

Dynamic Firewall Configuration Using Mobile Agents

by

Muhamad Hazwan b Hamzah

3010

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Business Information Systems (Hons)
(Information Systems)

JULY 2005

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

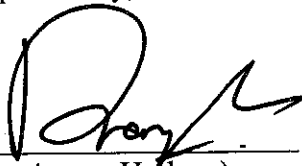
Dynamic Firewall Configuration Using Mobile Agents

by

Muhamad Hazwan b Hamzah

A project dissertation submitted to the
Information Systems Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF BUSINESS INFORMATION SYSTEMS (Hons)
(INFORMATION SYSTEMS)

Approved by,



(Mr. Anang Hudaya)

t

UNIVERSITI TEKNOLOGI PETRONAS

GA

TRONOH, PERAK

76.76

July 2005

158

1452

1998

- 1) Mobile agents (computer software)
- 2) Computer security

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

MUHAMAD HAZWAN B HAMZAH (3010)

ABSTRACT

The nearly omnipresence of the Internet and the steady increase of wireless computing and mobile devices require highly dynamic adaptable distributed system architectures. Building such architectures needs a combination of key concepts from component technology and distributed systems. Mobile agents provide this combination. We use mobile agents as the building blocks of a component-based system for remote supervision and control of both hard- and software in a distributed environment. In this paper we concentrate on the configuration of individual components and component relationships in our system. We identify requirements for remote configuration of agent-based component systems and discuss architectural and user interface related issues of our approaches. We use a code-on-demand approach for supporting elaborate user interfaces. We use a generative approach based on enhanced meta-information for reducing development effort. The presented approaches are applicable for remote configuration of component-based systems in general and consider additional requirements imposed through the use of mobile agent technology.

ACKNOWLEDGEMENT

Completing a project on this title is a very intensive process and it takes the support and dedication of many people to make it possible. Above all, I would like to express my gratitude to the Lord Almighty for giving me the strength, wisdom and patient to complete this project on time.

My deepest gratitude goes to Mr. Anang Hudaya b Muhd Amin, Final Year Project (IS/IT) Supervisor for all the guidance, advices, incentives and of course his patience which have given me the motivation and strength towards finishing this project.

I must give thanks to my beloved parents whom had given me the inspiration to carry the project through the end. Not only this FYP project but their encouragement and support for me to come as far as I am now.

Lastly, this project would not be completed without acknowledging the contributions of my friends, for being helpful and committed throughout the project development. I could not mention anyone's name without slighting dozen others, so cooperation that you give is so valuable for me in order completing this project.

TABLE OF CONTENTS

CERTIFICATION	
ABSTRACT	1
ACKNOWLEDGEMENT	2
CHAPTER 1:	INTRODUCTION	3
	1.1	Background of Study	3
	1.2	Problem Statement	5
	1.3	Objectives and Scope of Study	6
CHAPTER 2:	THEORY	8
	2.1	Introduction	8
CHAPTER 3:	METHODOLOGY	13
	3.1	Procedure Identification	13
	3.2	Tool Required.	15
CHAPTER 4:	RESULTS AND DISCUSSION	16
	4.1	Configuration Requirement	16
	4.2	Configuration Of Individual Agents	22
	4.3	A Generative Approach for Configuration Uis	25
	4.4	Limitation of the Project	28
CHAPTER 5:	CONCLUSION AND RECOMMENDATION	29
	5.1	Conclusion	29
	5.2	Recommendations	29
REFERENCES	30
APPENDICES	31

1. INTRODUCTION

1.1 BACKGROUND OF STUDIES

Distributed software architectures are currently increasingly influenced by two major technological movements—the Internet and pervasive computing (including wireless and mobile systems). In the last years, the Internet has mainly been used as the technological basis for creating the Web, a global hypertext and hypermedia network, enriched with simple interactive (HTML-based) services, like search engines, electronic shops, and electronic auctions. Currently, the Internet and its protocols are more and more becoming the infrastructural backbone for arbitrary services and systems. Nearly every distributed application is required to work in an Internet context or is based on standardized Internet protocols.

The Internet is also changing application deployment and maintenance. Internet-based deployment comprises not only the transfer and installation of software, but all activities from installation until deinstallation and removal of a software system at a consumer's site [1]. This includes tasks like remote activation, deactivation, configuration, reconfiguration, addition, removal, and update of software. All these activities are not only performed for whole applications but also for individual components, and sometimes even at run-time. The result is the need for highly flexible and adaptable software architectures as well as the need for remote configuration and management tools.

The second major technological movement is wireless and mobile computing, which makes further demands on distributed software architectures. Examples are adaptation to different environmental conditions, dynamic service discovery, scalability, robustness and security [2]. Remote configuration tasks may be performed using a whole range of potentially different end-user devices with dedicated user interfaces.

Many of the challenges stated above are addressed by component technology [3][4]. Component models [5] provide standards for component customization, communication, evolution and composition. Components are the basis for adaptable software

architectures. Mobile agent technology has similar characteristics as component technology [6]. Nearly all distinguishing features of component systems that are standardized in general component models are equally important in mobile agent systems. However, mobile agent technology additionally emphasizes support for distribution, heterogeneity, adaptation to different environments, code mobility, and spontaneous computing. These features are especially important for the application domains outlined above. In fact, mobile agent platforms may be viewed as powerful and flexible component environments.

We use mobile agent technology as the basis for a flexible component system for remote diagnosis and monitoring of hard- and software resources in heterogeneous distributed environments. Currently the main usage areas are process automation systems though the system is not limited to this domain. A main characteristic of our system is its highly dynamic structure. Diagnosis and monitoring components may move within the network to their intended place of action, which is the hard- or software resource to be monitored or analyzed. This requires support for code mobility. Other features that are needed and supported by our system are dynamic service discovery, dynamic services, native-code management, multi-protocol remote access of various types of components, robustness, and security.

A main feature of our system, which is also the topic of this paper, is remote configuration and management of monitoring and diagnosis components over Internet connections. Since the components of our system are mobile agents, we will use the terms component and agent interchangeably in the remainder of this paper. We have experimented with a number of approaches for remote configuration of individual agents and of system properties like agent relationships. While most of the explored techniques apply to remote configuration of components in general, some are specific to the characteristics of mobile agent systems.

1.2 PROBLEM STATEMENT

1.2.1 PROBLEM IDENTIFICATION

1. *Difficulties to change or setup firewall when user does not around (in front of computer).* Usually, if users want setup a firewall they need to be in front of their computer. This can bring difficulties for users who travel a lot and do not around (in front of their computer) to change their firewall configuration.
2. *Traditional way does not offer flexibility.* Nowadays, users setup their firewall by using the traditional ways. This technique do not offer a flexibility for users who travels and does not around (in front of their computer) to setup their firewall because they need to be in front of their computer. This also caused the user to lost their times.

1.2.2 SIGNIFICANT OF THE PROJECT

Based on the problems statements, it is very effective to change the way of doing the firewall configurations by replacing the currently used chores with more advance approach. Mobile agent technology is considered as new to Malaysia even most of the developing countries are making use of it and gain advantages upon this.

As mentioned above, the limitation of old technique is mostly focus on range. Therefore by using mobile agents it can offer an efficient way simultaneously can eliminate the problems.

1.3 OBJECTIVE AND SCOPE OF STUDIES

1.3.1 OBJECTIVE

1. *To use mobile agents as the basis for a flexible component system.* The main objective of the project is to make mobile agents as the basis for a flexible component system. With this system, it can help user to setup their firewall anywhere simultaneously eliminate the difficulties they face before.
2. *Replace manual system with the new technology for more effectiveness.* Second objective of the project is to replace manual system with the new technology for more effectiveness. With this new technology, it can help users who travels a lot to do their task with more efficient and help them to save their time because they do not have to be in front of their computer to setup their firewall.

1.3.2 SCOPE OF STUDIES

1. *Develop a mobile agents for configure a firewall.* In this project, mobile agents will be created as a new solution for user who travels a lot to configure their firewall in a wide range (using mobile). The project will be creating by using JAVA MOBILE AGENTS WITH AGLET for the mobile agents and LINUX for the database (host). With this system, it can help to simplicity user and implement new technology in a day life.
2. *To facilitate user that travels to many places.* With this new system, it can help to facilitate travelers/users in current life. Travelers will not find any difficulties to setup their firewall if their using this system because the configuration will be done by mobile agents and it is very easy to carry out with them while they are working.
3. *To replace old system with a new technology for more effectiveness.* This project will replace the old system because it is more effective and efficient to be used. It also can save users/travelers time when they are working outstation and need to setup/configure their firewall.

1.3.3 FEASIBILITY OF THE PROJECT

The product of this project is a Dynamic Firewall Configuration Using Mobile Agents prototype. The prototype applies technical requirement that is obtain from the research. Feasibility of the project is depending on time and tools available. The scope of project seems to be feasible for author to complete on time. However, there are some mobile agents' features needs to be implemented.

2. LITERATURE REVIEW AND THEORY

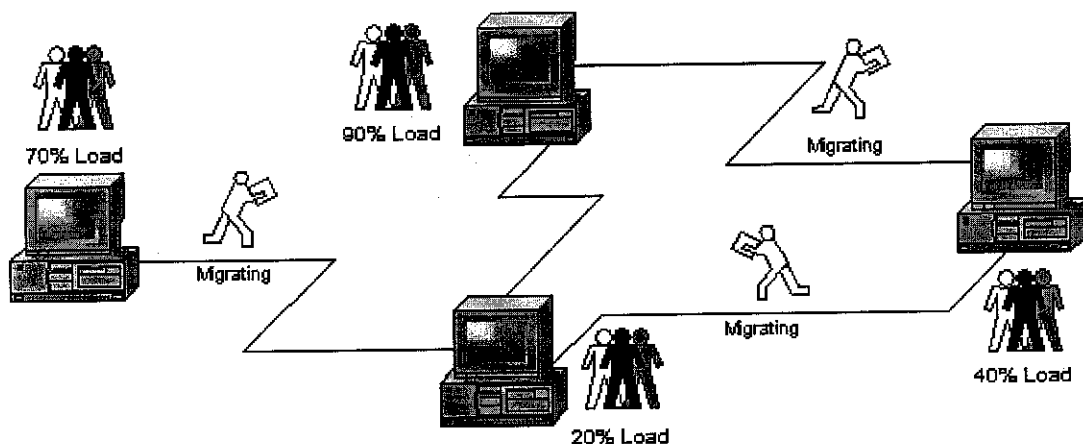
2.1 INTRODUCTION

2.1.1 Entry into the Mobile Agents world

To view the internet as one single computer and make use of its immense potential has been a dream for many. Sharing the computing power and, most important, distributing services is one of the prime goals in network computing.

Whenever someone wants a service that goes beyond the bounds of one single computer (e.g. a game), there is one client, which requests a service, and one server, which fulfills the service (client-server paradigm). In traditional network computing, both server and client are static, i.e. both applicants remain on their respective computer.

In the world of mobile agents, there is no need for that. The client would become an agent who can execute wherever she needs to, performing tasks on behalf of the user; the server may remain static (this is certainly required if the servers task require special hardware or the servers' carrier is not willing to give his code away) or also become a mobile agent, a kind of traveling salesman selling his services at the door of her customers. More than that, in that paradigm static services could become redundant and fault-tolerant in an easier manner, when agents just move on to next host providing the same service if one fails to work.



Mobile Agent Programs roaming the Net

So what are Mobile Agents? Mobile agents are objects (data and code) that can move over a network without prior need to install its code - only a generic Host is needed to execute the agents. This Host usually provides a unified interface to the services available on the computer it is running on, as well as an interface to the other agents currently residing.

This service interface becomes more interesting when looked at in not-commonly-used terms. The internet as-is is a place where only commonly used services like web, mail, ssh, and the like are widely understood - and used. Not only is integrating a new service on server side quite an act - introducing the user to yet another kind of protocol is the main difficulty. With the service interface of a Host, services provided are accessed in a unified manner, thus easing the act of introducing new ones.

For the client, accessing those services becomes easier: Once the user has got used to employing agents, new tasks are explored by the use of new agents, or better sometimes, thru the use of different options with an already well-known agent. Mobile agents can act in a semi-intelligent manner, thus relieving the user from monitoring and operating the tasks.

2.1.2 Understanding Firewalls

These days, firewalls are on the rise as everyone tries to protect their networks to the utmost, blocking and masquerading to the extreme, impeding everything but what is explicitly allowed for the people behind the firewall. The internet is no longer the free, everyone-connects-everyone place that it used to be.

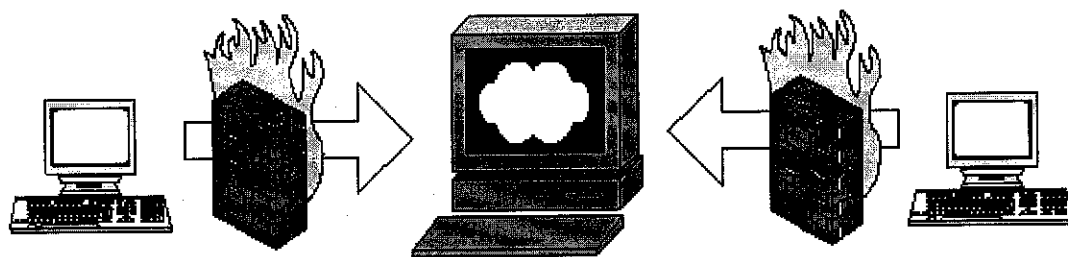
Firewalls accomplish three tasks: Port blocking, masquerading and analyzing.

The first one, port blocking, is not entirely correct to be named as such. Actually, it is the network components of the underlying operating system which open or close network ports.

To understand this issue, suppose someone wants to offer a service on a computer. This computer has one address, e.g. "computer.com", but many services on it, which must somehow be identified. To solve this problem, every service has a permanent port number between 0 and 65535 on this computer. This port number is usually well known (e.g. 80 for http, 25 for telnet, and so on) and must at least be known to the client that wishes to use this service (who would then e.g. use telnet by addressing "computer.com:25").

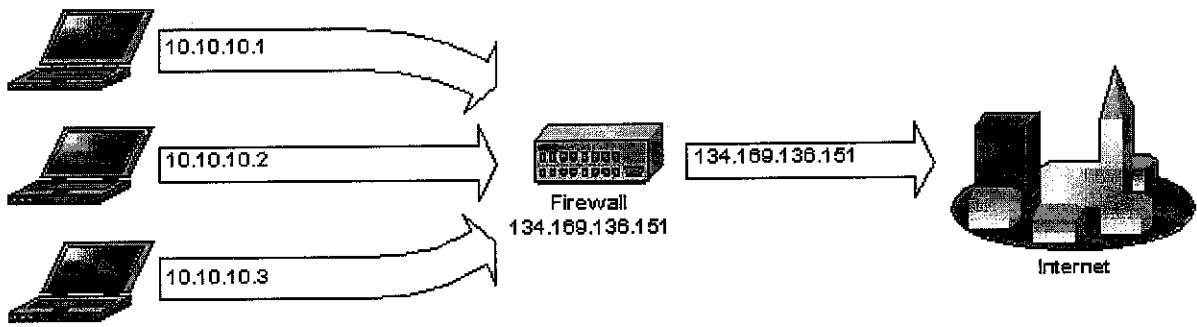
Blocking firewall computers usually block all ports for incoming traffic and most ports for outgoing traffic (in extreme cases everything but 80 so that the people can access the web, but nothing more). Most configurations block all incoming ports but let a wide range of outgoing ports open - this way, no one can attack the services behind the firewall, at the same time giving the people behind the firewall full access to the internet.

The problem arises when two people that are both fire walled by port blocking try to communicate: Neither of them can connect to the other; the only possibility to establish contact is to use a common server that acts as a "relay" or "meeting point" for the participants. Which, if the server ceases to work or begins spying on its users, is a very inconvenient approach?



Meeting at an intermediate Relay Server

The second issue is masquerading. Suppose one company computers are all connected to the internet by way of one firewall computer. When one of these computers opens a connection to the internet, this firewall computer replaces the originators address with its own. This way, all connections seem to come from the firewall computer. This is called masquerading. It is quite obvious that nobody can connect to a computer behind the firewall as even outgoing connections do not reveal the address of the sender.



All Computers wear the Mask of the Firewall in the Internet

Third, analyzing is a process on the firewall computer in which all TCP/IP streams are analyzed for possible attack patterns. While this is a very interesting field of research, it has no effect on this thesis whatsoever.

The major problem that average users face is the inability to be contacted from the outside because they are port blocked or masqueraded. For a mobile agent environment, where agents should be able to transfer freely between any hosts, this situation is fatal. Throughout this thesis, it will be assumed that hosts are troubled by this obstacle.

AUTHOR="Xiaotao Wu and Henning Schulzrinne",

TITLE="Location-based Services in {Internet} Telephony Systems",

BOOKTITLE="IEEE Consumer Communications and Networking Conference",

MONTH=Jan,

YEAR=2005,

LANGUAGE="English",

ABSTRACT="Many applications used in the Internet today benefit from using location information. To better handle location information in Internet telephony applications, we did a comprehensive application-layer analysis of location information and location-based communication services. We first summarize and categorize end-user-oriented location description and location detection approaches. We then summarize and categorize how to use location information to provide communication services and introduce several interesting location based communication services. Based on the

analysis, we have incorporated location-based service handling in our Session Initiation Protocol (SIP) based Internet telephony infrastructure and our Language for End System Services (LESS).",

URL="<http://www.cs.columbia.edu/~xiaotaow/rer/Research/Paper/ccnc2004.pdf>",

AUTHOR="Ron Shacham and Henning Schulzrinne and Srisakul Thakolsri and Wolfgang Kellerer",

TITLE="The Virtual Device: Expanding Wireless Communication Services through Service Discovery and Session Mobility",

TYPE="Technical Report",

INSTITUTION="Department of Computer Science, Columbia University",

ADDRESS="New York, NY",

NUMBER="CUCS-001-05",

MONTH=Jan,

YEAR=2005,

LANGUAGE="English",

KEYWORDS="Internet multimedia; mobile communications; ubiquitous computing, location-based services",

ABSTRACT="We present a location-based, ubiquitous service architecture, based on the Session Initiation Protocol (SIP) and a service discovery protocol that enables users to enhance the multimedia communications services available on their mobile devices by discovering other local devices, and including them in their active sessions, creating a virtual device. We have implemented our concept based on Columbia University multimedia environment and we show its feasibility by a performance analysis.",

URL="<http://www1.cs.columbia.edu/~library/TR-repository/reports/reports-2005/cucs-001-05.pdf>",

3. METHODOLOGY/PROJECT WORK

3.1 PROCEDURE IDENTIFICATION

1. Requirement analysis and definition

The system's services, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

2. System and software design

The systems design process partitions the requirement to either hardware or software systems. It establishes overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

3. Implementation and unit testing

During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

4. Integration and system testing

The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

5. Operation and maintenance

Normally this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of the system units and enhancing the system's services as new requirements are discovered.

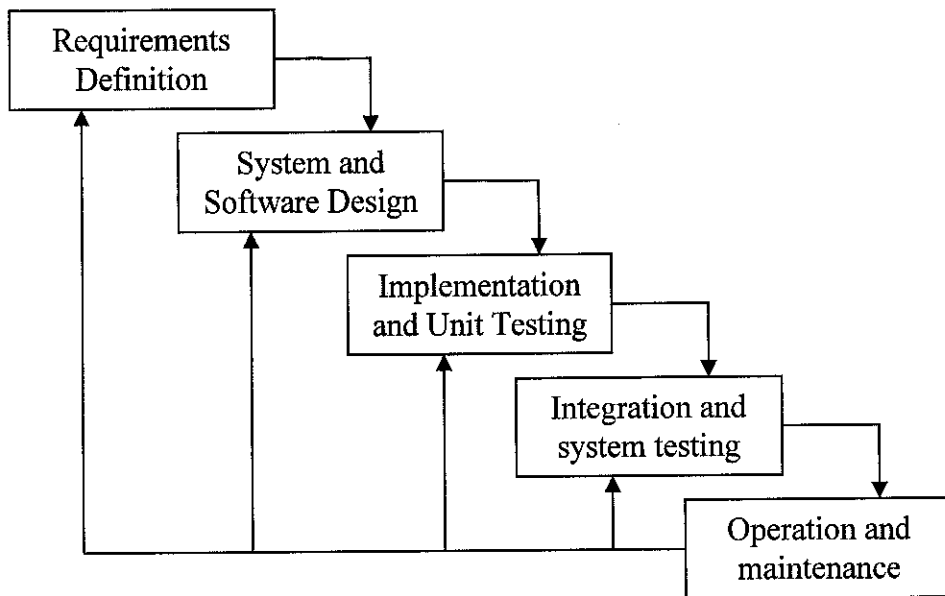


Figure 3.1: The Waterfall Model

3.2 TOOL REQUIRED

3.2.1 SOFTWARE

3.2.1.1 PROGRAMMING AND CODING

- JAVA MOBILE AGENTS WITH AGLET

3.2.1.2 WEB SERVER

- Linux

3.2.1.3 BROWSER

- Mozilla
- Internet Explorer

3.2.1.4 DOCUMENTATION AND PRESENTATION

- Microsoft Word
- Power Point
- Microsoft Project

3.2.2 HARDWARE

- Operating System : Linux
- Processor : Intel Pentium 4, 2.27 GHz
- Memory : 512 MB of RAM
- Display : G Force MX 400
- Display Mode : 1024 x 768 (16 bit) (60 Hz)
- Network : Internet TCP/IP Connection
- Input : Mouse and Keyboard
- Hard Disk requirement :50 MB

4. RESULT AND DISCUSSION

4.1 CONFIGURATION REQUIREMENT

In order to discuss typical requirements and approaches for configuring components and mobile agents, first need to be present different variants of system structures for remote configuration over the Internet. In its simplest form—depicted in Figure 1—the host for remote configuration is directly connected to a host where the components to be configured are installed and possibly activated.

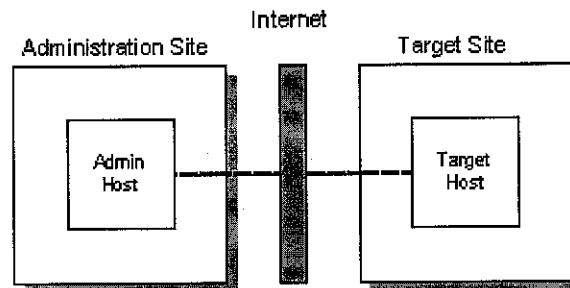


Fig 1: A simple system structure for remote configuration

We call the location where remote configuration tasks are performed by human operators the administration site. The administration site may be only one host or a network of hosts, which may all be used for configuration purposes. The location where the software is installed and running is called the target site. If the components to be configured are implementing the middle-tier of a three-tier application model, the target site might be a single computer with the application server hosting these components as shown in Figure 1.

A typical agent-based system, however, is a distributed system where the components to be configured are distributed to a number of hosts at the target site (T1, T2, ...) as depicted in Figure 2. Configuration may be performed from different hosts at the administration site (A1, A2, ...), also shown in Figure 2.

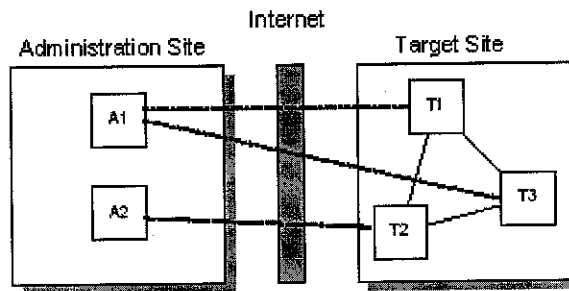


Fig 2: Configuring a distributed system from multiple hosts

Usually company networks are guarded by firewalls and not every host at the administration site may directly access the Internet. Likewise only selected hosts at the target site are visible to the Internet. Communication has to be routed through proxies (P) at the administration site and through dedicated entry points at the target site as depicted in Figure 3. In addition, the host acting as proxy in Figure 3 may also serve as administration server (AS) for centralized management of configuration tools and component repositories.

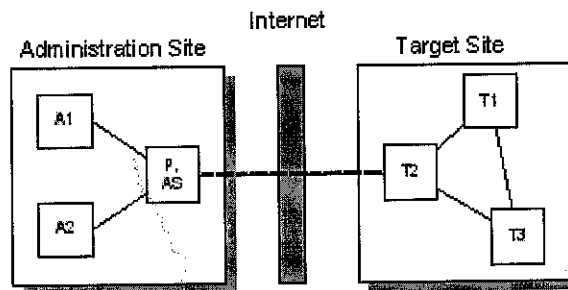


Fig 3: System structure with firewalls in mind

The presented system structures for remote configuration serve as the basis for the description of requirements on remote configuration systems in general and on our system in particular. Important requirements are:

- a. *Dynamic configuration of individual mobile agents and of the system structure.*

We need to support the configuration of both individual mobile agents and general system properties and structure. System structure is defined through agent communication relationships. Parts of the structure may be defined through rather

fixed relationships that can be changed manually. For example, our system allows the configuration of publish/subscriber relationships between agents. General system properties may be changed by configuring special agents that are responsible for distributing the information within the target site (see Section 4). Dynamic configuration refers to the ability to configure the system while it is up and running. This requires a highly dynamic system architecture which allows adding and removing components at run-time – a natural feature of any agent-based system. However, it also requires special protocols to change the properties of individual agents. Mobile agents are active objects encapsulating their own thread of control. It is not possible to change a certain property at any time and sometimes it is not possible to change an agent's properties at all. This has to be taken into account when designing protocols for updating agent state at run-time.

- b. *Minimal administration of configuration tools at administration site:* This requirement refers to the administrative effort that is involved in managing the configuration tools and repositories at the administration site. Changes or updates of the tools itself should require no or only minimal activities at the configuration hosts (see A1, A2, ... in Figure 3). Pre-installing the configuration tools at each configuration host is not desirable. Centralized configuration can be achieved by loading the tools on demand from a central administration server (see AS in Figure 3). This requires a dedicated run-time environment at each host. In the ideal case such an environment is a standard equipment of the client host, like web browsers, which are able to host HTML-based user interfaces. If HTML-based user interfaces are not powerful enough, additional environments for hosting user interfaces based on other technologies have to be preinstalled at each configuration host. Examples are the Java Plug-In [8] and Java Web Start [9] technologies for Java-based user interfaces. This is still preferable to installing the application at each host, since update and other changes of the configuration tools require no management activity at the client hosts.
- c. *Support for different types of configuration clients:* The rise of mobile and wireless computing is leading to a large number of different end-user devices with different display sizes and capabilities. The system structure at the administration

site—as depicted in Figure 3—is also appropriate for supporting different kinds of configuration clients (A1, A2, ... in the figure). An administration server (AS) could provide different user interfaces depending on the end-user device used for configuration. For example, it might provide WML-pages for a WAP-enabled device [10].

- d. *Loose coupling of tools at administration site and of components at target site:* Certain implementation decisions might lead to a tight coupling of the tools at the administration site and of the agents at the target site. Tight coupling may be the result of using a platform specific type system for configuration data, since this presumes that agents and tools are based on the same platform. For example, if configuration data is represented as Java objects both tools and agents need to be Java-based. Platform independent data formats and type systems (e.g., based on XML) are more flexible, since tools and target components may be implemented in any language. However, such type systems may not be as expressive as platform-specific ones, confining agent properties to simpler data types with no associated behavior. In the case of agent-based systems one might be tempted to install an agent platform not only at the hosts of the target site but also at the hosts of the administration site. However, this also leads to tight coupling of administration site and target site since it assumes that the configuration tools are only used for configuring agents of a particular agent platform. This rules out systems like ours, where one administration site is used for configuring multiple target sites with possibly different agent systems installed. We will present our solution to this problem in Section 3.1. In addition, the notion of migrating an agent to an administration host, changing its configuration and sending it back to the target site is often not feasible. Two problems that come immediately into mind are security and agent activity. A firewall aware system structure as depicted in Figure 3 would need flexible agent platforms that allow control of message routing. However, agent platforms usually support peer-to-peer communication as depicted in Figure 2. Also, firewall settings at the configuration site might not allow an agent entering the site at his will; most of the time even callbacks are denied. A further problem is that an agent is an active entity. It is

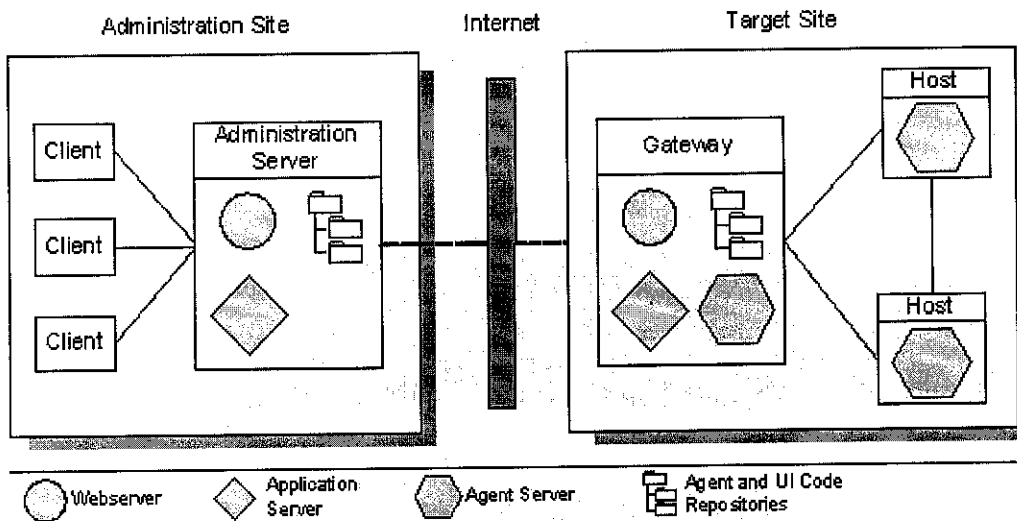
often not possible to stop an agent's activity just for changing some configuration settings.

- e. *Evolution support*: In dynamically adaptable systems components (mobile agents in our case) are added, removed and replaced by newer versions over time. Multiple versions of the same component may exist simultaneously in the system. This is supported by mobile-agent systems, since features like code mobility require flexible mechanisms for code management. Typically the agent system provides separate name spaces for different agents and a code loader which makes sure that the code of different versions of the same agent type can be loaded at the same time [11]. From the configuration viewpoint we have to make sure that we are able to configure an agent at any time during its life time. Even if some agent code has been removed from the repository at the administration site or if it has long been replaced by newer versions there may still exist some instances of older versions at the target system, which need to be configured. The most obvious solution to this problem is to store the user interface code for configuring the properties of a particular agent with the agent itself at the target site. If the agent is to be configured, the user interface code is requested from the agent and sent to the tools of the administration site (code on demand [12]). Otherwise the user interface code is integral part of the agent and is transferred along with agent state and code when the agent is roaming the network at the target site. However, the solution of storing user interface code with the agent itself also has drawbacks. It is a form of tight coupling of the target site with the tools at the administration site, since the user interface code needs a special execution environment at the administration site. In addition, multiple different end user devices for configuration are not supported. Still it may be useful for some kind of remote configuration systems and we will present a similar approach in Section 3.1. A better solution is to store a platform independent user interface description with the agent. This allows device independent user interface generation at the administration site while maintaining the ability of configuring each agent in the system. A further enhancement is generating the user interface by analyzing the agent itself. We present such an approach in Section 3.2.

f. *Minimization of user interface development*: Development of user interfaces for remote configuration is a tedious task and component (agent) developers should focus on developing the application logic instead of providing remote configuration support. In the ideal case, no user interface needs to be developed at all. One approach for supporting user interface development tasks is user interface frameworks. Component developers just need to adapt general framework classes providing generic functionality for setting new values, for reverting to old values and for performing consistency checks. As stated above this approach not only involves coding effort but also tightly couples configuration tools to the platform of the user interface framework. For example, a Swing-based user interface requires a Java runtime environment at the administration site. A better approach is to generate the user interface from some kind of User Interface Specification Language (UISL). This is platform independent but still the user interface has to be specified. The most preferable approach is generating the user interface by analyzing the agent itself. This approach is based on the availability of meta-data about components, a distinct feature of each component-based system (see [5] for the importance of meta-data). By using meta-data the user interface can be generated automatically and involves no development effort at all. Meta-data is usually extracted from component implementation and interfaces and is stored as part of the component. However, meta-data provided by component platforms like Java and .NET often lacks important information that is necessary for generating “well-formed” user interfaces and for providing sufficient validation of component property values. In Section 3.2 we present an approach for automatically generating user interfaces from enhanced agent meta-data.

We have outlined and discussed basic requirements and solutions for remote configuration of dynamic and adaptable component-based systems. Most of the presented requirements are typical for remote configuration of component-based systems in general. Some are imposed through the use of mobile agent technology.

4.2 CONFIGURATION OF INDIVIDUAL AGENTS



Agents are installed and configured from configuration clients at the administration site. Upon installing an agent, its code, an initial configuration and its user interface code are transferred to the target site. The user interface code is not directly stored as part of the agent code. Instead, it is stored in a code repository at the target site. In principle, this would allow to implement the user interface for configuration based on other technology than the agent itself. In our system, however, both user interface and agent are implemented in Java (AGLETS) and LINUX. An agent does not store its user interface code directly but holds a unique ID that identifies the user interface code in the repository. If a configuration request is issued from one of the clients at the administration site, this ID is requested from the agent and used for identifying and transferring the user interface code to the configuration client.

Storing the user interface for configuring an agent in a repository at the target site ensures that for each agent that has been installed at the target site a configuration user interface can be found, no matter which administration site is used. Administration clients may

even be placed within the target site since, from a logical perspective, the user interface that is needed for configuring an agent is always with the agent. From a technical perspective this solution enables code sharing. An installation tool might check whether an appropriate user interface for a newly installed agent is already available at the target site and assign its unique ID to the agent.

Storing the user interface code at the gateway server does not raise security problems, as only properly authenticated users are allowed to install or change mobile agents at a target system. Therefore the issue of malicious target sites tampering with the stored user interface code can be omitted.

We should note that we have also experimented with implementing the user interfaces themselves as agents and thus using agent mobility for transferring the user interface to the configuration clients at the administration site. This proved not feasible for mainly two reasons: (1) Configuring multiple target sites with different agent platforms is not possible and (2) agent platforms are not adaptable to the underlying network infrastructure.

First, we need to administrate multiple target sites based on different agent platforms from one administration site. The user interface as agent would require an agent platform at the configuration client. Since target sites can use different agent platforms, a client would need multiple agent platforms for configuring agents from different target sites. However, standardization would need to include the underlying execution platform (e.g., the Java platform). We have defined an Agent Platform Abstraction Layer (APAL) specifying platform-independent abstractions for agent creation, disposal, communication and migration. This allows at least platform independent implementation and configuration of agents at the administration site and thus supports different agent platforms at different target sites (We should note that the implementation is still confined to Java-based agent platforms).

The second problem is concerns about network security based in firewalls. Corporate networks are usually secured by (multiple layers of) firewalls. Agent platforms need to be

adaptable in terms of message routing and protocols to operate in such environments. However, typical agent systems are designed for operating in open environments based on peer-to-peer connections between agent servers. An additional problem for agent mobility is that accessing the administration site from the target site is prohibited by firewall settings.

We have implemented an adaptable communication infrastructure, which is used for sending agent properties from the configuration clients in Figure 4 to an agent at one of the agent servers at the target site. Communication is routed through a proxy at the administration server and through the host acting as entry point (gateway) at the target site. Agent properties are not directly updated. Instead, the target agent first caches the configuration data and updates its properties only if it reaches a consistent state.

Configuration data is encoded as Java objects. This might tightly couple user interface and agent code and imply that configuration user interfaces need to be Java based. However, this is not the case in our system. The target site can only be accessed through the gateway host. Messages from external sources like configuration clients are routed through an application server at the gateway host which converts the protocol to the native protocol of the agent platform at the target site.

4.3 A GENERATIVE APPROACH FOR CONFIGURATION UIs

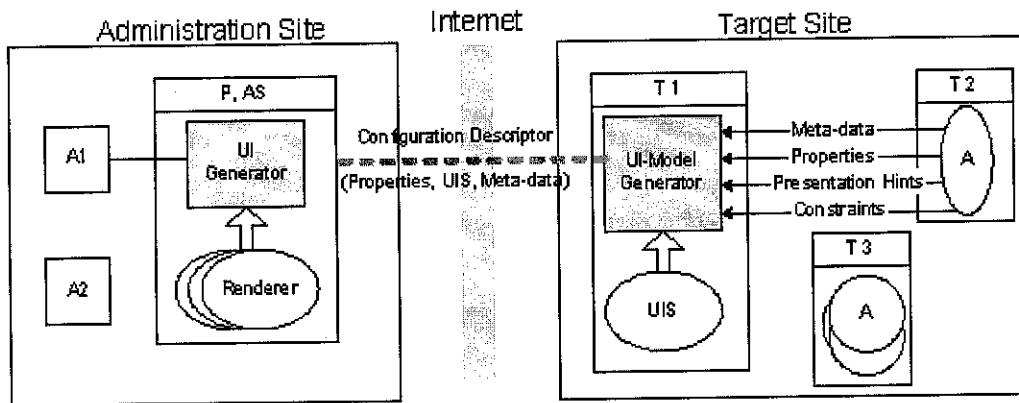


Fig 5: Generic User Interfaces

The basic system structure is similar to the one presented in the previous section and is shown in Figure 5. The figure also shows the main system components and the data needed for user interface generation.

In our system, agents are implemented in Java. Therefore meta-data about agents, like configurable properties, can be retrieved using introspection and reflection [16]. Meta-data and property values are transferred from an agent (A) to the UI-Model Generator as shown in Figure 5. The UI-Model Generator automatically generates an XML-based User Interface Specification (UIS) and transfers it together with the meta-data and property values to a User Interface Generator that is located at the administration server at the administration site. We call the package consisting of UIS, meta-data about properties, and property values Configuration Descriptor (see Figure 5). Property meta-data and values are not represented as Java objects in the configuration descriptor, since this would lead to a tight coupling between the tools at the administration site and the components at the target site (see requirement 2d). Instead, we convert the retrieved meta-data (property names and types) as well as property values to a platform independent representation based on XML.

In principle, meta-data about property names and types is sufficient for automatically generating the user interface. However, the meta-data extracted from agents lacks

important information like units of measurement and allowed ranges for property values. This information is needed for presenting and validating property values at the user interface. We enable an agent developer to provide such information either using an extended meta-data API or by providing XML-based constraint specifications for individual properties, which have to be deployed with the agent code. These constraints are sent to the user interface generator at the administration site as part of the configuration descriptor.

```
<dialog for="insight.agent.logfile.LogfileAgent" label="Logfile Agent Properties">
  <category label="General">
    ...
  </category>
  <category label="Task Schedule">
    ...
  </category>
  <category label="Protocol Files">
    <input name="rootDirectory" label="Root Directory:"> </input>
    <list name="files" label="Files: "> </list>
    <input name="expression" label="Expression:"> </input>
    <check name="sendFiles" label="Attach Files:"> </check>
  </category>
</dialog>
```

Fig 6: Presentation Hint Example

User interfaces based on constraint-enhanced meta-information are still rather crude in appearance. For example, field names that are derived from component properties are not verbose enough and all fields are just presented as one long list and lack semantic grouping (see left part of Figure 7). We allow agent developers to enhance the user interface layout and appearance by providing presentation hints as shown in Figure 6.

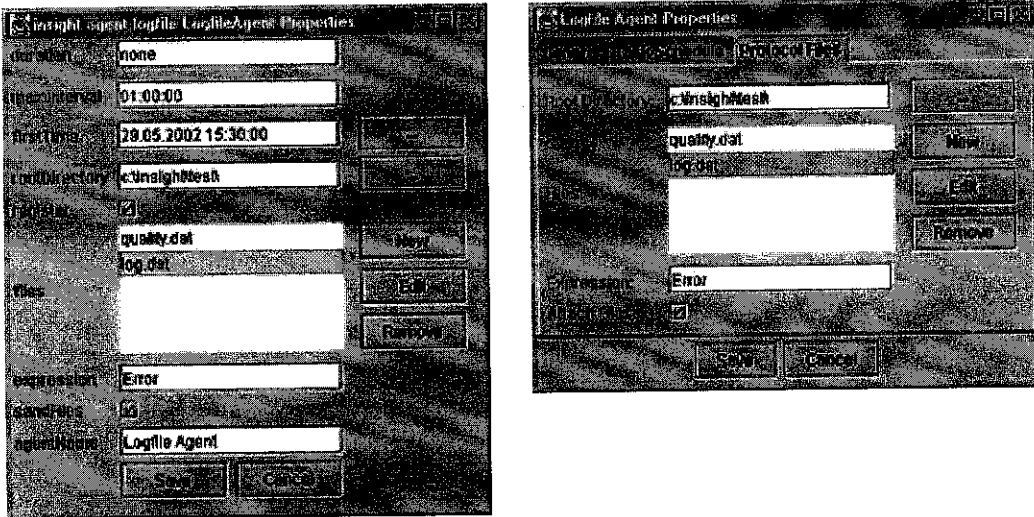


Fig 7: Use Interface without/with presentation hints

Using this information we are able to improve the appearance of the user interface as shown in Figure 7. The user interface shown in the right part of Figure 7 has been generated using the presentation hints presented in Figure 6. The main differences are verbose field labels and more clearly arranged user interface elements.

Summarizing, meta-data about properties, constraint specifications and presentation hints are extracted from an agent and sent to the user interface model generator at the target site. The model generator creates an XML-based user interface specification (UIS) which is transferred to the administration site along with property meta-data and property values in XML-format. The generated UIS is a hard- and software independent description of the layout of the agent properties and thus independent of any specific configuration client (see requirement 2c).

Instead of providing presentation hints, the complete UIS may be created manually and stored in a UI repository at the gateway host (see Figure 5). The main advantages of this approach are even more elaborate user interfaces, albeit the effort for user interface specification is increased, also.

The configuration descriptor (including the UIS) is transferred to the user interface generator at the administration site, which finally generates a client specific user

interface. The UI generator uses pluggable renderers for generating different kinds of user interfaces. The user interfaces depicted in Figure 7 have been generated using a JFC/Swing renderer. Renderers for HTML, WML and other kinds of user interfaces may be provided as well.

4.4 LIMITATION OF THE PROJECT

In this project, it cannot be completely done because of individual limitation. The part that cannot be done is where the connection between mobile agent GUI and Linux firewall configuration. Supposedly after mobile agents have sent their message such as firewall setup, Linux will receive the message and automatically setup the firewall and saved it. This part are not completed because of the complexity of the mobile agents which use the java-based language and high skills and expertise also needed to complete this project due this is the first mobile agent project in Universiti Teknologi Petronas.

5. CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION

We have presented requirements and approaches for configuring remote and mobile components in a typical real world setting. Currently we use the system for configuring mobile agents performing monitoring and supervision tasks in process automation systems. Many of the presented requirements and solutions are important and useful for remote configuration of distributed components in general.

The use of mobile agent technology as the basis for the components at the target system imposes specific requirements on the configuration system like support for dynamically adaptable system structure and agent mobility. In terms of implementing the configuration system itself, we had to sacrifice seemingly obvious solutions for configuring remote agents (like migrating the agent and performing the configuration locally) in favor of other techniques like code on demand and automatic user interface generation.

5.2 RECOMMENDATION

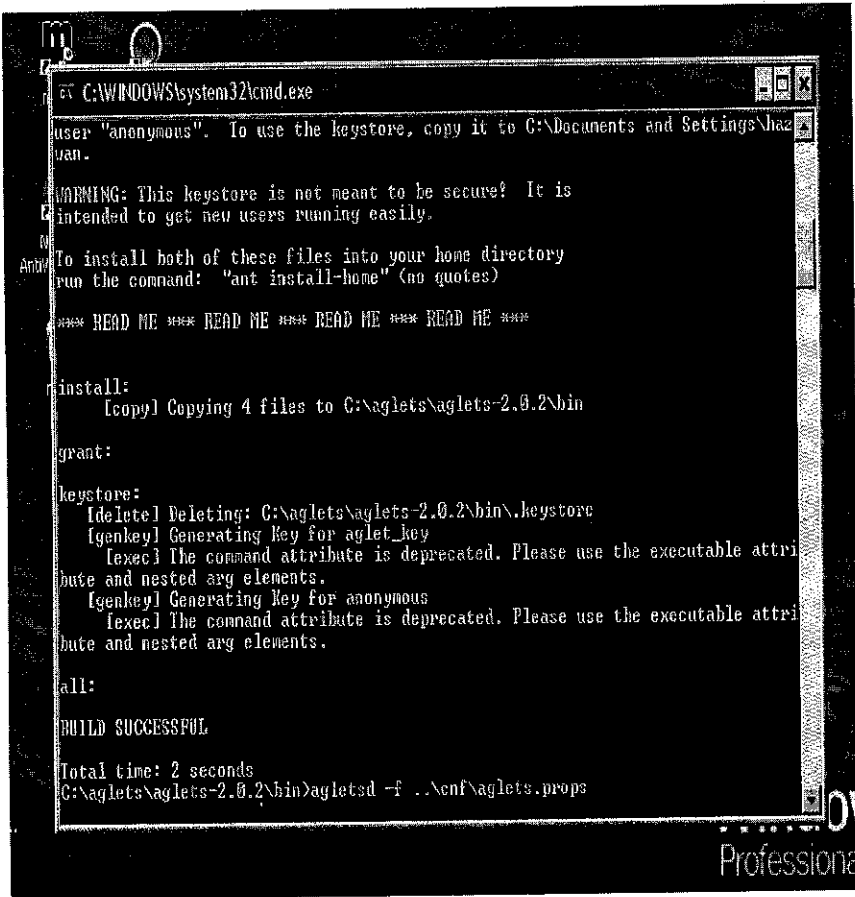
There are recommendation and suggestions that can be done in the future for the system enhancement:

- Enhance the usability of the mobile agent with the prediction features

REFERENCE

- Programming And Developing JAVA Mobile Agents With AGLETS
- R. Weinreich, R. Plösch: "An Agent-Based Component Platform for Dynamically Adaptable Distributed Environments", *Informatica Journal*, Special Issue on Component Based Software Development, Vol. 25 Nr. 4, November 2001.
- Wireless Application Protocol Forum: "Wireless Application Protocol Architecture Specification, WAP-210-WAPArch-20010712", July 2001.
- A. Fugetta, J.P. Picco, G. Vigna.: "Understanding Code Mobility", *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998.

APPENDICES



```
C:\WINDOWS\system32\cmd.exe
user "anonymous". To use the keystore, copy it to C:\Documents and Settings\haz
van.

WARNING: This keystore is not meant to be secure! It is
intended to get new users running easily.

To install both of these files into your home directory
run the command: "ant install-home" (no quotes)

*** READ ME *** READ ME *** READ ME *** READ ME ***

install:
[copy] Copying 4 files to C:\aglets\aglets-2.0.2\bin

grant:

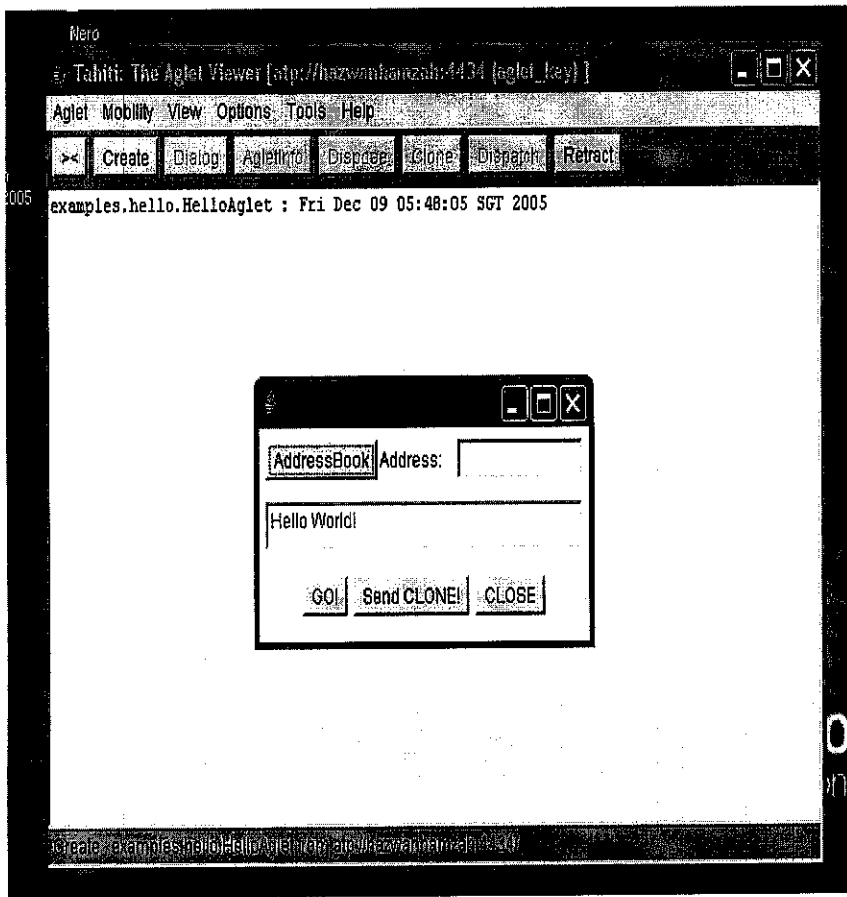
keystore:
[delete] Deleting: C:\aglets\aglets-2.0.2\bin\keystore
[genkey] Generating Key for aglet_key
[exec] The command attribute is deprecated. Please use the executable attri
bute and nested arg elements.
[genkey] Generating Key for anonymous
[exec] The command attribute is deprecated. Please use the executable attri
bute and nested arg elements.

all:

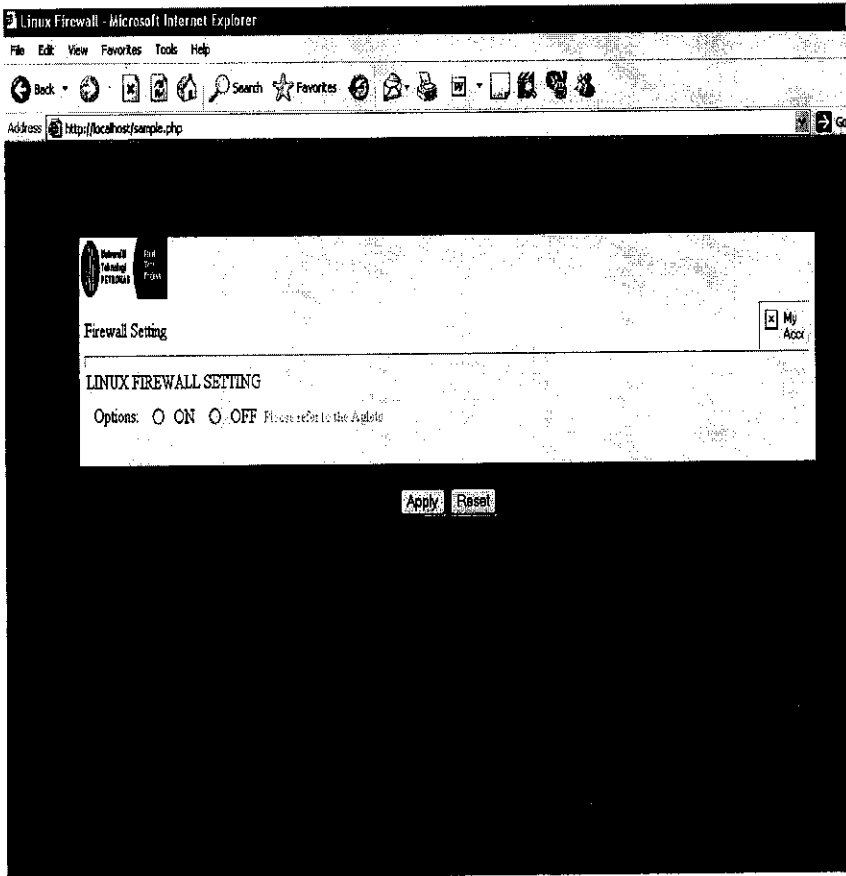
BUILD SUCCESSFUL

Total time: 2 seconds
C:\aglets\aglets-2.0.2\bin>agletsd -f ..\conf\aglets.props
```

- Aglets installation – done in MSDOS
- Software needed – ASDK and JDK



- Tahiti server – mobile agents
- Send message – configure firewall to Linux



- Linux Firewall Settings -- receive message from mobile agent then respond either to turn on/off

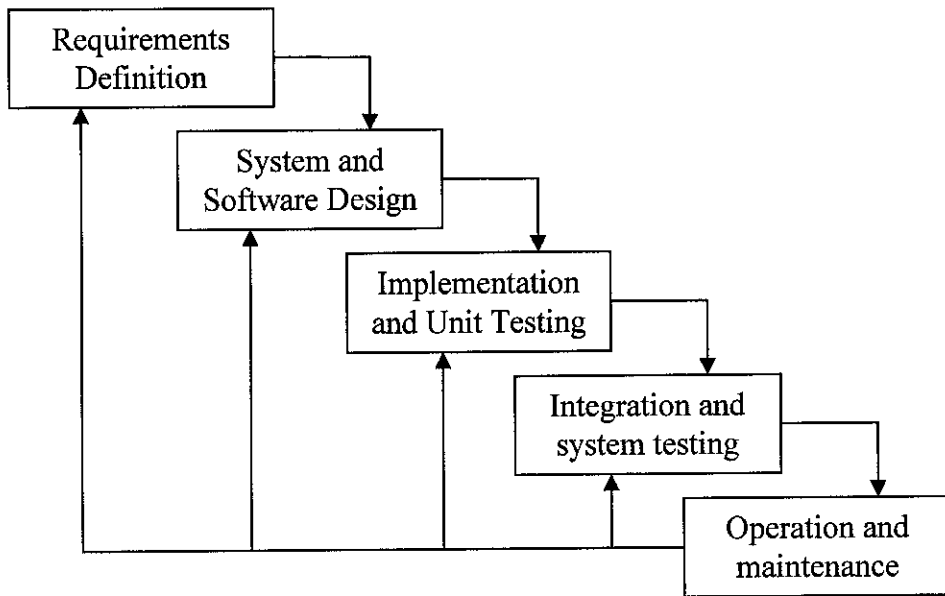


Figure 3.1: The Waterfall Model