

Flow of events -

Basic flow: -

- The Use case will begin work. When the Admin press host button.
- System shall display the host page.
- System will display the list of latest error for all the hosts.
- System will order the list of errors, according to the error risk.

Alternate flow:

- Not Applicable.

Exception flow:

- Not Applicable.

Rules:

- Not Applicable

4.2.2 View Network Error Statistics

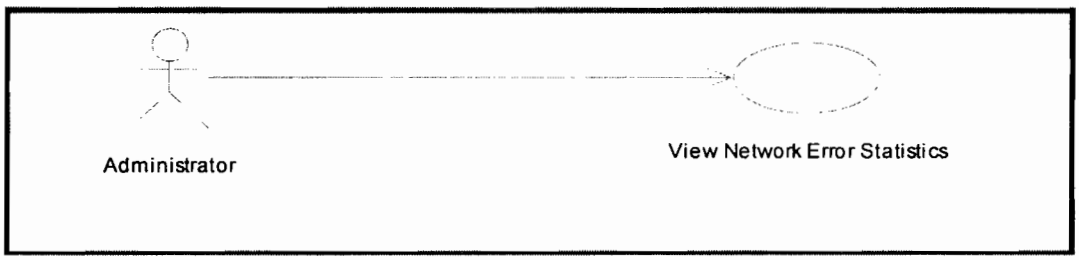


Figure 4. 3: View Network Error Statistics Use Case

BRIEF DESCRIPTION

- This use case is initiated by the administrator; this use case will enable the administrator to view all the hosts' errors, and the number of these errors.

ACTORS:

- Admin

REQUERMENTS:

- IMS_02

PRE-CONDITIONS

- Not Applicable.

FLOW OF EVENTS

Basic Flow (CBE_02_01)

- The Use case will begin work. When the Admin press chart button.
- System will display the list of error for all the hosts.
- System will order the list of errors, according to the host address.
- System will display statistical chart, for all the hosts.

Exceptional Flow

- Not Applicable.

POST-CONDITIONS

- Not Applicable.

RULES

- Not Applicable.

CONSTRAINT(S)

- Not Applicable.

4.2.3 View Host Error Statistics

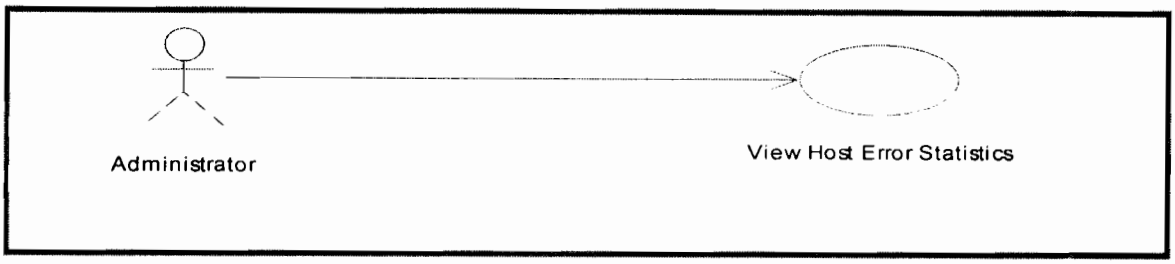


Figure 4. 4: View Host Error Statistics Use Case

BRIEF DESCRIPTION

This use case is initiated by the Admin. This use case will enable Admin to view the list of errors for each host.

ACTORS:

- Admin.

REQUERMENTS:

- IMS_03

PRE-CONDITIONS

Admin has to access the host page by click the host button on the homepage, before start this process.

FLOW OF EVENTS

Basic Flow (CBE_02_01)

- The Use case will begin work, when the Admin select one of the host's addresses.
- System shall display the list of errors for the host.
- System will order the list of errors, according to the risk error.
- System will display statistical chart, for the hosts.

Alternative Flow

- Not Applicable.

Exceptional Flow

- Not Applicable.

POST-CONDITIONS

- Not Applicable.

RULES(S)

- Not Applicable.

4.2.4 Organize and Collect data

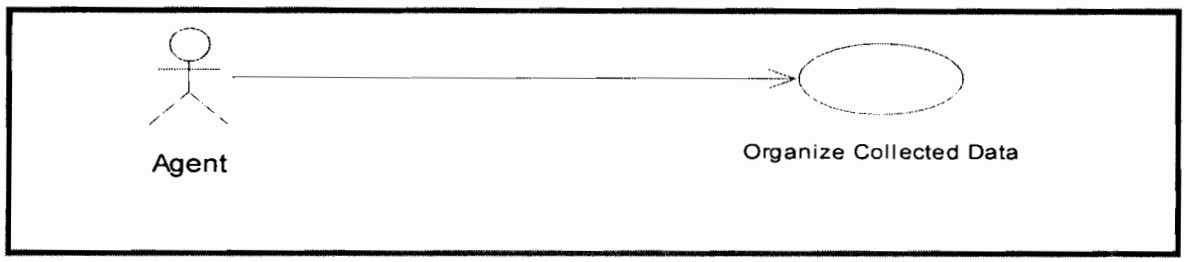


Figure 4. 5: Organize and Collect data Use Case.

BRIEF DESCRIPTION

- This use case is initiated by the Agent. This use case will enable Agent to collect the data from all hosts, and send it to the server.

ACTORS:

- Admin.

REQUERMENTS:

- IMS_04

PRE-CONDITIONS

- Not Applicable

FLOW OF EVENTS

Basic Flow (CBE_02_01)

- The Use case will begin work, when the system begin run.
- Agent will collect data from each host.
- Agent will organize collected data in the host report.
- Agent will send the collected data to the server.

Alternative Flow

- Not Applicable.

Exceptional Flow

- Not Applicable.

POST-CONDITIONS

- Not Applicable.

RULES(S)

- Not Applicable.

4.2.5 Collect and Identify

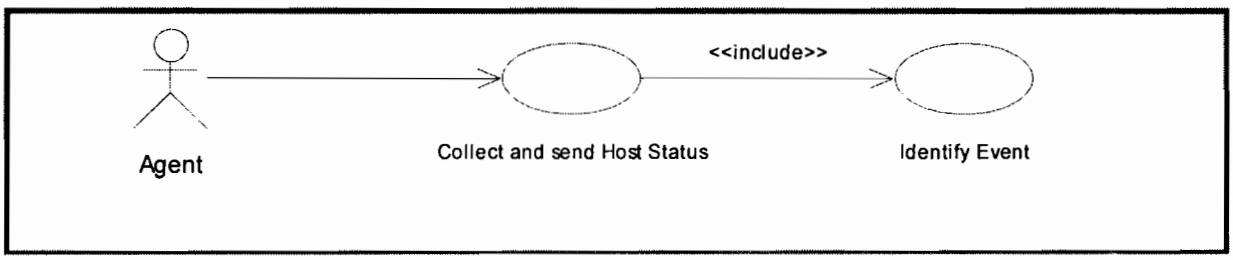


Figure 4. 6: Collect and Identify Events Use Case

BRIEF DESCRIPTION

This use case is initiated by the Agent. This use case will enable Agent to manage the host event, check if there is an error or not, and identify each error.

ACTORS:

- Agent

REQUERMENTS:

- IMS_05

PRE-CONDITIONS

- Not Applicable.

FLOW OF EVENTS

Basic Flow (CBE_02_01)

- The Use case will begin work, when the system begin run.
- Agent will manage all the events on the host.
- Agent will check if there is error or not.
- Agent will identify and determine the error source and risk.

Alternative Flow

- Not Applicable.

Exceptional Flow

- Not Applicable.

POST-CONDITIONS

- Not Applicable.

RULES(S)

- Not Applicable.

CONSTRAINT(S)

- Not Applicable.

4.3 Activity Diagram

4.3.1 View Latest Errors

Figure 4.7 shows activity diagrams describing the behavioral overall flow of control of the system when requesting latest errors.

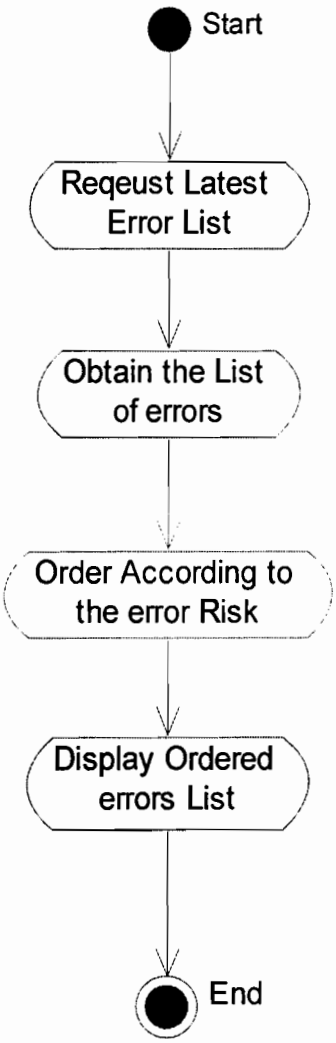


Figure 4. 7: View Latest Errors Activity Diagram

4.3.2 View Network Error Statistics

Figure 4.8 shows activity diagrams describing the behavioral overall flow of control of the system when requesting network errors statistical view.

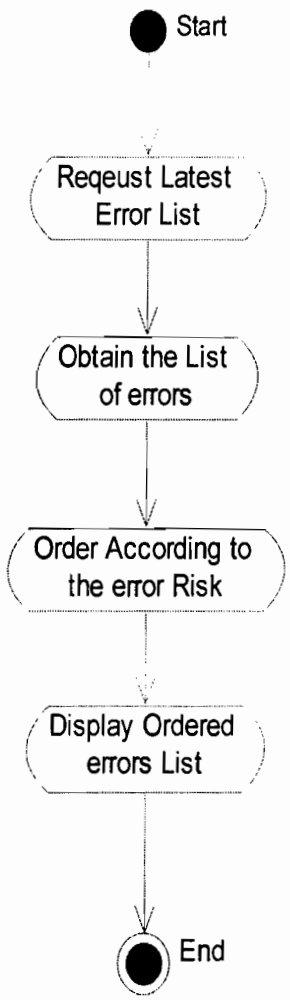


Figure 4. 8: View Network Error Statistics Activity Diagram

4.3.3 View Host Error Statistics

Figure 4.9 shows activity diagrams describing the behavioral overall flow of control of the system when requesting a specific host errors statistical view.

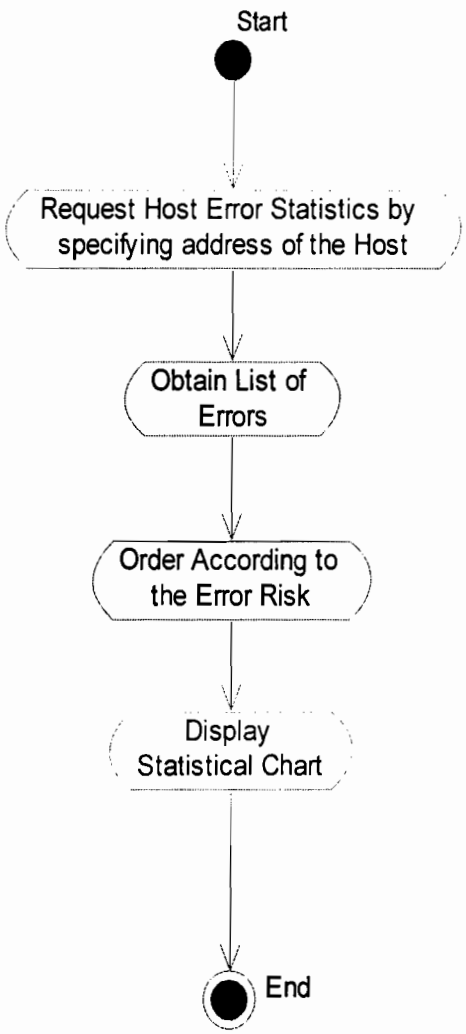


Figure 4. 9: View Host Error Statistics Activity Diagram

4.3.4 Organize Collected Data

Figure 4.10 shows activity diagrams describing the behavioral overall flow of control of the system when requesting organization of transferred reports and the methods of collecting information for the purpose of detecting errors.

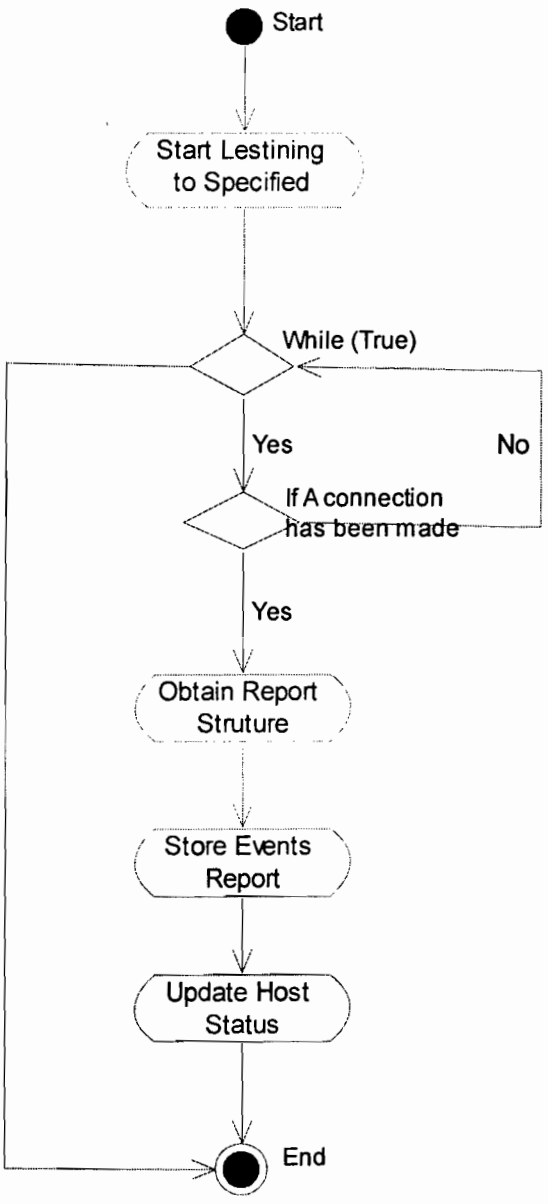


Figure 4. 10: Organize Collected Data Activity Diagram

4.3.5 Collect & Identify

Figure 4.11 shows activity diagrams describing the behavioral overall flow of control of the system when requesting identification of collected operating system metrics and the methods of collecting information for the purpose of detecting errors.

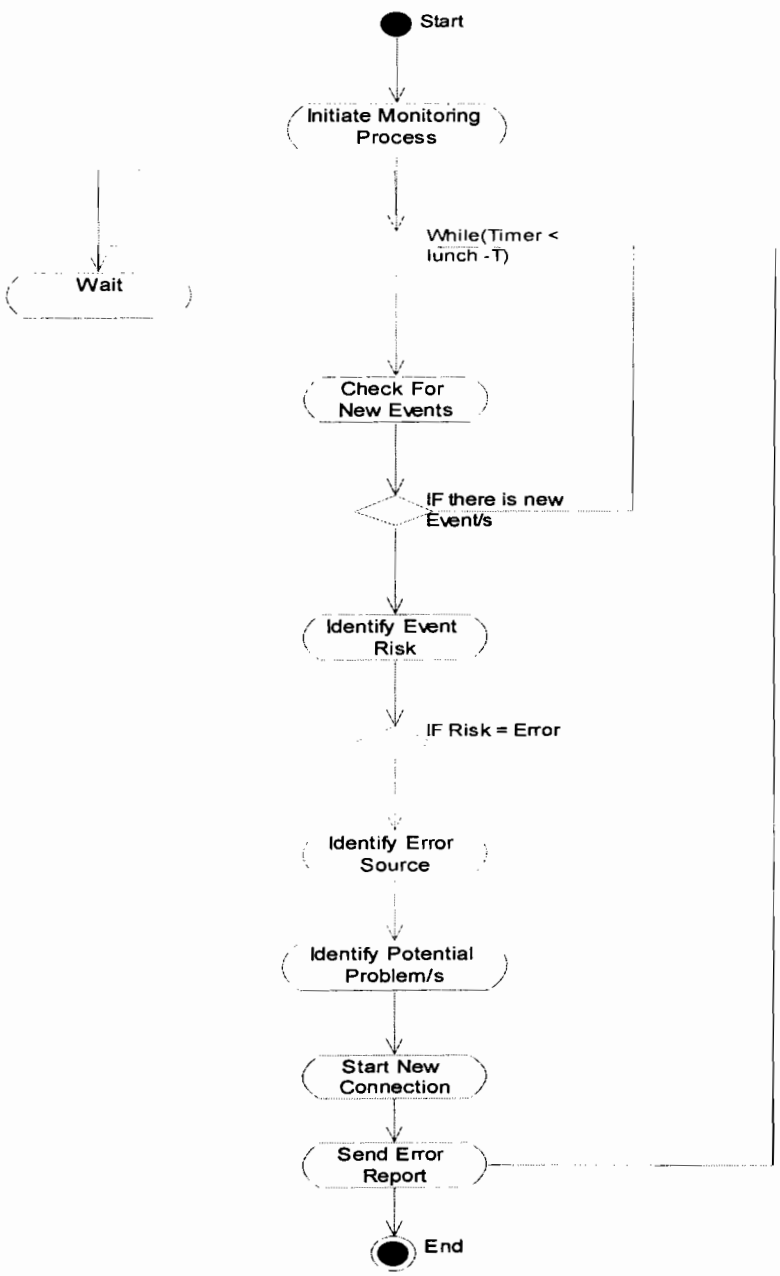


Figure 4. 11: Collect & Identify Events Activity Diagram

4.4 Sequence Diagram

4.4.1 View Latest Errors

Figure 4.12 show the sequence diagram that illustrates the specification of runtime interaction in a graphical manner of the view latest errors procedure.

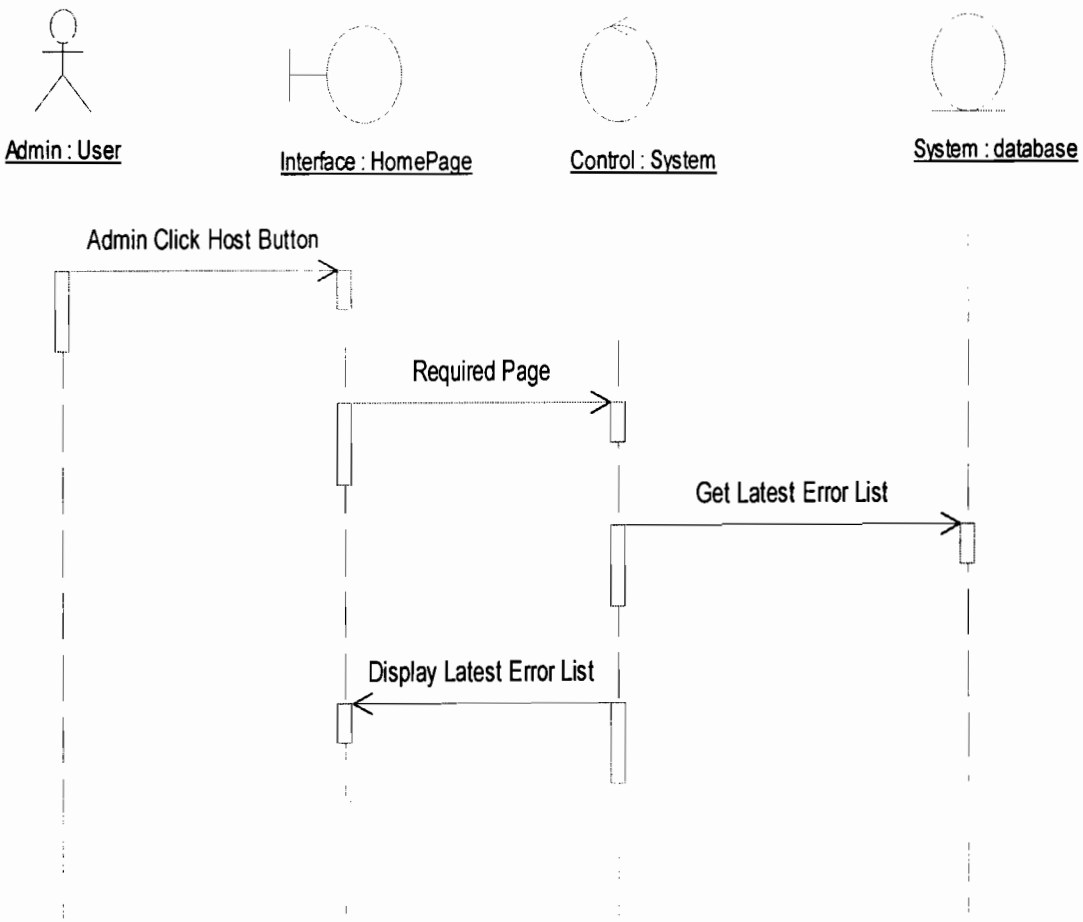


Figure 4. 12: View Latest Errors Sequence Diagram

4.4.2 View Network Errors

Figure 4.13 show the sequence diagram that illustrates the specification of runtime interaction in a graphical manner of the view network errors procedure.

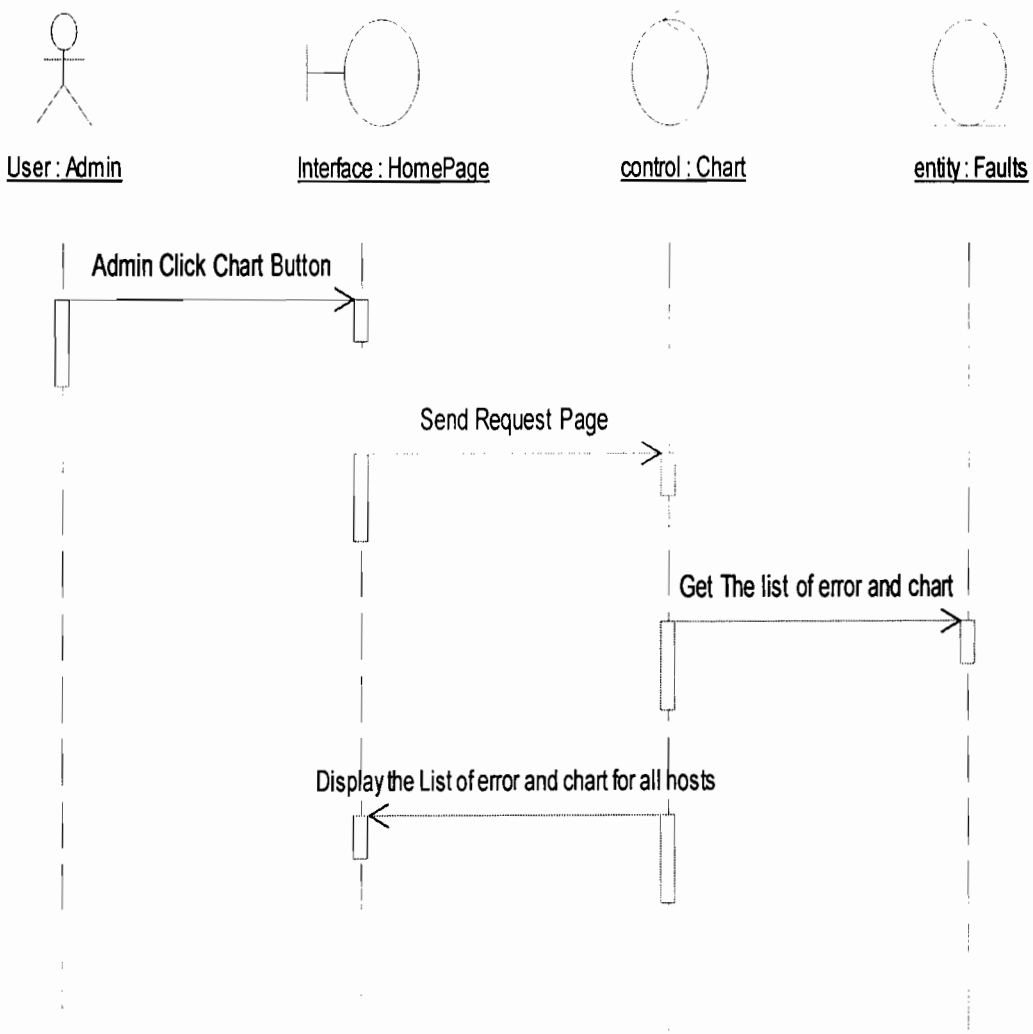


Figure 4. 13: View Network Errors Sequence Diagram

4.4.3 View Host Errors

Figure 4.14 show the sequence diagram that illustrates the specification of runtime interaction in a graphical manner of the view host errors procedure.

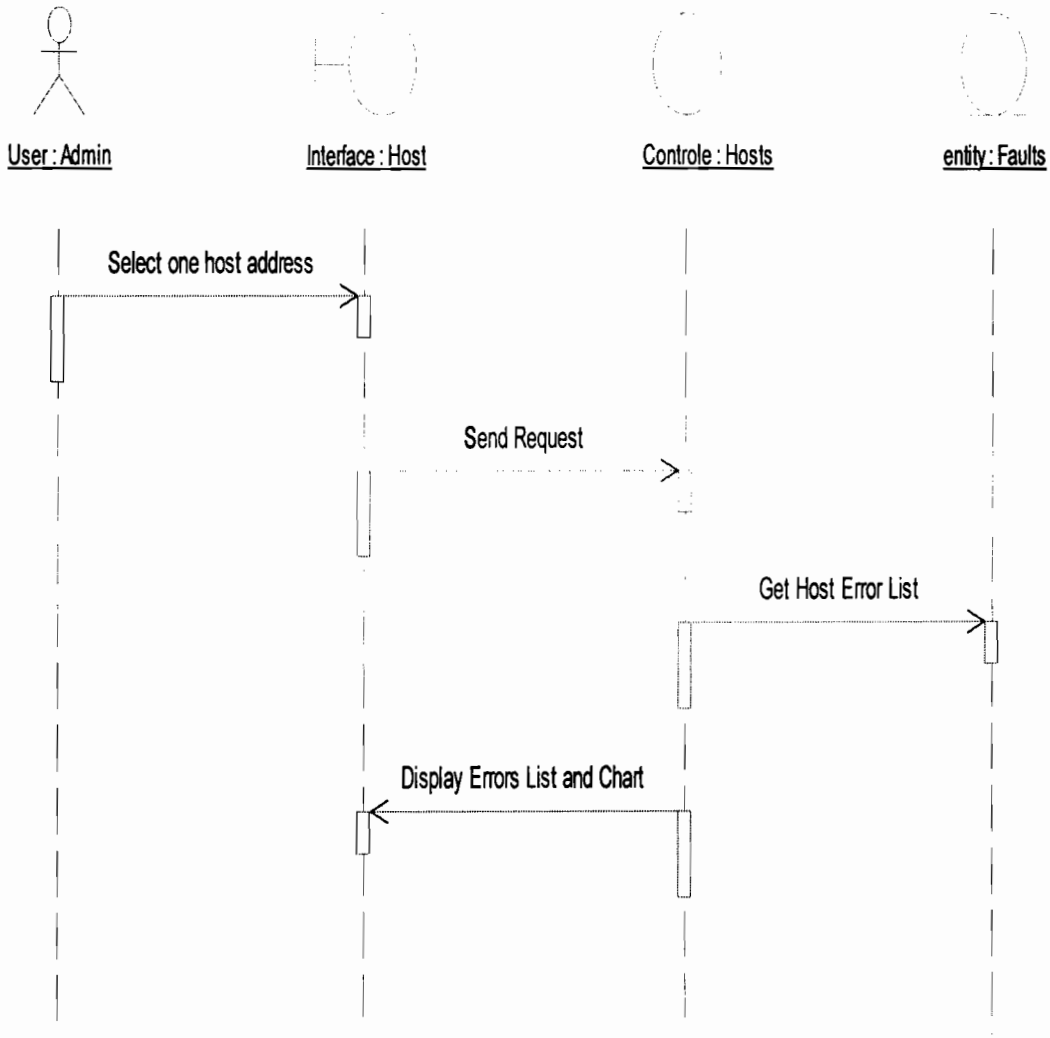


Figure 4. 14: View Host Errors Sequence Diagram

4.4.4 Collect System Metrics and Identify Events

Figure 4.15 show the sequence diagram that illustrates the specification of runtime interaction in a graphical manner of the reports identification and organization procedure.

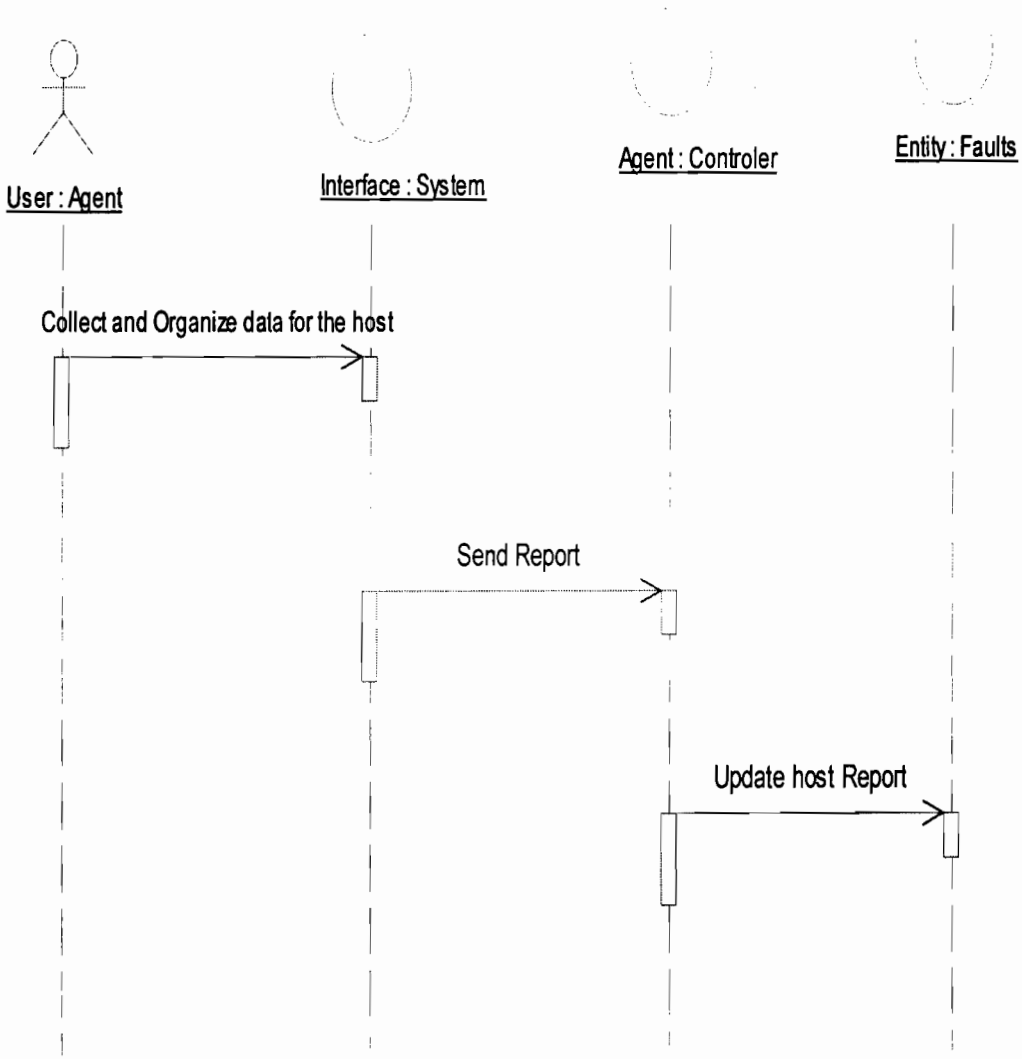


Figure 4. 15: Check Results Sequence Diagram

4.4.5 Collect and Identify Events

Figure 4.16 show the sequence diagram that illustrates the specification of runtime interaction in a graphical manner of system metrics collection and identification of faults procedure.

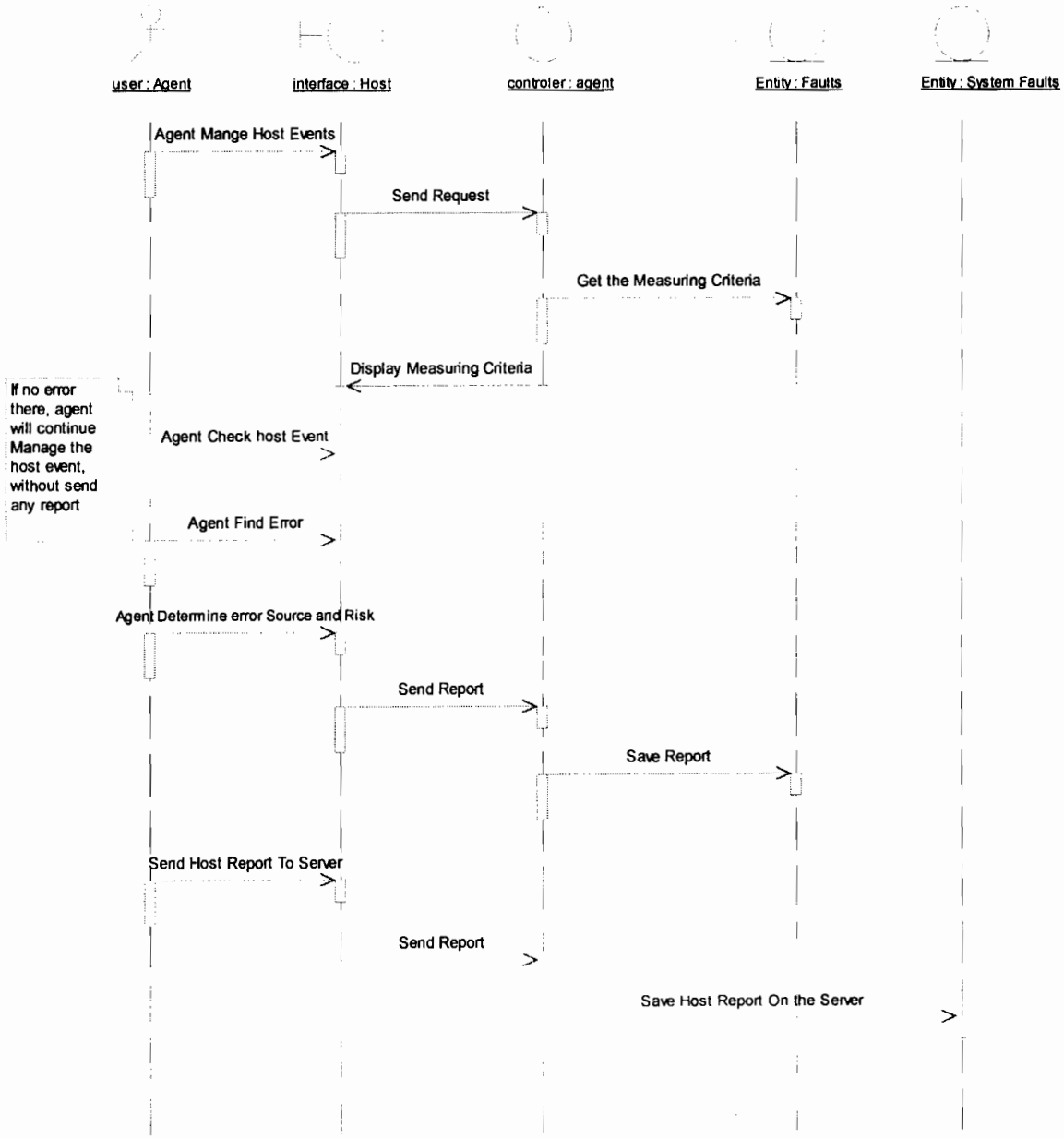


Figure 4. 16: Collect & Identify Events Sequence Diagram

4.5 Collaboration Diagram

4.5.1 View Latest Errors

Figure 4.17 illustrates the relationship and interaction between the systems objects when performing a view latest errors procedure.

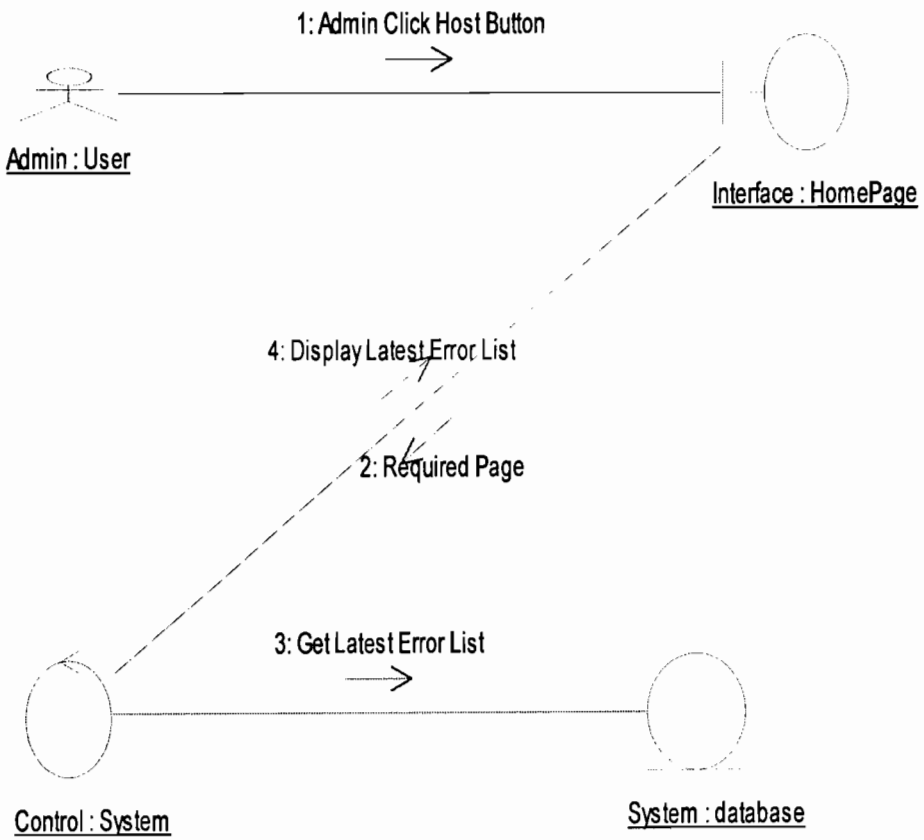


Figure 4. 17: View Latest Errors Collaboration Diagram

4.5.2 View Network Errors

Figure 4.18 illustrates the relationship and interaction between the systems objects when performing a view network errors procedure.

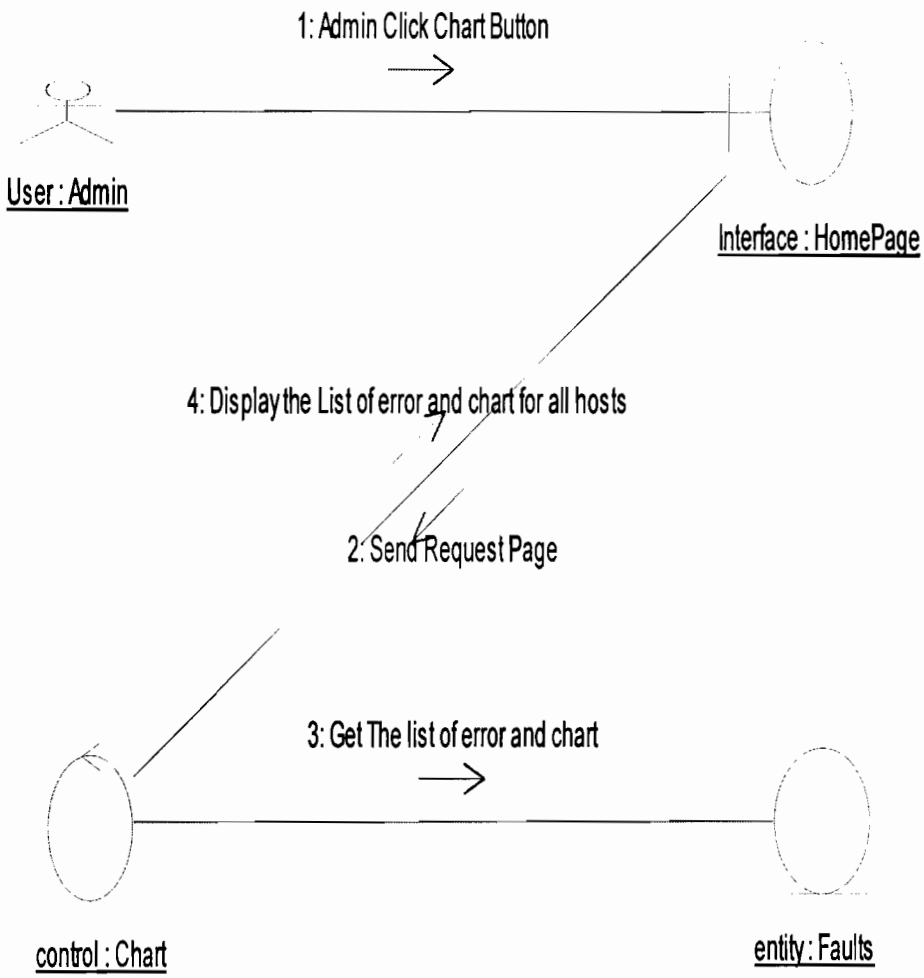


Figure 4. 18: View Network Errors

4.5.3 View Host Errors

Figure 4.19 illustrates the relationship and interaction between the systems objects when performing a view a specific host errors procedure.

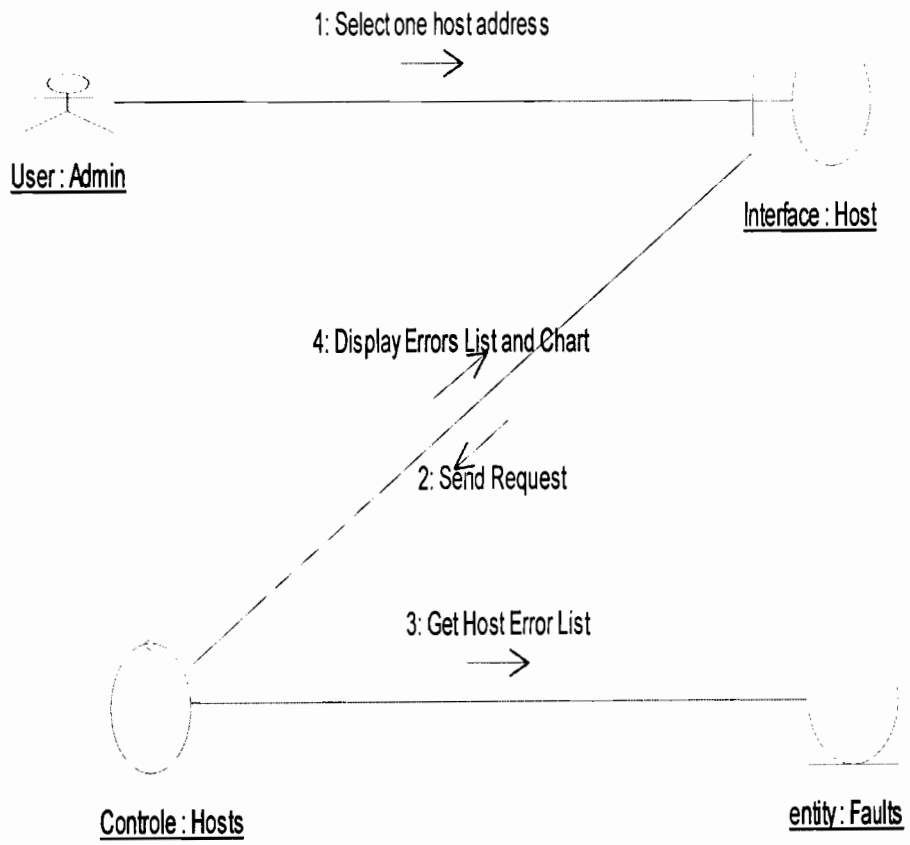


Figure 4. 19: View Host Errors Collaboration Diagram

4.5.4 Organize Reports

Figure 4.20 illustrates the relationship and interaction between the systems objects when performing a organization and identification of transferred reports procedure.

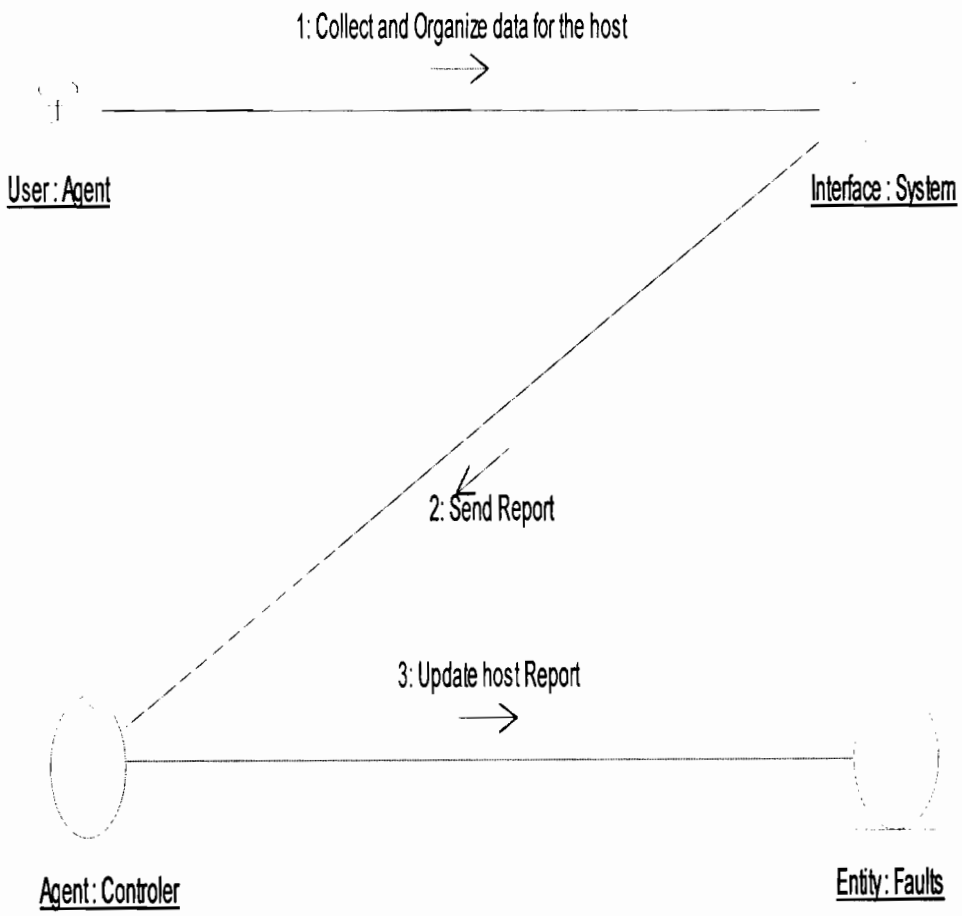


Figure 4. 20: Organize Reports Collaboration Diagram

4.5.5 Collect and Identify Events

Figure 4.21 illustrates the relationship and interaction between the systems objects when performing a collection of system metrics and identification of fault events procedure.

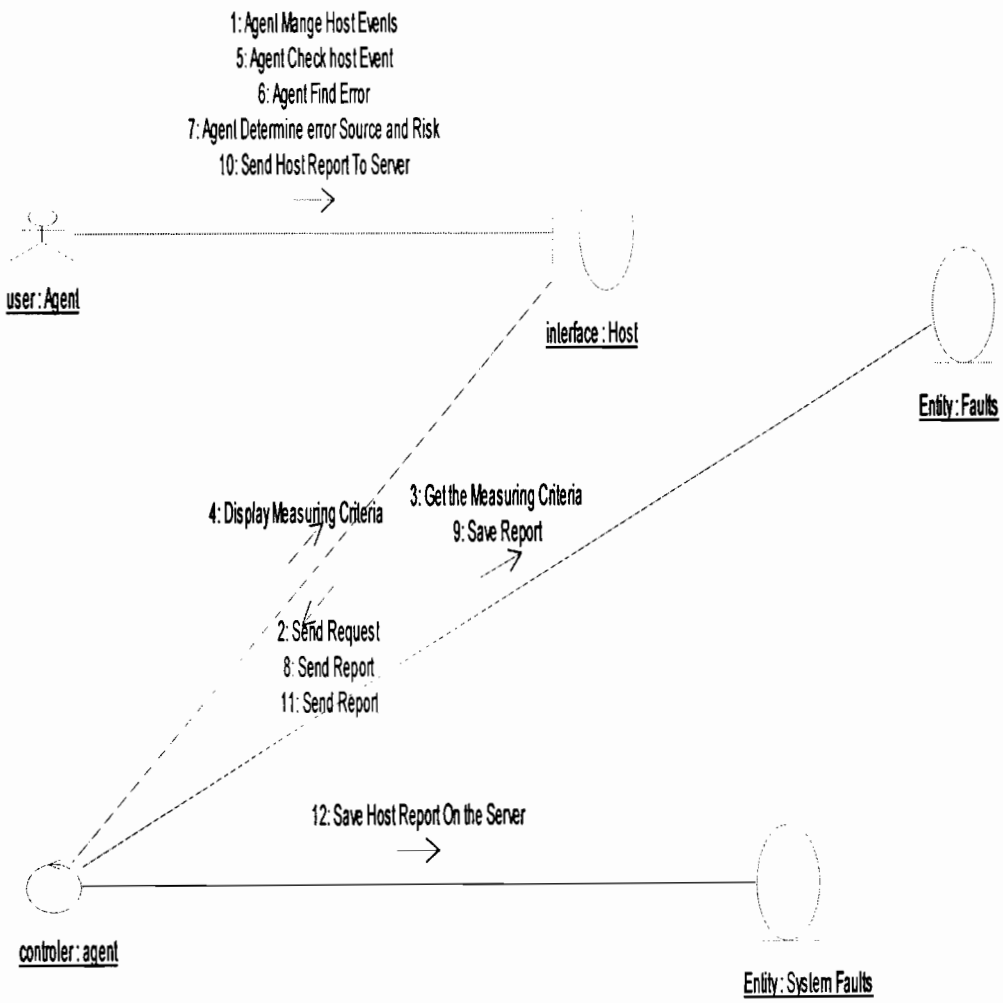


Figure 4. 21: Collect & Identify Events Collaboration Diagram

4.6 Class Diagrams

Figure 4.22 shows the existing packages in the system which are divided into two packages server and agent package.

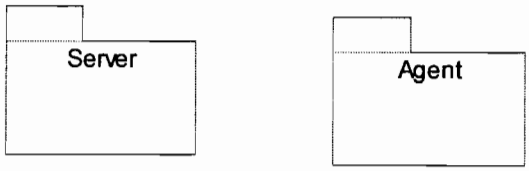


Figure 4. 22: Packages Diagram

Figure 4.23 shows the inter-relationships, and the operations and attributes of the classes existing in the agent package.

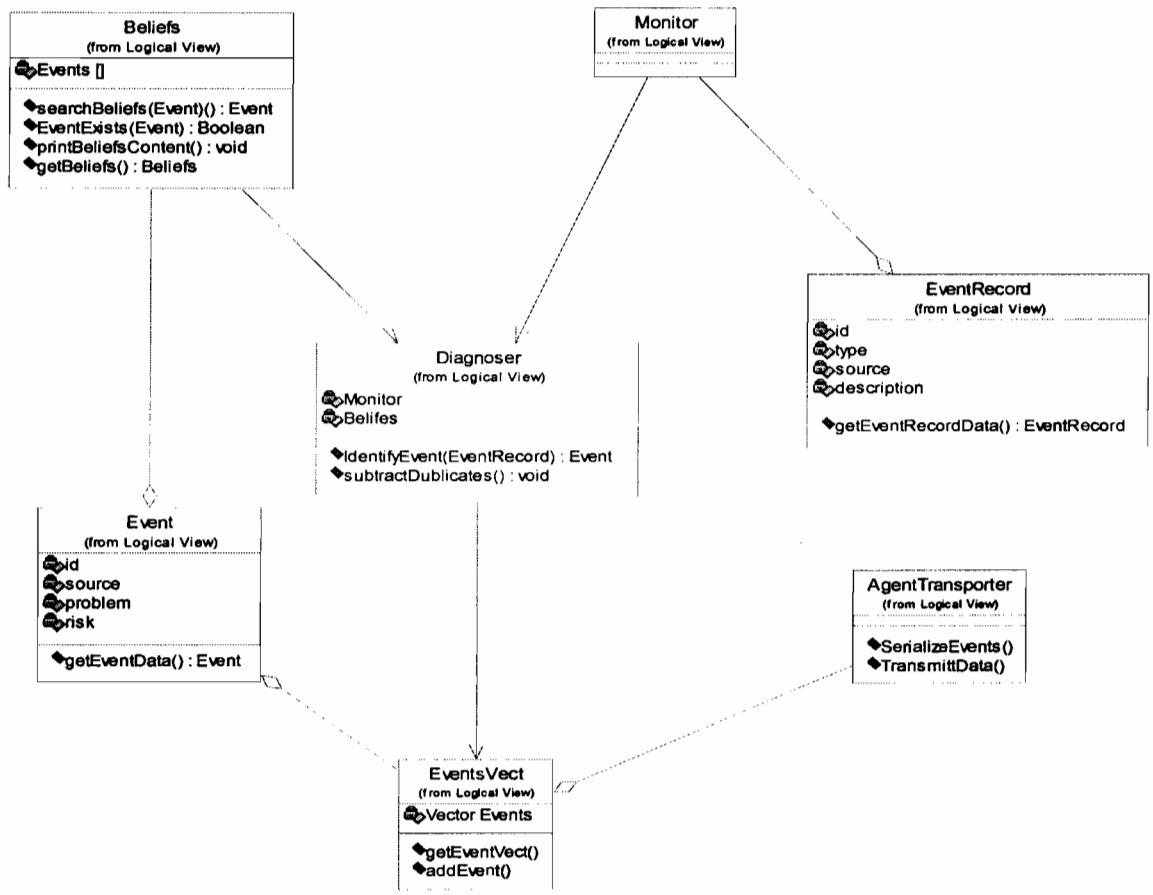


Figure 4. 23: Agent Package Class Diagram

Figure 4.24 shows the inter-relationships, and the operations and attributes of the classes existing in the server package.

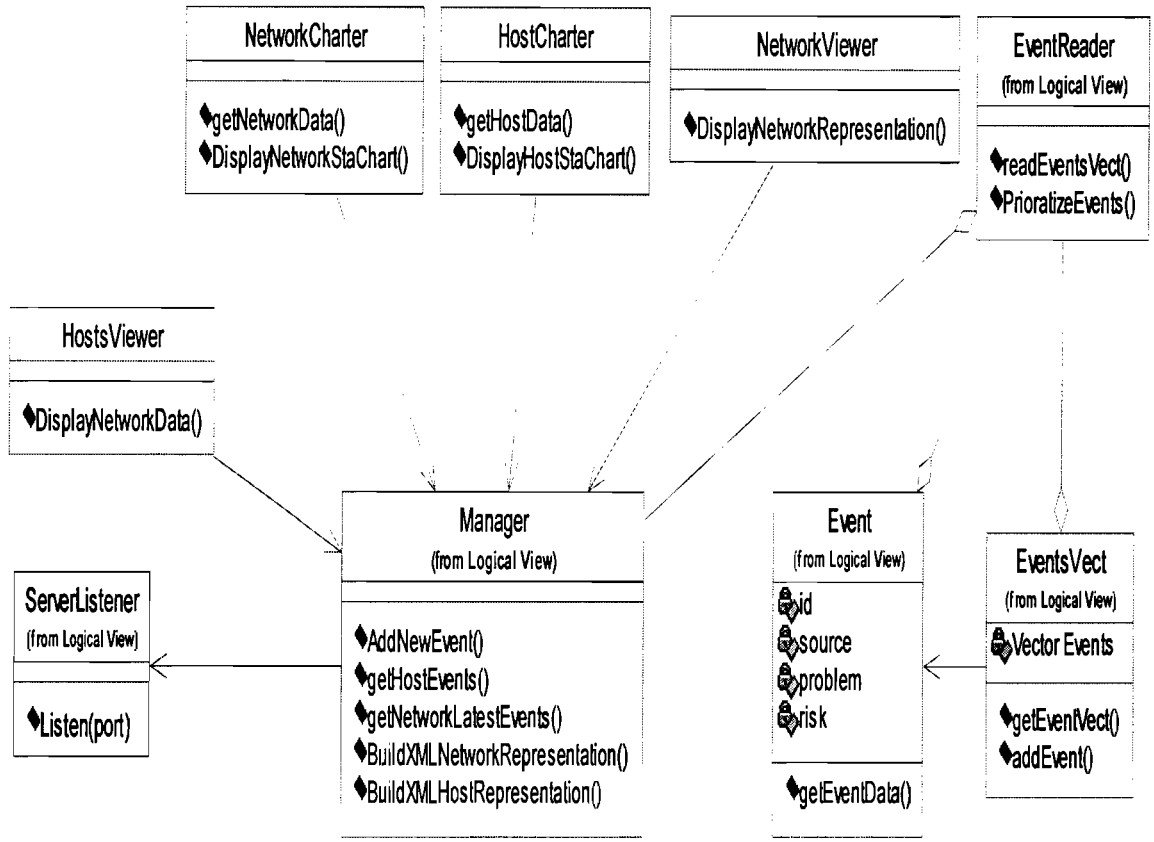


Figure 4. 24: Server Package Class Diagram

4.7 Summary

In this chapter an illustration of the system components , its relationships , and its interaction with each other has been presented.UML diagrams has specified the all of the activity diagrams, sequence diagrams, collaboration diagrams and class diagrams along with the specification of the system use cases.

CHAPTER FIVE

SYSTEM EVALUATION & RESULTS

In this chapter an explanation of the nature of the experiment conducted to evaluate the implemented system, describing the setup process of the experiment and discussing the obtained results.

5.1 Experimental Setup and Data

In order to test the system, a network sample consist of 9 PCs one of them was designated to be the server computer which hosts the web interface was prepared, the system database and the server agent. Agent were distributed over the network and were given "10.8.3.133" as the reporting address destination to transmit reports to it, which is the IP address of the server designated computer.

The sample network computers were cleaned from any symptoms of error existence, to insure correctness in the calculation of the experiment results. Then we created a list of faults to be injected over the network samples randomly, in order to cover different scenarios and situations. Table 5.1 shows the list of injected error.

Table 5. 1 : List of Selected faults used in the Testing process

Fault	Description
W32Time	Time provider NtpClient were not able to perform time synchronization due to sources connectivity problems
Browser	Failed to capture network representation (Microsoft Network Browsing services)
User environment	Due to faulty registration the operating system were not able to query DllName registry entry (False Registry)
Hanging Application	Faulty program execution or compatibility issues
Avira AntiVir	Avira Antivirus failed to update.

In this experiment, five different types of faults explained in Table 5.1 were injected. The selection of these faults was made to simulate the errors and faults on both application and system levels. W32Time error is time synchronization related failure such as host unable to connect to any of the reliable time synch sources. User environment errors are about Internet browser or other network browser services errors (Internet Explorer) related failure such as failures during update or installation operations which may cause application failures. Hanging Application are introduces to report faults for corrupted application which hangs during execution too often, this may happen due to several reasons such as software incompatibility with the operating system, Avira AntiVir errors are related to the antivirus used in the UUM network, it reports faults generated from the

antivirus itself such as failure of update the viruses signature database. Browser errors are related to the Microsoft Network Browsing services, such failures can produce an inconsistent network representation to the faulting host which may cause connectivity problems over the network. We inject the faults and observe the effect of injected faults.

5.2 Trustworthiness and Validation for Fault Types

In this experiment, we have five different fault scenarios explained in table 1 and section 5.1. We categorize and inject faults by building two different scenarios .first scenario is by injecting faults on every PC separately in random numbers. The second scenario includes all faults explained and they are randomly injected stimulusly. Table 5.2 shows the total number of injected faults during both of the two scenarios.

Table 5. 2 : Number of injected faults on their respective hosts

Computer	Error Injected	Number of Faults
/10.8.3.178	W32Time	2
/10.8.3.216	Browser	1
/10.8.3.218	Userenv	2
/10.8.3.246	Hanging Application	5
/10.8.3.247	Userenv , Hanging Application , W32Time , W32Time2	20
/10.8.3.251	Avira AntiVir	1
/10.8.3.254	Hanging Application	7

For the scenarios explained, we evaluate the system ability to detect abnormal and faulty behavior. The system was able to detect all of the injected faults at the time of occurrences. Figure 5.1 shows number of detected faults on their respective hosts taken from the system statistical report.

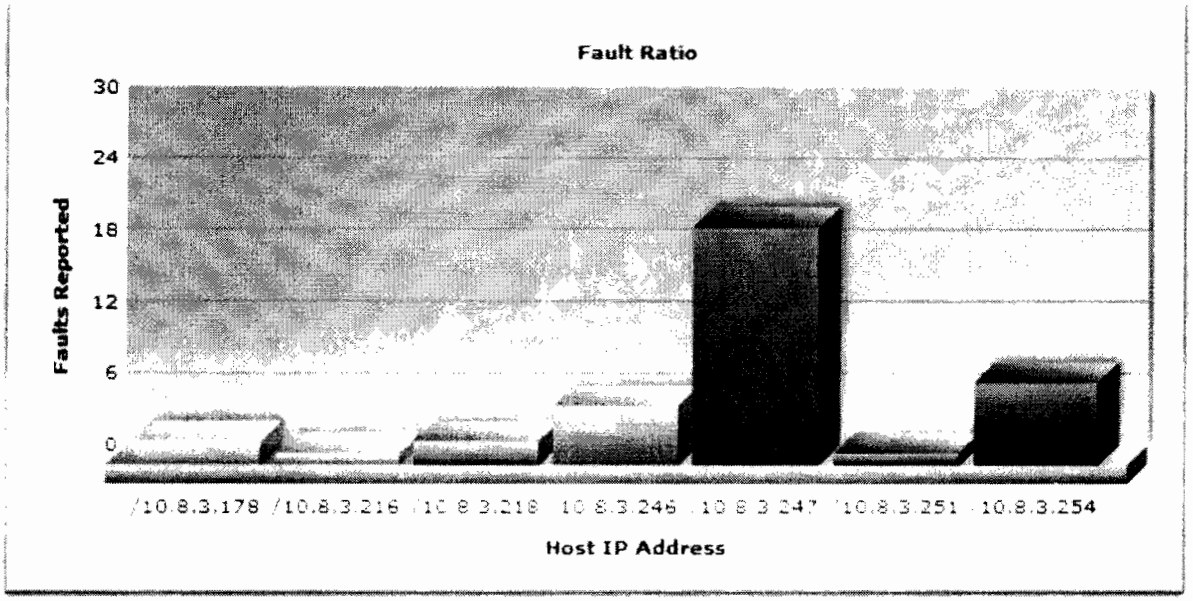


Figure 5. 1 : Number of detected faults on the sample network hosts

The scenario had very good results considering the detection criteria. The system was able to detect all of the injected faults.

5.3 Performance Validation for Testing Data

In order to estimate the system impact on the network bandwidth, we have recorded network usage on the server node because a bottleneck is more likely to happen there. Recording of network usage was made on a one second intervals to insure more accurate results. Results reading consider that there is a current network noise estimated of 2000 bit per second.

As explained before the evaluation was based on two different scenarios. In the first scenario faults were injected separately. Figure 5.2 shows a diagram representing the network activity when transmitting a report from host with IP address 10.8.3.216. The diagram shows that it took approximately one second to transmit the reports with a maximum 10000 b/sec network usage, which is considered to be trivial.

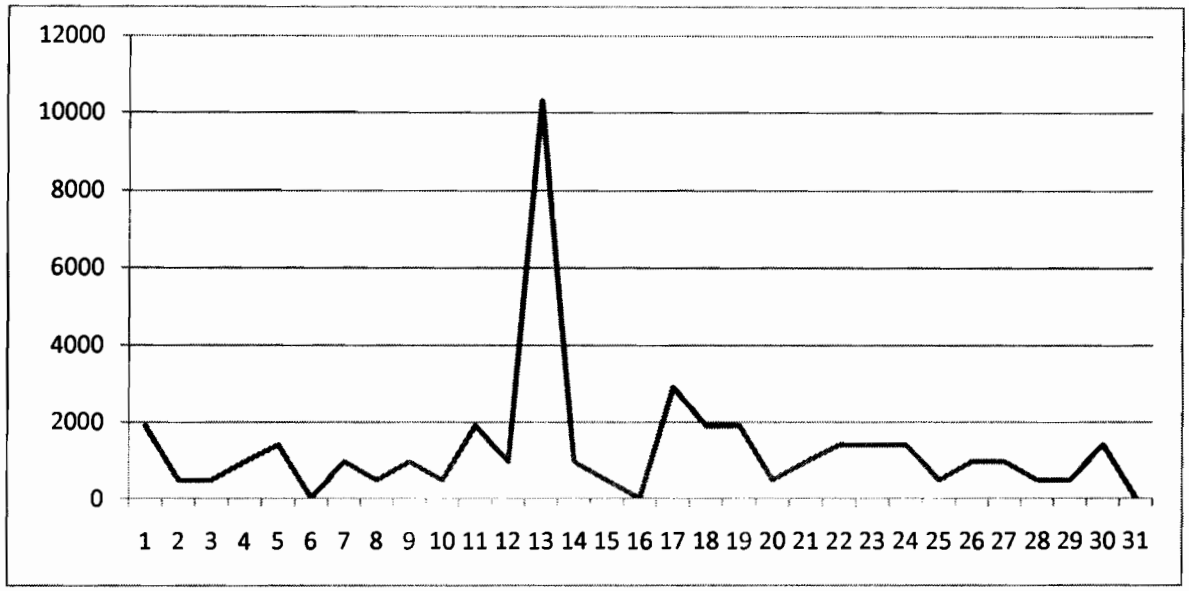


Figure 5. 2 : Network activity when transmitting a single report from one host

Figure 5.3 shows a diagram representing the network activity when transmitting reports from host with IP address 10.8.3.216 which was injected with four faults. The diagram shows that it took approximately one second to transmit the reports with a maximum 25000 b/sec network usage, which is considered to be trivial. Comparing these results with the ones taken from reporting a single report we can see that the difference is rather trivial and there is still no considerable overload on the network bandwidth.

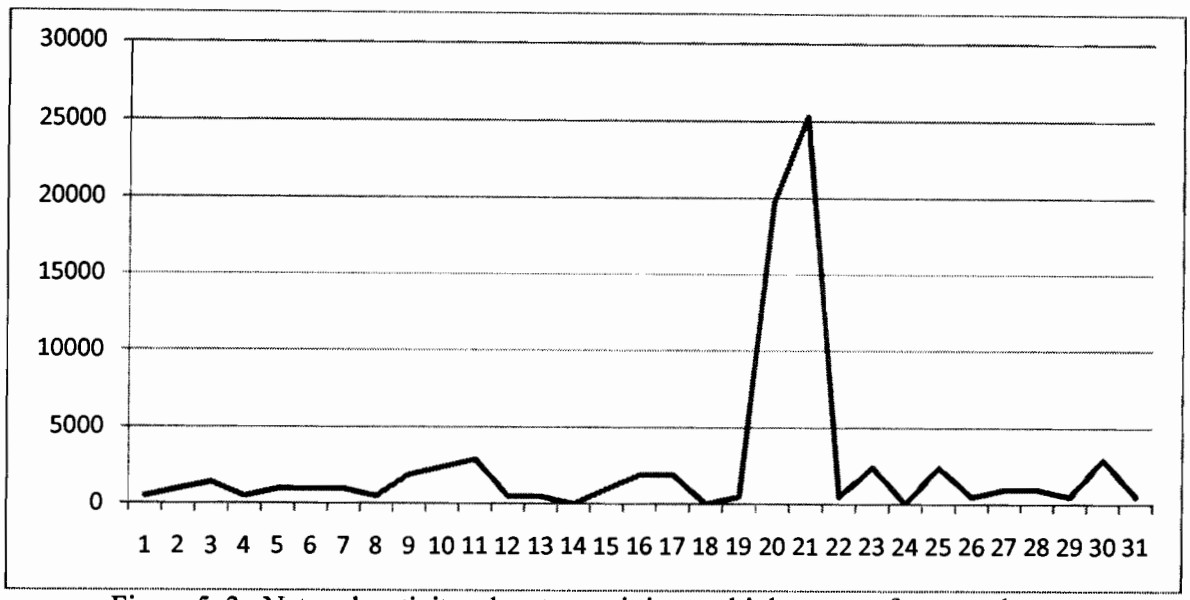


Figure 5. 3 : Network activity when transmitting multiple reports from one host

The previous results have shown that increasing the number of reports transmitted over the network does not have a noticeable effect taking in consideration that LAN network have high speed.

Although the previous scenario have shown the effectiveness of the system when transmitting single or multiple faults reports, but another scenario had to tested simulating the case of bottleneck. In the second scenario we have injected all the faults shown in Table 5.2 in an approximate one second period. Figure 5.4 shows the network usage readings. The diagram shows that it took approximately two seconds to transmit the reports with a maximum 140000 b/sec network usage, forming a 2% network utilization ratio. These results show that the system does not affect the network bandwidth even when a large number of reports are being transmitted and there is no bottleneck situation created.

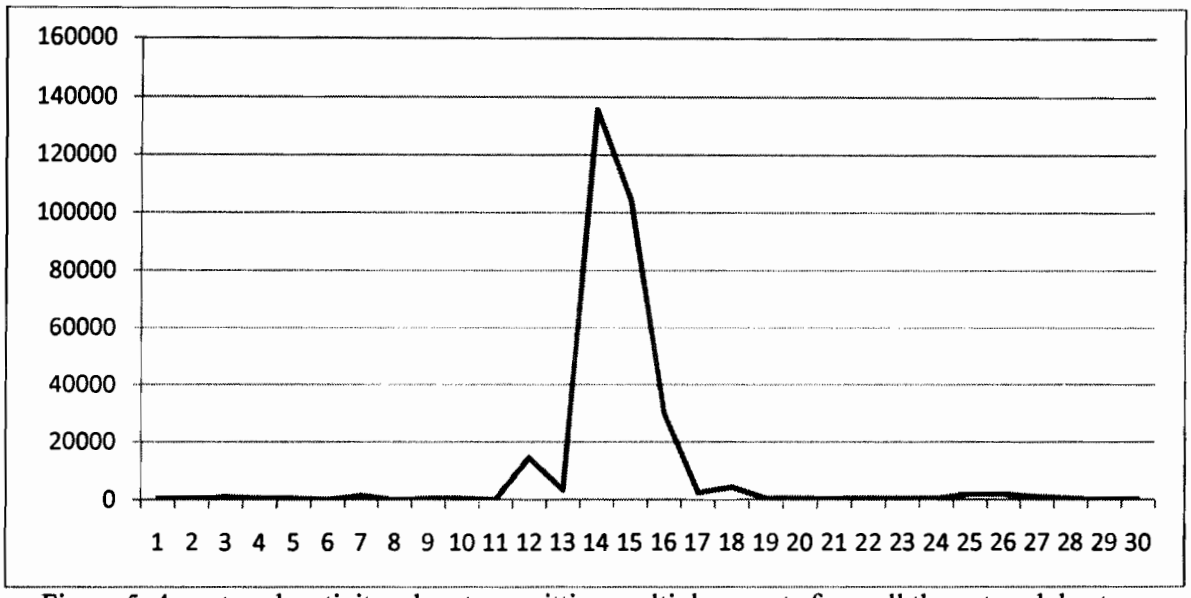


Figure 5. 4 : network activity when transmitting multiple reports from all the network hosts

5.4 Summary

In this chapter a fault injection test has been conducted to evaluate the system ability to detect faults that simulates real life situations. The injected faults were selected on both application and system levels. Testing the system was made to measure two criteria's, trustworthiness of the system and performance. The system has shown a good fault detection ratio when, and has shown low impact on the network bandwidth in different transmission scenarios.

CHAPTER SIX

CONCLUSION & FUTURE WORK

This chapter shows an overall conclusion of the project, the process of conduction and the results of the project. It also shows suggestions for improving the system by adding new characteristics.

6.1 Research Conclusion

In this thesis, an innovative distributed intelligent agent based fault detection system that operates on Windows platform was presented, to capture abnormal and faulty behaviors on both application and system levels, by constantly reading the operating system metrics. The system architecture had two parts. One is the agent which is distributed on the network connected hosts and performs the metrics collection. The other part is the server, which takes responsibility of collecting fault reports transmitted over then network from running agents. We have also developed a web based user graphical interface, to facilitate accessibility to the system. We have evaluated the system trustworthiness to detect faults by injecting real life situation faults into the operating system. During the experiment we recorded the network activity to determine the system impact on the network bandwidth. The experiment results proved the system ability to detect different types of faults injected in different scenarios. Moreover, the results showed that the system a trivial impact on the network bandwidth. Overall, the system showed ability not only to detect faults but also to identify the root cause of the fault.

6.2 Future Work

The IA-based fault detection system developed in this project can be future improved. Some features can be added to the agent abilities such as prediction of faults occurrence. Currently the agent detects and report faults by the time of occurrence. However, if the agent is embedded with knowledge base with the ability to update its beliefs the IA-based system that can analyze the possibility of fault occurrence.

Also improvements in the faults representation in the system graphical user interface by including better statistical views of the network status, which gives a better assessment of the network in general and helps network administrators to isolate contagious faults before spreading.

Lastly, if the agent is embedded with a fault recovery schema in order to perform automated maintenance and recovery, such addition to the system can take such systems into a new level which contains a great value for both academic and commercial fields.

REFERENCES

Anne, C., & Alexis, D. (1998). USING THE CASSIOPEIA METHOD TO DESIGN A ROBOT SOCCER TEAM. *Applied Artificial Intelligence, 12*(2/3), 127.

Baldini, A., Benso, A., Chiusano, S., & Prinetto, P. (2000, 2000). *BOND: An interposition agents based fault injector for Windows NT.* Paper presented at the Proceedings. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2000, Yamanashi, 387-395.

Bianchini, C. d. P. (2003). *Intelligent management of network devices aided by a strategy and a tool.* Paper presented at the Proceedings of the 2003 IFIP/ACM Latin America conference on Towards a Latin American agenda for network research, La Paz, Bolivia,141-151.

Boutaba .R and Xiao. J. (2002). Network Management: State of the Art, *IFIP, 220*, 127 146.Cooperative Network Management Architecture. *IEEE Network, 10 April, 1-3.*

Buschmann, F. (1998). *pattern-oriented software architecture a system of patterns.* Chichester: Wiley.

Cisco Systems, I. (2004). *Internetworking technologies handbook.* Indianapolis, IN: Cisco Press.

Dong-Liang, L., Sheng-Yuan, Y., & Yi-Jen, C. (2009, 3-5 Dec. 2009). *Developing an active mode of network management system with intelligent multi-agent techniques.* Paper presented at the Joint Conferences on Pervasive Computing (JCPC) 2009, 77- 82, Tamsui, Taipei.

Franklin, S., & Graesser, A. (1997). *Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents*. Paper presented at the Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, Verlag, 21-35.

Gavalas, D., Greenwood, D., Ghanbari, M., & O'Mahony, M. (2000). Advanced network monitoring applications based on mobile/intelligent agent technology. *Computer communications., 23*(8), 720.

Gavalas, D., Tsekouras, G. E., & Anagnostopoulos, C. (2009). A mobile agent platform for distributed network and systems management. *J. Syst. Softw., 82*(2), 355-371.

Ibrahim, M. A. M. (2006). *Distributed Network Management with Secured Mobile Agent Support*. Paper presented at the Proceedings of the 2006 International Conference on Hybrid Information Technology,Cheju Island, 244-251.

Kendall, E. A., Krishna, P. V. M., Suresh, C. B., & Pathak, C. V. (2000). An application framework for intelligent and mobile agents. *ACM Comput. Surv., 32*(1es), 20.

Kim, B. U., Al-Nashif, Y., Fayssal, S., Hariri, S., & Yousif, M. (2008). *Anomaly-based fault detection in pervasive computing system*. Paper presented at the Proceedings of the 5th international conference on Pervasive services, Sorrento, Italy, 147-156.

Kuechler, W., Vaishnavi, V., & Kuechler, W. L. (2007). *Design [Science] Research in IS: A Work in Progress*. Paper presented at the 2nd International Conference on Design Science Research in Information Systems and Technology, USA, California, 234-239.

Liotta, A., Pavlou, G., & Knight, G. (2002). Exploiting Agent Mobility for Large-Scale Network Monitoring. *IEEE NETWORK, 16*, 7-15.

Pugazendi, R., & Duraiswamy, K. (2009, 27-28 Oct. 2009). *Mobile Agents - A Solution for Network Monitoring*. Paper presented at the ARTCom '09. International Conference on Advances in Recent Technologies in Communication and Computing, 2009, Kottayam, Kerala, 579-584.

Ray, J., Hoe, J. C., & Falsafi, B. (2001). *Dual use of superscalar datapath for transient-fault detection and recovery*. Paper presented at the Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture, Austin, Texas, 214-224.

Reinhardt, S. K., & Mukherjee, S. S. (2000). Transient fault detection via simultaneous multithreading. *SIGARCH Comput. Archit. News, 28*(2), 25-36.

Sakai, M., Matsuba, H., & Ishikawa, Y. (2007, 17-19 Dec. 2007). *Fault Detection System Activated by Failure Information*. Paper presented at the 13th Pacific Rim International Symposium on Dependable Computing, 2007. PRDC 2007., Melbourne, Qld, 19-26.

Satoh, I. (2002, 2002). *A framework for building reusable mobile agents for network management*. Paper presented at the Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP, Tokyo, Japan, 51-64.

Staff, M. C. (1992). *The Basics Book of OSI and Network Management*: Addison-Wesley Longman Publishing Co., Inc.

Stallings, W. (1999). *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Reading, Mass.: Addison-Wesley.

Utton, P., & Scharf, E. (2004). A fault diagnosis system for the connected home. *Communications Magazine, IEEE, 42*(11), 128-134.

Vaishnavi, V. and Kuechler, W. (2004). "Design Research in Information Systems" January 20, 2004, last updated January 18, 2006. URL: http://www.isworld.org/Researchdesign/drisISworld.htm

Wooldridge, M. (2002). *An introduction to multiagent systems*. Chichester: Wiley.

Wu, F., Zhao, Z., & Ye, X. (2008, 20-22 Dec. 2008). *A New Dynamic Network Monitoring Based on IA.* Paper presented at the International Symposium on Computer Science and Computational Technology, 2008. ISCSCT '08,, Shanghai, 637-640.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing Multiagent Systems: The Gaia Methodology. *ACM transactions on software engineering and methodology., 12*(3), 317.

Zhang, S., Cohen, I., Symons, J., & Fox, A. (2005). *Ensembles of Models for Automated Diagnosis of System Performance Problems*. Paper presented at the Proceedings of the 2005 International Conference on Dependable Systems and Networks, CA, USA, 644-653.