SERVER WATCH

BTech 450DT – Final Report

Anish Doshi

ados001@ec.auckland.ac.nz

Academic supervisor: *Dr. S. Manoharan*

October 25th, 2007.

Department Of Computer Science

University Of Auckland

TABLE OF CONTENTS

ABSTRACT

The value of a service provided by a server is very high. This makes it very crucial to monitor the servers when they are running a valuable service. When such a server breaks down, the cost of not providing the service is very high. There are many different ways in which this problem can be tackled. But these servers need to get fully functional and start providing its service as soon as possible.

For this project, we create a system which will monitor the health of a server at regular intervals. This information can be viewed from a mobile application. Monitoring a server through a mobile phone gives the administrator the power to monitor the server from any part of the world where he can access a mobile network. Also, this system adds the functionality to alert the user by sending warning messages through SMS (Short Messaging Service).

1. INTRODUCTION

When computer systems grew in size, the concept of distributed systems was invented. Distributed computing is a method of running different parts of the application on different computers, which connect each other over a network (Distributed computing., n.d.). This gave birth to the idiom of client/server architecture. Server is a computer or software that runs on a computer that provides specific kind of service to client software running on the same computer or other computers on the network. Client is a computer or software on a computer that requests utilize the service served by a server. Also, the server application can be distributed over different computers.

The services served by a server are the most valuable resources for some systems and eventually for the organization (e.g. a web hosting server, a banking system or a mail server or a database server). The organizations depending on these services suffer a setback when these servers go down. There are many ways to monitor this. But they are all traditional ways in which the servers are monitored using an application that resides on the server or on the network that the server resides on. The server administrator needs to be on a computer to check the health of the system. The other way is when some one whose client depends on the server calls the administrator and notifies the administrator. This is of course not a preferred method.

This report starts with a brief introduction to what the system that is being developed in this project is about followed by detailed information about the research done prior to the development of the project. Then we will have a look at the different design options and the final design on which the system is developed. Then we will see a breakdown of different parts of the system that were developed and look at them in details. The report concludes by suggesting future developments to the system and the learning outcome of doing this project.

1.1. THE PROJECT

The aim of this project is to design and deliver a system which can monitor a Windows-based server using a browser as well as a mobile device. Also, this system should be able to deliver alert messages to the user in the case of an emergency such as server shutdown or high usage of resources for an abnormally long period.

This system has to be designed and implemented from scratch. In the start, a research of the possibilities and scope of such a monitoring system is done. Then the technologies that be used for this system are researched. The system is then designed and implemented from scratch. A few technologies have already been researched by the industry mentor as well as academic supervisor. A detailed analysis of these was done during the course of this project.

1.2. THE COMPANY

MCS was founded in 2006 to develop new ways for humans to easily interact with their remote assets and information relating to important assets via mobile phone. MCS is now engaged in developing products in the marine asset monitoring, mobile commerce, and remote control and monitoring sectors. The focus is on developing new and easier ways for humans to use their mobile phones/internet to remotely control, monitor and interact with remote assets, information and services.

MCS major focus is the human interface to mobile devices. This involves optimizing informatics and human interface to mobile devices by the innovative use of application software in mobile phone and related devices.

2. PROJECT DETAILS

This section will present the problem that this project solves and the aim of this project.

2.1. MOTIVATION

The motivation of this project is to deliver a system which provides a virtual presence, with an asset of value, to the user. This system should not only be able to monitor the asset but also deliver alert messages to the user and also posses the ability to control this asset remotely.

A server that serves information for an organization is an asset of value to that organization. When a server breaks down, the cost of not providing a service is very high. A service needs to be continually served by its server. For e.g., if a web-site hosting server breaks down, the number of hits that the web-site gets goes down. This results into loss of advertising and/or poor quality of service. If a server of a banking system breaks down, many services provided by the bank will stop. There are different ways to tackle this problem. One of the ways would be to have a backup server. Another would be to setup grid computing (Snavely, Chun, Casanova, Van der Wijngaart, & Frumkin, 2003) But faster the breakdown is detected and faster the user is informed about the breakdown, faster the user can act upon the situation. And hence reducing the time taken to get the server up and running. When monitoring a server, a user should be able to monitor it from anywhere, whether he is on a computer or not.

2.2. TECHNICAL REQUIREMENTS

The aim of this project is to monitor and control a Windows based server running on .NET platform, remotely. To monitor a server remotely, this information has to be sent over an easily accessible medium - Internet. This enables us to monitor it using a browser from a laptop or a computer. Internet also makes it possible to view this information on a mobile phone. Mobile phones enable the user to monitor the server on-the-go. They can access this information from anywhere, anytime. This system should be able to work on a very wide range of mobile devices.

The system should be able to control facilitate user to control a few basic operations of the server. Below is the list of elements that this system is supposed to monitor. Also, the basic operations that are to be performed on the server are listed below.

The system should be able to monitor the following elements of the server:

- Average CPU utilization, an average CPU utilization metric must be maintained (resettable).
- Average CPU temperature, an average CPU temperature must be maintained (resettable).
- Peak CPU temperature, a peak CPU temperature must be maintained (resettable).
- Average RAM usage, an average RAM utilization metric must be maintained (sample time base must be configurable).
- Peak RAM usage, a peak RAM usage metric must be maintained (resettable).
- Current hard drive usage, the current value of used hard drive space (all installed hard drives).
- (S.M.A.R.T) Average hard drive temperature, an average hard drive temperature must be maintained (resettable).
- (S.M.A.R.T) Peak hard drive temperature, a peak hard drive temperature must be maintained (resettable).
- (S.M.A.R.T) Error read rate, the error read rate will enable the diagnosis of a failing hard drive (research to be done into what suitable information should be retrieved from the S.M.A.R.T information to diagnose a failing hard drive).
- List current executing processes, a snap shot of the names of the systems currently executing processes.
- A break down of each executing processes resource utilization (CPU usage, RAM usage etc).

The system should be able to perform the following basic operations on the server:

- Reset the server (hardware and software).
- Start a process.
- Stop a process.

The above functionalities should be performed by a password protected application (web or client). Along with a mobile application, the system should also provide these functionalities of the server through a website. The system should also be able to send alerts to the user through SMS in case of any problems detected in the server. The diagram below shows an overview of the system.

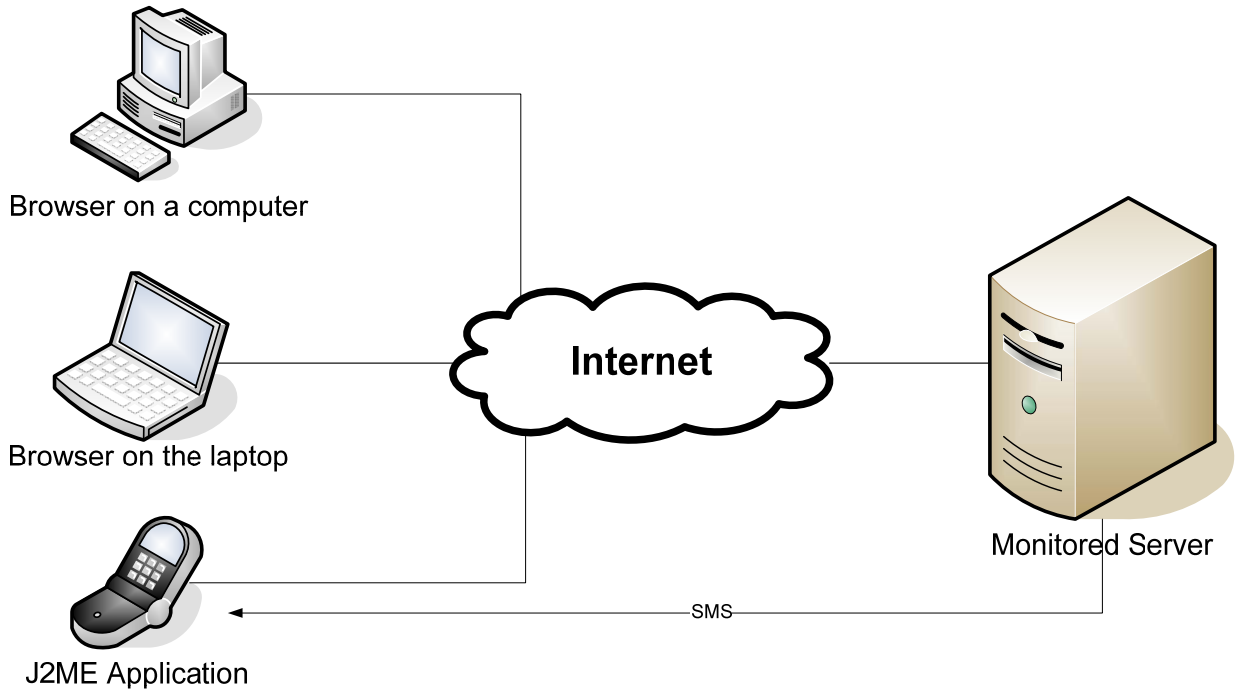


Figure 1: Overview of the system.

2.3. Learning Opportunities

This project will give me a lot of learning opportunities starting from project basic research of technologies to project management. Since the system is to be build from scratch and I am the only person working on this project (of course under supervision of my academic), there is quite some project management tasks to be learned. Another thing to learn here is how to document the project details and how important the communication between the developer and the client/company is.

3. TECHNICAL INVESTIGATION

This section is divided into two sections. First section discusses the ways in which the elements of the server can be monitored and controlled. Second section talks about the ways in which the mobile application can be implemented. The section after that would introduce us to other key technologies.

3.1. MONITORING SERVER

For a .NET platform, the best way to get information about the hardware, services running on the computer and processes running on the computer is by using Windows Management Instrumentation (WMI). Here, we will investigate what WMI is and how useful it is to fulfill our goals of the project.

3.1.1. WINDOWS MANAGEMENT INSTRUMENTATION (WMI)

"WMI is an infrastructure for managing data and operations on Windows Operating systems." (Windows Management Instrumentation (WMI), n.d.). WMI is used to automate administrative tasks on remote computers. It can also supply management data to different parts of an operating system. "WMI uses the Common Information Model (CIM) industry standard to represent systems, applications, networks, devices, and other managed components." (About WMI, n.d.) These WMI applications can get management data or perform operations on the operating system or its services on a variety of languages. We can write a client script or application to get data about hardware or software from WMI. Also, we need to provide data from hardware or software to WMI by creating a WMI provider. Provider basically translates the WMI script into its corresponding windows API. It gets the information and then provides that information to the client (The .NET Show: WMI Scripting, 2006). All services, processes, hardware such as monitor, hard drive, memory (RAM), etc. have their own WMI class. These classes are the providers of management data. WMI basically provides a level of abstraction to get specific information about the operating system and the processes and services on it as well as perform operations on them. This achieved by using WMI queries. This query language is called WQL (WMI query language) (Querying with WQL, n.d.). WQL is a subset of ANSI SQL. Below is a snippet to give an example of a WQL query.

SELECT FreeSpace FROM Win32_LogicalDisk WHERE DriveType=3

The above query will return the free space in a disk where the type of disk is 3. Similarly,

SELECT CurrentClockSpeed FROM Win32_Processor WHERE Manufacturer='Intel'

will get the current speed of all the processor whose manufacturer is Intel. This makes it easy to retrieve information about the system. In a similar fashion we can also query information about processes and services using Win32_Process and Win32_Service classes.

3.1.2. WMI .NET

The Technology Summary for WMI .NET link from (WMI .NET Overview, n.d.) says "Windows Management Instrumentation (WMI) is a component of the Windows operating system that allows you to programmatically access management information in an enterprise environment". The WMI .NET framework is built on the same WMI technology. It gives us the benefit of using WMI technology in a .NET application. Using the System.Management namespace in the .NET framework, we can run WMI on different .NET languages like C# .NET, Visual Basic .NET, etc. System.Management.

Instrumentation workspace can be used to instrument the application so that it can provide information to WMI layer about the behavior of the application. The key benefits of WMI are as follows (Benefits of WMI in .NET Framework, n.d.):

- Leverage of common language runtime features, such as garbage collection, custom indexer, and dictionaries.
- Definition of classes and publication of instances entirely with .NET Framework classes to instrument applications so the applications can provide data to WMI.
- Simplicity of use.
- Access to all WMI data.

There are a few limitations of WMI which are not relevant to our requirements. Hence, I have not placed them here. Below is a diagram showing the framework of WMI.

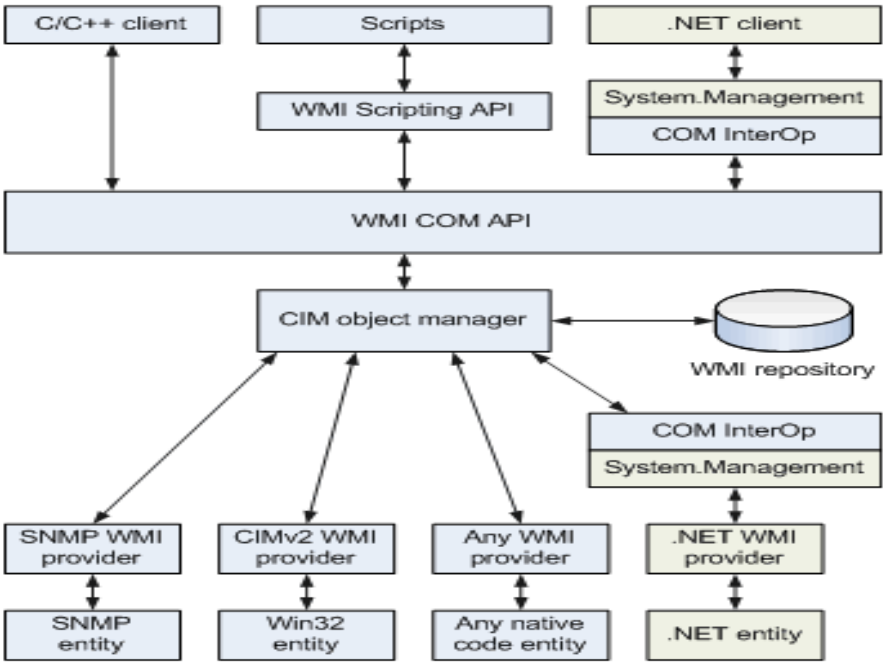


Figure 2: Overview of WMI framework.

As seen in the above figure, the WMI COM API acts as an interface between different types of manage objects like Win32 entities, a .NET entity, etc. And any programming languages, including scripting languages can access this information via WMI COM layer.

3.2. Mobile Application

The purpose of the mobile application is to be able to display the information about the elements of the server on a mobile device. Many different types of technologies are present to make a mobile application. A few of these are discussed below.

3.2.1. J2ME

In (White, 2001) James White shows the very basic understanding of the J2ME platform. J2ME stands for Java Micro Edition. It's a lighter version of java which can be used on a low power, network connectivity and graphical user interface capabilities(White, 2001). J2ME provides an application platform to a very wide range of devices. This is because it provides the ability of writing the code once and running it on several platforms.

J2ME is divided into three elements: configurations, profile and optional APIs.

3.2.1.1. CONFIGURATIONS

Specifications that address the virtual machine and general Java API running on a large group of devices are called configurations. J2ME has been divided into two basic configurations, one targeted to the small devices and the other towards more capable mobile devices. The configuration for the small devices is called “Connected Limited Device Configuration” (CLDC) and the other is called “Connected Device Configuration” (CDC). CDLC is specifically designed to meet the needs for a Java platform to run on devices with limited memory, processing power and graphical capabilities.

3.2.1.2. PROFILES

On top of these on configurations, it also specifies a number of profiles for describing a set of higher-level APIs that could further define the application. A profile addresses the needs of the specific subset of devices.

MIDP 1.0

MIDP stands for Mobile Information Device Profile. The MIDP profile the core profile for CLDC. It was developed specifically for mobiles and pagers (Kolsi, 2004). A MIDP application is called a MIDlet. A MIDlet has no knowledge of other MIDlets through the MIDlet API. MIDP 1.0 environment includes APIs that can define the MIDP application model and capabilities available for MIDlet applications.

MIDP 2.0

“MIDP 2.0 enhances the overall end user experience, improves application portability, and provides greater extensibility” (What's New in MIDP 2.0?, n.d.). MIDP 2.0 has more flexibility layout for greater application portability. Its also has greater extensibility. For e.g., it has something called Custom Items. This is used mainly to create our own forms and UI components. MIDP 2.0 also has support for media files. (What's New in MIDP 2.0?, n.d.) also describes about the new OTA (Over-the-air) feature which only a recommended feature in the previous version. The MIDP specifications define how the application suites would behave in a particular environment.

3.2.1.3. SECURITY COMPARISON

(Kolsi, 2004) talks about a lot of security issues concerning Mobile Environment including threats in this article. They talk majorly about Internet environment threat, Mobile Network threats, Java threats and MIDP 1.0 security problems. This article then evaluates the problems MIDP 1.0 security. It evaluates the security threats in the MIDP 1.0 and tells the solutions that are provided in MIDP 2.0. The table below summarizes the problems faced with MIDP 1.0 and the solutions provided by MIDP 2.0.

Shown below is the table that describes the level of protection provided by MIDP 1.0 and how the threat not handled in MIDP 1.0 is handled in MIDP 2.0.

Threat	*MIDP 1.0*	*MIDP 2.0*
Network Security threats in GSM/GPRS or internet protocols	Application level protection on top of HTTP	**SSL/HTTP security protocols**
Unauthorized network connections	User permissions	**Protection domains, security policy and user permissions.**
Physical threats against devices	Application level protection for data	**Application level protection for data.**
Malicious applications performing Dos	Flawless JVM and sandbox	**Flawless JVM and sandbox, signed applications and authentication of origin.**
Application binary integrity		**Signed applications**

Table 1: Security comparison between MIDP 1.0 and MIDP 2.0

In (Debbabi, Saleh, Talhi, & Zhioua, 2005) Mourad Debbabi et al. describe the wide spread use of J2ME and explain how it is susceptible to security threats. They then talk about the CLDC security architecture and describe how the CLDC architecture affect the security of a mobile application when it is ran on MIDP 1.0 environment. They then talk about the security problem solved by the MIDP 2.0 security model.

3.2.2. GUI

J2ME has a very limited support for GUI. The MIDP 2.0 profile features very basic GUI components. A very good looking front-end application is very necessary for the project. This makes the product of this project even more competitive in the market. As recommended in (Wesson & van der Walt, 2005), the users are very much inclined towards good visual appearance and screen layouts. MIDP clearly lacks these. So we need to find out different ways to enhance the appearance of the mobile application. There are a few commercial as well as Open Source GUI APIs available in the market.

3.2.2.1. J2ME POLISH

J2ME Polish is a commercial tool for building GUI for J2ME applications. J2ME Polish is suite of tools for creating "polished" J2ME applications. Each tool meets a definite need of J2ME developers: Build-tools with an integrated device-database, a powerful GUI, a framework for building localized applications, a game-engine, a logging framework and a collection of utilities. It supports a very wide range of mobile devices as well. J2ME Polish also has an Open Source version which is available under the GNU GPL license. This license needs the application made using J2ME Polish to be open source as well. If the application developing company wants it to be a closed source application, then the company has to pay for the licenses.

For the mobile application in this project, the decision was made to use J2ME because it was supported on a very wide range of devices. Other technologies such as FlashLite, etc. are platform dependent and are not supported on a very high range of devices. Hence, it was decided to use J2ME. Also, it was decided to create our own API for controls to be used on the mobile phone. This would give other developers at MCS a chance to aid GUI development of other projects as well as get some help from other projects to develop their own GUI. Also, browser applications (thin client) for mobile application would increase the data transfer because the HTTP header for browser pages is much higher compared to that for mobile application such as J2ME.

3.3 EMBEDDED DATABASE

(Koopmann, 2005) An embedded database is a software component that is a part of an application and not a separate running application. The end user has no knowledge of its existence. Embedded databases are fast and reliable. In this project, an embedded database can be used to store the user information for authentication and for logging the report generated by the system.

3.4 ISOLATED STORAGE

Isolated storage is, essentially, a special spot on the hard drive that only your application can find. The .NET Framework takes care of managing the actual disk storage. Once you've opened your isolated storage, you can create files or directories in it, and treat it pretty much like any other disk space. The nice thing is that even if your application doesn't have permissions to access the file system, you can still use isolated storage.

4. DESIGN

4.1. SYSTEM ELEMENTS

In this we will review the design that was created to develop the application for this project. The basic design of the system was to have four separate entities. These entities would perform their respective functions.

4.1.1. AGENTS

Agent is a component that monitors single element of the server. For e.g., a Processor agent will monitor only the CPU. Disk agent will only monitor a disk. This component uses WMI .NET to get information from the operating System. This component also logs the information that is retrieved using WMI for back tracing the fault. Also it should be able to perform a few tasks for system (e.g. kill a process of run a process, etc.)

4.1.2. REPORTING SERVICE

A reporting service is a component that runs the agents and gathers information from them. This service contains a communication layer which will provide a few functionalities of the agents to other parts of the system such as the mobile channel etc. A GUI configuration panel enables user to set the different parameter for the system such as the scan interval, max CPU usage, etc. It will also be responsible to send alert messages to the external device that would pass the message through SMS. Also it is responsible for polling this device indicating that the server is still running and the health of the server is still acceptable.

4.1.3. MOBILE CHANNEL

A mobile channel will facilitate the HTTP data to be transferred between the reporting service and the mobile application. Having a separate mobile channel has a few advantages like the mobile channel can send data with minimal HTTP headers. Custom encoding and compression can be performed for the mobile application. Upon receiving a request, this channel will query and retrieve data from the reporting service. This data will then be transferred into an xml string and passed to the mobile application by attaching required HTTP headers. This channel will act as a proxy for the mobile application. This will be built using .Net 2.0 framework in C#.

4.1.4. Mobile Application

A Mobile Application will enable user to access the information regarding the elements of the server on a mobile phone. This application will be build using J2ME. This application will accept xml (Quin, 2007) data from the server and display this information on the mobile application. The user interface will be semi-dynamic (depended on the type of information displayed). The user will need to authenticate himself before requesting any data from the server.

4.1.5. External Alerting Device

An external alerting device will be used to alert the user in case of an emergency such as system shutdown or abnormal behavior of the server resources. This device will be a cell phone module connected to a TINI device. The TINI device will be connected to the server through serial port. It will send continuous ping requests to the server. Upon not receiving a reply from the server, the cell modem will send an alert signal to the user's mobile phone via SMS. Below diagram shows an overview of the system.

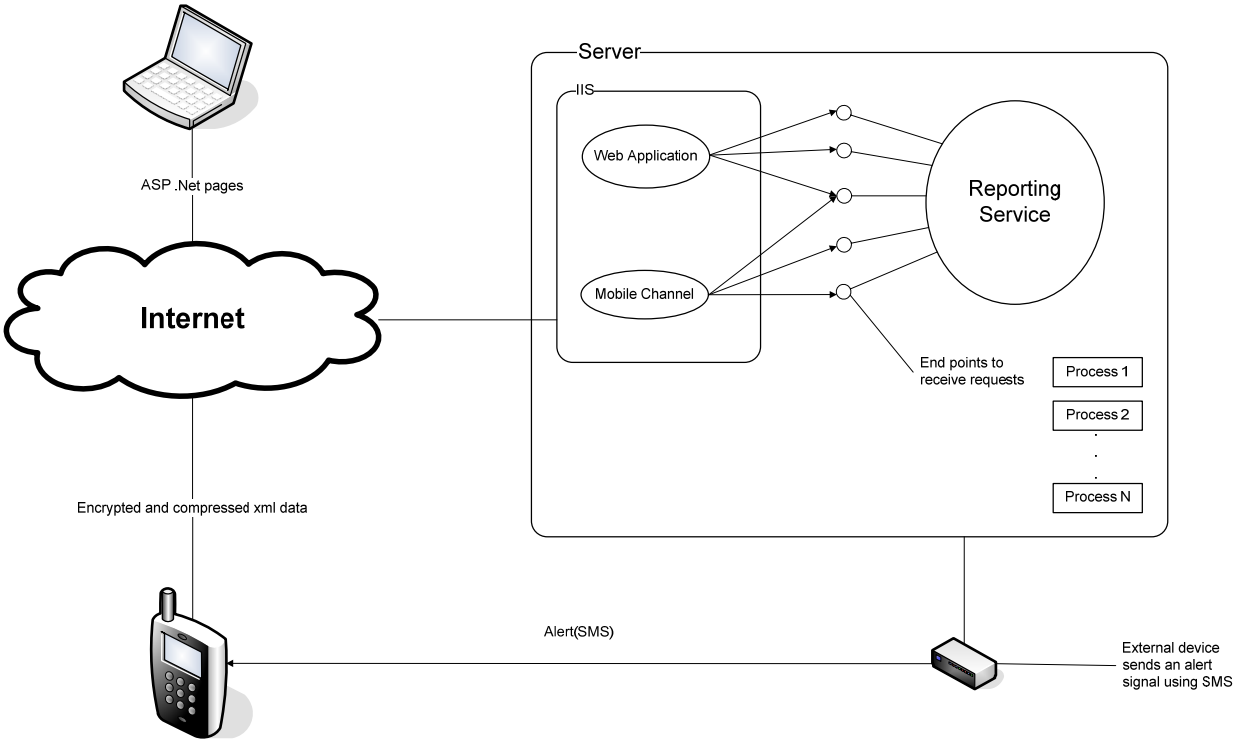


Figure 3: System Overview – Parts of the system.

4.2. System Architecture Options

4.2.1. Semi-Platform-Independent System Architecture

For this system, one of the designs sketched is for a semi-platform-independent system. The aim of this design is to make the entire system modular and semi-platform-independent. This could be achieved by developing agents that collect information about the server elements in a platform specific environment and the rest of the system in a platform-independent environment. Figure below shows an overview of the design architecture of such a system.

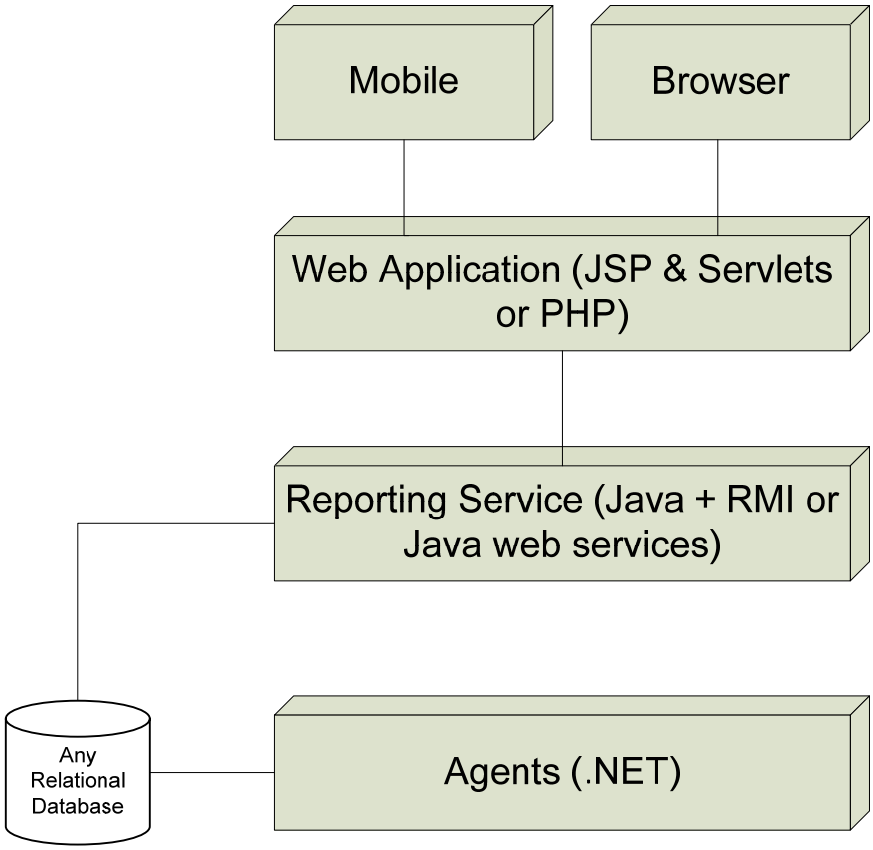


Figure 4: Semi-Platform-Independent system architecture overview.

Note: Since, the project is aimed at creating a system for a Windows-based server; the Agents are shown to be developed in .NET environment.

Here, the Agents are build in a platform specific environment (in this case .NET). This gives us the chance to get all the information from the underlying Operating System. This information then gets stored into any relational database (Codd, 1970). This database can be on the same machine or a separate

dedicated database server. Using any relational database gives the user the choice of having the database in any environment. The rest of the server modules (i.e. the reporting service & web application) can be made in a platform independent environment such a Java. Reporting service could be developed in Java and the communication layer using RMI(Remote Method Invocation Home, n.d.) or Java web services(Java Web Services At a Glance, n.d.). Similarly, the web application could be developed using Java Servlets and JSP (Java Servlet Technology, n.d.) or in PHP (PHP: Hypertext Preprocessor, n.d.) which can consume Java web services (Java Web Services At a Glance, n.d.). This will make the system semi-platform-independent. Now, we can just change the way the agents collect and store information into the database (i.e. any environment like using C, C# or ruby on Linux or Java on Solaris, etc.) and the reporting service will simply retrieve this information from the database whenever requested by the web application.

4.2.2. .NET-SPECIFIC SYSTEM ARCHITECTURE

The other design sketched for the system was a platform specific design. Here the entire system will be developed for a specific platform. Figure below shows the design architecture of such a system.

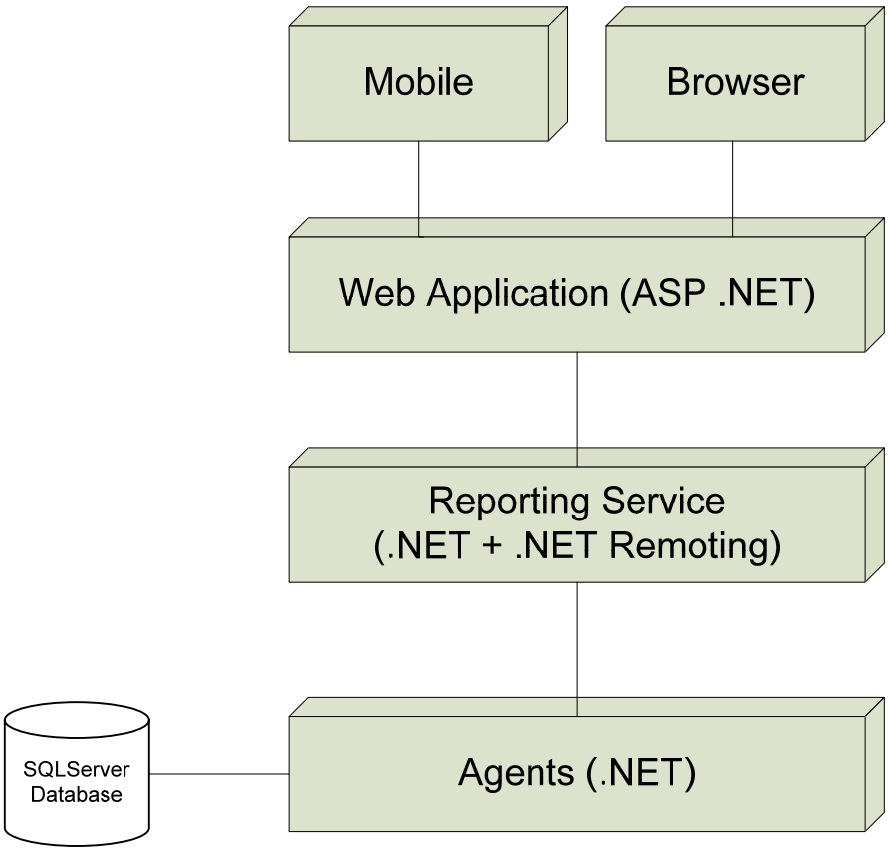


Figure 5: .NET specific system architecture overview.

Here, as we can see in the figure above, all the parts of the system will be developed for .NET platform. The agents collect the data and store it into a database server. The agents are integrated with the reporting service to carry out its operation. Now, the reporting service (being developed for the .NET platform) can fetch the data from the agents straight away without needing to go to the database at all. This makes it faster for the reporting service to get the current information about the server and provide it to the web application. This can be done using .NET remoting (.NET Remoting Overview, n.d.). The web application will have a remoting interface as well which will request information from the reporting service. Hence, developing the web application in .NET is a logical choice since remoting, via TCP channel, works faster than web services.

4.2.3. Design Architecture Choice

A choice had to be made as to which of the above two designs (or a slightly modified version of these designs) should be used to implement the system. The decision had to be based on using the already available resources. The system developed in this project was targeted towards windows-base servers and specifically for a web server as only one server is being currently monitored. This means that the server would already be running an IIS(IIS.net : HOME : Microsoft Internet Information Services, n.d.) web server. Also it would be easy to assume that they run ASP .NET web applications. So, they already have .NET installed on them. This makes it easy to decide that the .NET specific architecture is the right choice since all the resources needed are already available. Also, on the other hand, it would be rather increase the complexity of the system if a separate web server such as Apache(Welcome! - The Apache HTTP Server Project, n.d.) is needed to be installed on this server. This creates a cost of installing and maintaining multiple web servers on the same physical machine. Also, the user needs to keep track of applications on different web servers. This is not a desirable situation. Rather the user would like to have a system which does not need anything more than installing the software on the server and letting it take care of itself. Hence, we had to go ahead with the .NET specific system architecture.

4.3. Entity Relationship Model

The information that is extracted from the agents is supposed to be logged somewhere. This is data gets stored into a database. The “what and why” of logging of the data is explained later in a section below. The diagram below shows the Entity-Relationship of the data tables that are used to store data into the database. This is a complete ER diagram. The tables Processor, Hard_Drive and Process_Table store information about the each processor, hard disk[1] and process respectively. The tables Processor_Data, Hard_Drive_Data and Process_Data store timely information about the running and status of each of the

[1] If a RAID system is used for storing data, this system will see it as a single Hard Disk with an SCSI interface.

elements. Total_Memory_Data stores information regarding the memory usage of the server. It needs only single table since we do not need a reference of the memory type it is referring to. We just need to know the amount of memory available and used by the system. The details of each table are explained in the sections that describe the corresponding agents.

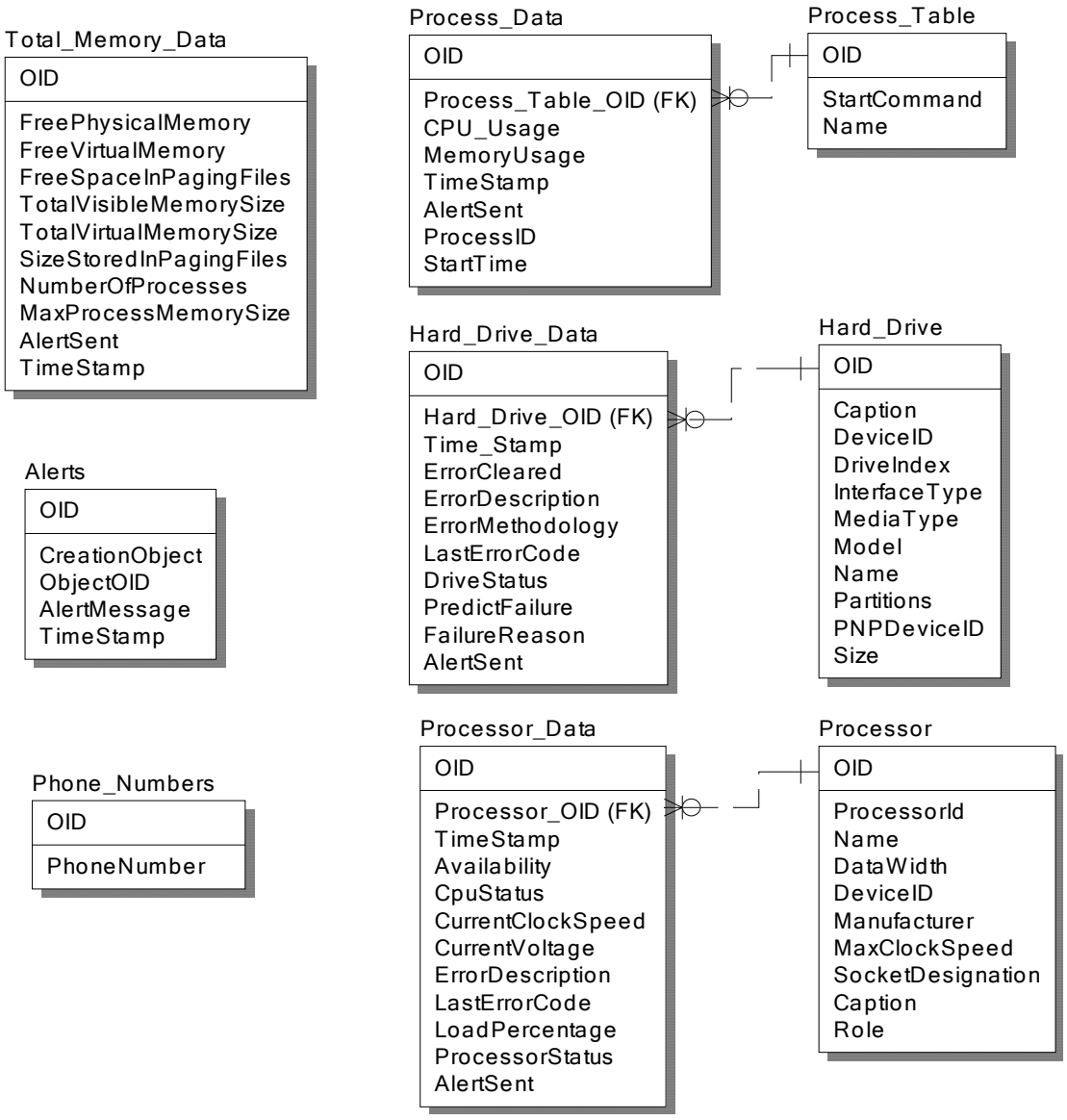


Figure 6: ER diagram for logging the data collected by the agents.

5. IMPLEMENTATION

Now, we will have a lose look at every part of the implementation phase. In the sections below we will have a look at the implementation of the agents, reporting service, mobile channel and the mobile application.

5.1. AGENTS

Agents are the components of the system that actually monitors the system. There are four agents in this system namely: Memory Agent, Processor Agent, Hard Disk Agent and the Process Agent.

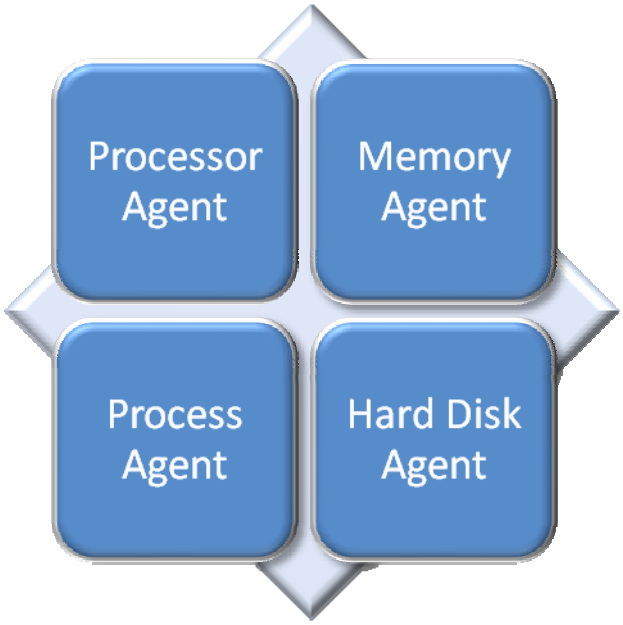


Figure 7: Types of Agents

They use a timer to periodically perform scan on the server operating system[2] to get information regarding the server. Each agent's task runs in a separate thread. They make use of WMI .NET (WMI .NET Overview, n.d.) by executing WMI queries (Querying with WQL, n.d.). These queries are executed using ManagementObjectSearcher by passing the query to the constructor and then getting a ManagementObjectCollection by calling the searcher's Get() method. Now, the collection will contain a list of ManagementObject which we go through in a loop and get the required information from it. Below is an example of such a query. These classes are available in System.Management namespace.

[2] We assume that the server runs windows operating system for the system that I developed as a part of this project. But this concept can be used on other platforms as well when a system is developed with the same architecture but different platform.

```
SelectQuery query = new SelectQuery("Select * from Win32_DiskDrive");

// Initialize an object searcher with this query
ManagementObjectSearcher searcher = new ManagementObjectSearcher(query);

// Get the resulting collection and loop through it
foreach (ManagementObject envVar in searcher.Get())
{
      HardDiskData data = new HardDiskData();
      hardDiskParameters.Caption = (string)envVar["Caption"];
      hardDiskParameters.DeviceID = (string)envVar["DeviceID"];
      .
      .
```

Code Snippet 1: Using WQL queries to gather information about the server.

Here, we can see how WQL (Querying with WQL, n.d.) can be used to get information about the server elements that are to be monitored. Now, we will look at each agent carefully.

5.1.1. Memory Agent

This agent collects information about the system's main memory. The information collected is the information about the main memory as seen by the operating system. The memory agent collects information about the physical as well as virtual memory and page files used by the server. This agent runs the following query to fetch information about the memory.

```
SelectQuery query = new SelectQuery("Select * from Win32_OperatingSystem");
```

Code Snippet 2: Query for memory agent

When this query is used in a format as shown in Code snippet 1, we can get information about the memory usage of the server. The following data is be retrieved by the memory agent:

- FreePhysicalMemory – Number, in kilobytes, of physical memory currently unused and available.

- FreeSpaceInPagingFiles – Number, in kilobytes, that can be mapped into the operating system paging files without causing any other pages to be swapped out.

- FreeVirtualMemory – Number, in kilobytes, of virtual memory currently unused and available.
- MaxProcessMemorySize – Maximum number, in kilobytes, of memory that can be allocated to a process.
- SizeStoredInPagingFiles – Total number of kilobytes that can be stored in the operating system paging files—0 (zero) indicates that there are no paging files.
- TotalVirtualMemorySize – Number, in kilobytes, of virtual memory.
- TotalVisibleMemorySize – Total amount of physical memory available to the operating system.
- NumberOfProcesses – Number of process contexts currently loaded or running on the operating system.

This information is retrieved in the following way:

```
memoryParameters.FreePhysicalMemory          = (UInt64)envVar["FreePhysicalMemory"];
memoryParameters.FreeSpaceInPagingFiles      = (UInt64)envVar["FreeSpaceInPagingFiles"];
memoryParameters.FreeVirtualMemory           = (UInt64)envVar["FreeVirtualMemory"];
memoryParameters.MaxProcessMemorySize        = (UInt64)envVar["MaxProcessMemorySize"];
memoryParameters.SizeStoredInPagingFiles     = (UInt64)envVar["SizeStoredInPagingFiles"];
memoryParameters.TotalVirtualMemory          = (UInt64)envVar["TotalVirtualMemorySize"];
memoryParameters.TotalVisibleMemory          = (UInt64)envVar["TotalVisibleMemorySize"];
memoryParameters.NumberOfProcesses           = (UInt32)envVar["NumberOfProcesses"];
```

Code snippet 3: Retrieving memory information

where envVar is the ManagementObject which contains the information retrieved after executing the query. This information is then stored into the database for logging. The table for memory data being stored is shown in the figure below. Here the database fields have meanings similar to the corresponding fields in Win32_OperatingSystem class. OID[3] is the key for the table.

[3] OID stands for Object Identifiers. Each entity in the database for this system can be treated as an object hence the name.

Figure 8: Table that stores information for the memory agent.

Any information we need about a particular aspect, can be derived mathematically. For example, percentage memory usage can be derived by

$$percentage\ memory\ usage = \frac{TotalVisibleMemorySize - FreePhysicalMemory}{TotalVisibleMemorySize} x\,100$$

Many other derivations are possible in a similar manner.

TimeStamp denotes the DateTime at which the data is collected. AlertSent represents whether an alert for this reading has been sent or not. This is usually 'No' for readings that are normal. For a reading which is exceptional (such as very high memory usage for a long period of time), an alert is sent by the system to user's phone through the external device. This is represented as 'Yes' in this field.

5.1.2. Processor Agent

This agent collects information regarding each processors of the server that are currently visible to the operating system. The query for getting the data for a processor is shown below.

```
SelectQuery query = new SelectQuery("Select * from Win32_Processor");
```

Code Snippet 4: Query to retrieve processor information

After this query is executed just like the one in the code snippet 1, we can get the information regarding each processor. Following information about a processor is retrieved using the above query:

- Name – Label by which the object is known.
- Caption – Short description of an object (a one-line string).
- DataWidth – Processor data width, in bits.
- DeviceID – Unique identifier of a processor on the system.
- Manufacturer – Name of the processor manufacturer.
- MaxClockSpeed – Maximum speed of the processor, in MHz.
- ProcessorID – Processor information that describes the processor features.
- Role - Role of the processor. Examples: Central Processor or Math Processor.
- SocketDesignation - Type of chip socket used on the circuit.

This information is retrieved in the following way.

```
processorParameters.Name = ((string)envVar["Name"]).Trim();
processorParameters.Caption = (string)envVar["Caption"];
processorParameters.DataWidth = (UInt16)envVar["DataWidth"];
processorParameters.DeviceID = (string)envVar["DeviceID"];
processorParameters.Manufacturer = (string)envVar["Manufacturer"];
processorParameters.MaxClockSpeed = (UInt32)envVar["MaxClockSpeed"];
processorParameters.ProcessorID = (string)envVar["ProcessorID"];
processorParameters.Role = (string)envVar["Role"];
processorParameters.SocketDesignation = (string)envVar["SocketDesignation"];
```

Code snippet 5: Retrieving Processor information

where envVar is the ManagementObject which contains the information retrieved after executing the query.

Now, the information regarding the running of the processor needs to be collected. This can be collected at the same time when the above information is collected as this information can be retrieved

straight from Win32_Processor class itself. Following information needs to be collected about the running of the processor needs to be collected:

- Availability – Availability and status of the device.
- CpuStatus – Current status of the processor. Status changes indicate processor usage, but not the physical condition of the processor.
- CurrentClockSpeed – Current speed of the processor, in MHz.
- CurrentVoltage – Voltage of the processor.
- ErrorDescription
- LastErrorCode – Last error code reported by the logical device.
- LoadPercentage – Load capacity of each processor, averaged to the last second. Processor loading refers to the total computing burden for each processor at one time.
- Status – Current status of an object.

This information can be collected in the following manner from the envVar variable used in the above code snippet:

```
processorDataParameters.Availability = (UInt16)envVar["Availability"];
processorDataParameters.CPUStatus = (UInt16)envVar["CpuStatus"];
processorDataParameters.CurrentClockSpeed = (UInt32)envVar["CurrentClockSpeed"];
processorDataParameters.CurrentVoltage = (UInt16)envVar["CurrentVoltage"];
processorDataParameters.ErrorDescription = (string)envVar["ErrorDescription"];
processorDataParameters.LastErrorCode = (UInt32)envVar["LastErrorCode"];
processorDataParameters.LoadPercentage = (UInt16)envVar["LoadPercentage"];
processorDataParameters.ProcessorStatus = (string)envVar["Status"];
```

Code snippet 6: Collecting information about the running of the processor.

This information is then stored into the database for logging. The table for processor data being stored is shown in the figure below. Here the database fields have meanings similar to the corresponding fields in Win32_Processor class. OID is the key for the table. Processor_OID is the foreign key in Processor_Data table from Processor table.

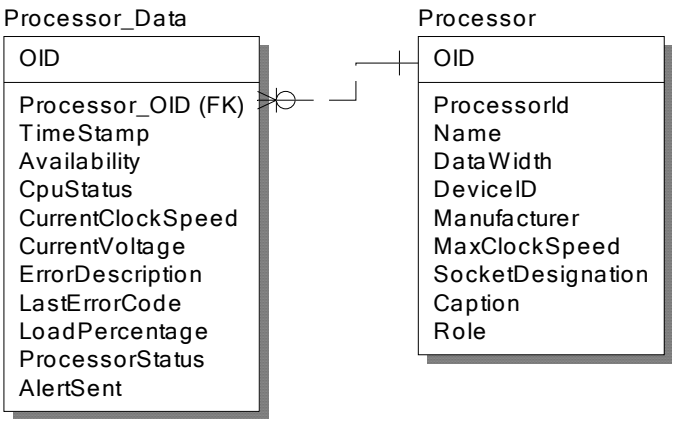


Figure 9: Table that stores information for Processor Agent.

The description of TimeStamp and AlertSent is the same a described in section 5.1.1.

5.1.3. Hard Disk Agent

This agent collects information regarding each hard disk drive of the server that is currently visible to the operating system. Information regarding a hard disk drive and its status is retrieved separately. First the information for a hard disk drive is retrieved. The query for getting the data for every disk drive is shown below.

```
SelectQuery query = new SelectQuery("Select * from Win32_DiskDrive");
```

Code Snippet 7: Query to retrieve Hard Drive information

After this query is executed just like the one in the code snippet 1, we can get the information regarding each disk drive on the server. Following information about the disk drives is retrieved using the above query:

- Caption – Short description (one-line string) of the object.
- DeviceID – Unique identifier of the disk drive with other devices on the system.
- Index – Physical drive number of the given drive.
- InterfaceType – Interface type of physical disk drive. Example – SCSI, IDE, USB, etc.
- MediaType – Type of media used or accessed by this device.

- Model – Manufacturer's model number of the disk drive. Example: "ST32171W"
- Name – Label by which the object is known.
- Partitions – Number of partitions on this physical disk drive that are recognized by the operating system.
- PNPDeviceID – Windows Plug and Play device identifier of the logical device.
- Size – Size of the disk drive. It is calculated by multiplying the total number of cylinders, tracks in each cylinder, sectors in each track, and bytes in each sector.

This information is retrieved in the following way.

```
hardDiskParameters.Caption = (string)envVar["Caption"];
hardDiskParameters.DeviceID = (string)envVar["DeviceID"];
hardDiskParameters.DriveIndex = (uint)envVar["Index"];
hardDiskParameters.InterfaceType = (string)envVar["InterfaceType"];
hardDiskParameters.MediaType = (string)envVar["MediaType"];
hardDiskParameters.Model = (string)envVar["Model"];
hardDiskParameters.Name = (string)envVar["Name"];
hardDiskParameters.Partitions = (uint)envVar["Partitions"];
hardDiskParameters.PNPDeviceID = (string)envVar["PNPDeviceID"];
hardDiskParameters.Size = (UInt64)envVar["Size"];
```

Code snippet 8: Retrieving disk drive information

where envVar is the ManagementObject which contains the information retrieved after executing the query.

Now, the information regarding the status of the disk drives needs to be collected. Most of the information can be collected at the same time when the above information is collected as this information can be retrieved straight from Win32_DiskDrive class itself. Following information needs to be collected about the status of the disk drives needs to be collected:

- ErrorCleared – If **True**, the error reported in **LastErrorCode** is now cleared.

- ErrorDescription – More information about the error recorded in **LastErrorCode**, and information on any corrective actions that may be taken.
- ErrorMethodology – Type of error detection and correction supported by this device.
- LastErrorCode – Last error code reported by the logical device.
- Status – Current status of the object. Values are: OK, Error, Degraded, Unknown, Pred Fail, Starting, Stopping, Service, Stressed, NonRecover, No Contact, Lost Comm.

This information can be collected in the following manner from the envVar variable used in the above code snippet:

```
hardDiskDataParameters.ErrorCleared = ((bool)envVar["ErrorCleared"]).ToString();
hardDiskDataParameters.ErrorDescription = (string)envVar["ErrorDescription"];
hardDiskDataParameters.ErrorMethodology = (string)envVar["ErrorMethodology"];
hardDiskDataParameters.LastErrorCode = (uint)envVar["LastErrorCode"];
hardDiskDataParameters.Status = (string)envVar["Status"];
```

Code snippet 9: Collecting information about the status of the disk drives

Now we need to extract information regarding the S.M.A.R.T.[4] indicators that give information about predicting hard drive failures, we need to run a separate query. The query shown below gets this information.

```
SelectQuery query2 = new SelectQuery("select * from MSStorageDriver_FailurePredictStatus where”
                        + "InstanceName = '" + hardDiskParameters.PNPDeviceID + "'");
```

Code snippet 10: Query to collect information about the S.M.A.R.T. status of a disk drive

Now the information can be collected using the “PredictFailure” and “Reason” fields of MSStorageDriver_FailurePredictStatus class. This information is then stored into the database for logging. The table for disk drive’s data being stored is shown in the figure below. Here the database fields have meanings similar to the corresponding fields in Win32_DiskDrive and MSStorageDriver_FailurePredictStatus classes. OID is the key for the table. Hard_Drive_OID is the foreign key in Hard_Drive_Data table from Hard_Drive table.

[4] Self-Monitoring, Analysis, and Reporting Technology, or S.M.A.R.T., is a monitoring system for computer hard disks to detect and report on various indicators of reliability, in the hope of anticipating failures.

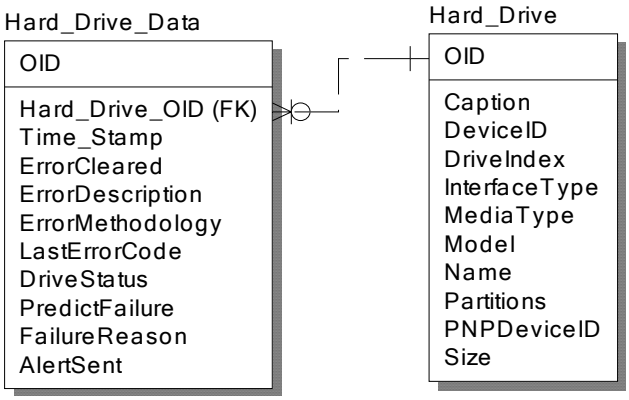


Figure 10: Table that stores information for Hard Disk Agent Agent.

The description of TimeStamp and AlertSent is the same a described in section 5.1.1.

5.1.4. PROCESS AGENT

This agent collects information regarding all the processes running on the server that are currently visible to the operating system. The query for getting the data for processes is shown below.

```
SelectQuery query = new SelectQuery("Select * from Win32_Process");
```

Code Snippet 11: Query to retrieve processor information

After this query is executed just like the one in the code snippet 1, we can get the information regarding each process. Following information about a processor is retrieved using the above query:

- Name – Label for an object.
- CommandLine – Command line used to start a specific process, if applicable.
- ProcessID – Global process identifier that is used to identify a process. ProcessID is not a part of the process's basic information such as the name or the command line etc. but it is included here to get information about the performance counters of a process.
- CreationDate – Date the process begins executing. This information is not a process's information either but it is included here since this information is available only in Win32_Process class and not from the performance counters classes.

This information is retrieved in the following way.

```
obj.Name = (string)envVar["Name"];
obj.CommandLine = (string)envVar["CommandLine"];
obj.ProcessID = (uint)envVar["ProcessID"];

int index = ((string)envVar["CreationDate"]).IndexOf(".");
string dateTempStr = ((string)envVar["CreationDate"]).Substring(0, index);
obj.CreationDate = DateTime.ParseExact(dateTempStr, "yyyyMMddHHmmss",
                        DateTimeFormatInfo.InvariantInfo);
```

Code snippet 12: Retrieving Process' information

where envVar is the ManagementObject which contains the information retrieved after executing the query.

Now, the information regarding the running of the process needs to be collected. Calculating the processor usage of each process was arguable the most difficult part of the project for me. WMI does not have any way to give the processor usage of each process. Neither does PerformanceCounter class give useful information on every run. The values returned by these measures are either zero or hundred. WMI has two classes to give this information, namely, Win32_PerfRawData_PerfProc_Process and Win32_PerfFormattedData_PerfProc_Process. The latter is supposed to provide information about the processes on the server by calculating the values from the first. But it doesn't give the right information. It just indicates if the process was using a processor on the cycle that the request was made or not. Hence this information had to be calculated manually. To do this, we need to get the amount of processor time that the process has already used and then after 500 milliseconds get the amount of the processor time the process has used. Get the difference between the processor times. Then divide it by the time that has already passed since the first snapshot was taken. Now divide it again by the number of processors. This gives us the percent processor usage by that process. Also, if we do this separately for each process, this takes up a lot of processor. Hence we store the information for all the processes in HashTable. Then we take the snapshots and then calculate the processor usage for all the processes at the same time. This slows down this process significantly. A change of processor usage from 45% to 5% was encountered due to this change. Following is the pseudo code that shows this in working:

```
SelectQuery query = new SelectQuery("Select * from Win32_PerfRawData_PerfProc_Process");
ManagementObjectSearcher searcher = new ManagementObjectSearcher(query);

foreach (ManagementObject envVar in searcher.Get())
{
        if ((string)envVar["Name"] != "_Total" && (string)envVar["Name"] != "Idle")
        {
                ProcessPerfObj perfObj = new ProcessPerfObj();
                perfObj.ProcessID = (uint)envVar["IDProcess"];
                perfObj.ProcessorTimeOne = (ulong)envVar["PercentProcessorTime"];
                perfObj.SystemTime100NSOne = (ulong)envVar["Timestamp_Sys100NS"];
                processPerfDictionary.Add(perfObj.ProcessID, perfObj);
        }
}

Thread.Sleep(500);

foreach (ManagementObject envVar in searcher.Get())
{
        if (processNames.ContainsKey((uint)envVar["IDProcess"]))
        {
                uint tempVar = (uint)envVar["IDProcess"];

                ProcessPerfObj perfObj = (ProcessPerfObj)processPerfDictionary[tempVar];
                perfObj.ProcessorTimeTwo = (ulong)envVar["PercentProcessorTime"];
                perfObj.SystemTime100NSTwo = (ulong)envVar["Timestamp_Sys100NS"];
                perfObj.MemoryUsage = (ulong)envVar["WorkingSet"];
        }
}
//Calculate the difference for each process
//(((double)(ProcessorTimeTwo - ProcessorTimeOne)) / (((double)(SystemTime100NSTwo -
SystemTime100NSOne)) * 2) * 100.0);
```

Code snippet 13: Collecting information about the running of all the processes.

This information is then stored into the database for logging. The table for processor data being stored is shown in the figure below. Here the database fields have meanings similar to the corresponding fields in Win32_Process class. OID is the key for the table. Process_OID is the foreign key in Process_Data table from Process_Table (Process is a keyword so we have call it something else) table.

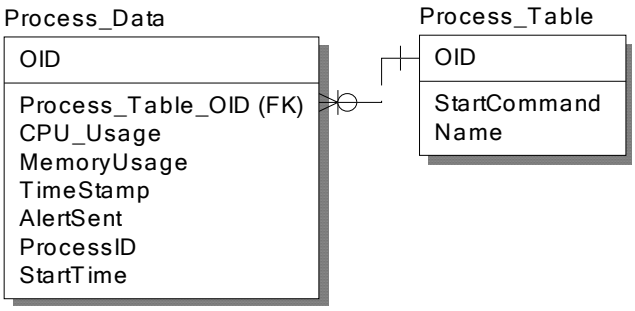


Figure 11: Table that stores information for Processor Agent.

The description of TimeStamp and AlertSent is the same a described in section 5.1.1.

In addition to collecting information from the operating system, the Process agent also provides three functionalities. Firstly to run a new command, second is to kill a process and third is to restart the server. This is done by through the following three methods.

```
public string RunCommand(string Command);
public void killProcess(int ProcessID);
public void restartMachine();
```

Code snippet 14: Methods to run and kill a process

5.1.5. LOGGING

The information that is collected by the agents has to be logged somewhere. There were several options in hand for this case. Following is the list of few

- Embedded database
- Isolated storage
- XML file
- Relational Database

All these options were looked at closely and the following advantage and disadvantages were found:

Embedded database is a very fast way to store the information and retrieve it while the application is running. It gives the speed that we get while dealing with an in-process storage, yet we get persistent data. The problem with embedded database is that the information cannot be stored at a central location which can be accessed in case of major breakdown[5]. Same is the problem with isolated storages. Isolated storages store data into the same system that the application is running on and hence, hard to retrieve the information in case of a major breakdown.

XML files are a good way to store structured data. XML can be stored anywhere on any system. But the problem with this approach is that there are more writes than reads. This makes a writing of the log data very slow. Also the entire data needs to be processed every time something needs to be added to the file. This brings up a huge overhead.

Relational databases provide a very good way to persist data in a central location. They are already scalable and reliable. Also the processing of certain information such as averages and maximums can be done on a database server itself. The only problem is a increase in network traffic but this system does

[5] Major breakdown such as hard drive crash or a similar situation where all the data is lost.

not need to send or receive a huge amount of data. This make relational databases the best way to log data for this system. Hence, a relational database was chosen for logging of the data.

5.1.6. Fetching Current Information

All agents have a method call ToXmlString(bool isForMobile). This method gets the information for each agent in xml string format. For this, all the data that is fetched by the agents from the operating system needs to be serialized into xml format. A natural choice was to use XmlSerializer class from System.Xml.Serialization namespace. But here we cannot refine the way we can serialize the way want the data to be serialized. For example, we cannot ask the serializer to serialize only the attributes for the mobile application and not the once that are needed for the ASP .NET application dynamically. We can only mark it with an XmlIgnore attribute. But this will completely ignore the property which is undesired. So I created a new attribute called CustomSerializableAttribute. It has a property called propertyFor. We can assign with property, a value to indicate what the property is for. For example, [CustomSerializable(propertyFor = "mobile")] indicates that the property that is decorated with this attribute has to be serialzed only when the data needed is for mobile. After that, I created a CustomXmlSerializer class that serializes the data. It has method called serialize which uses reflection to get all the properties of an object and check the propertyFor attribute if it is assigned any value and then checks for what types of the properties are needed and accordingly serializes the data. The following code snippet shows these in action.

```
public string ToXmlString2(bool isForMobile)
{
        CustomXmlSerializer serializer = new CustomXmlSerializer(typeof(MemoryData));

         MemoryStream s = new MemoryStream();
         XmlTextWriter writer = new XmlTextWriter(s, Encoding.ASCII);
         writer.Formatting = Formatting.None;
         String xmlString = "";
         writer.WriteStartDocument();
         writer.WriteStartElement("Memory");
         if (taskPerformer.Data.FreePhysicalMemory != 0)
         {
                serializer.serialize(writer, taskPerformer.Data, isForMobile);
        }
         writer.WriteEndElement();
         writer.WriteEndDocument();
         writer.Flush();

         s = (MemoryStream)writer.BaseStream;
        xmlString = new ASCIIEncoding().GetString(s.ToArray());

        return xmlString;
}
```

Code snippet 15: CustomSerializableAttribute and CustomXmlSerializer in action

5.2. REPORTING SERVICE

Reporting service is the component that runs the agents on the server. It is responsible for controlling the agents. A dynamic link library (dll) file is created for the agent's assembly. This assembly is used by the reporting service to run these agents. It is also responsible for making the functionalities of the agents such as providing the information about the server elements, running a process, killing a process and restarting the server available for the web application to request and process it. This is accomplished by exposing the service through a .NET remoting (.NET Remoting Overview, n.d.) interface. This allows for running this service on any server on a network and making it accessible to the web application hosted on a web server but doing this out of the scope of this project. This has been done just provide future extensibility. It is also responsible for notifying the external device such as TINI that server is still responsive.

5.2.1. GUI

A GUI for the reporting service was created to enable the server administrator to configure the server watch system. The GUI is separated from the reporting service utility class. The ReportingServiceClass class provides the functionalities that the reporting service is supposed to perform. All the agents are started and stopped by this utility class. However, the parameters that are to be passed to the agents are to be set through the GUI. This allows changing the internal working of the service and keeping the GUI unaffected and vice versa. The GUI form ReportingServiceForm can set values for:

- Scan interval – The interval at which agents will scan the system.
- Phone numbers – Phone numbers to send the alert message to during critical situations.
- Maximum memory usage – The maximum level at which the system memory is allowed to reach to qualify as normal.
- Maximum processor usage – The maximum level at which the processor usage in percentage is allowed to reach to qualify as normal.
- Maximum memory and processor usage - The maximum level at which the system memory and processor usage in percentage is allowed to reach to qualify as normal.
- Database settings – The location and authentication details for the database.

The GUI form also provides functionalities to start, stop or restart the service. This will infect call the startService and stopService methods of the ReportingServiceClass class. Below are a few screen shots of the GUI of the reporting service.

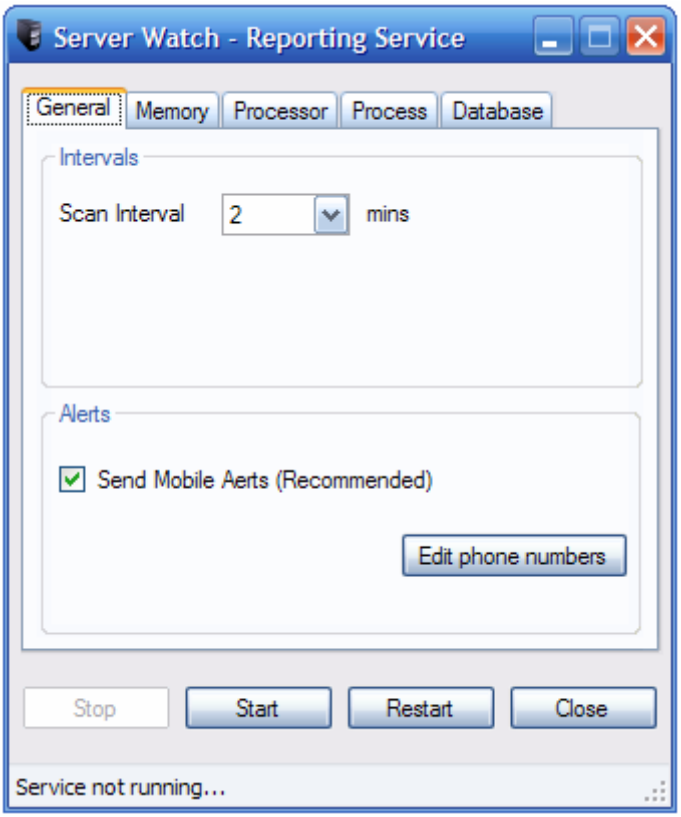


Figure 12: GUI form for reporting service

As you can seen in figure 12, the GUI looks like a configuration panel. Here all the agents have a tab of their own to set their parameters. The first tab is the general tab. Here, we can set the scan interval and phone numbers. You can add or delete phone numbers. Also, select up to three phone numbers where the alert can be sent. Figure 13 shows a screen shot of the phone number dialogue box. Also, in figure 10 we can see the buttons for starting stopping and restarting the service. The “Close” button will just put the reporting service GUI into the taskbar (figure 14).

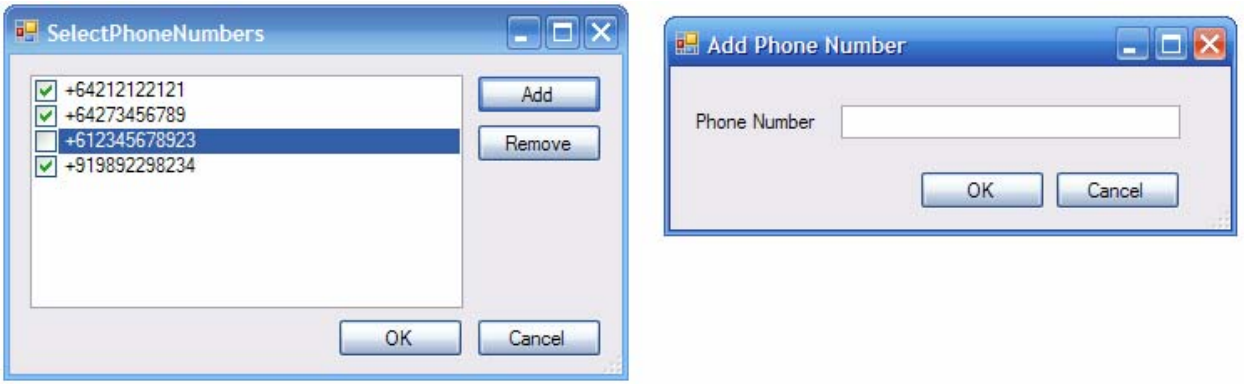


Figure 13: Handling phone numbers to send alert messages

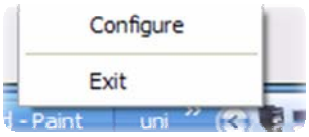
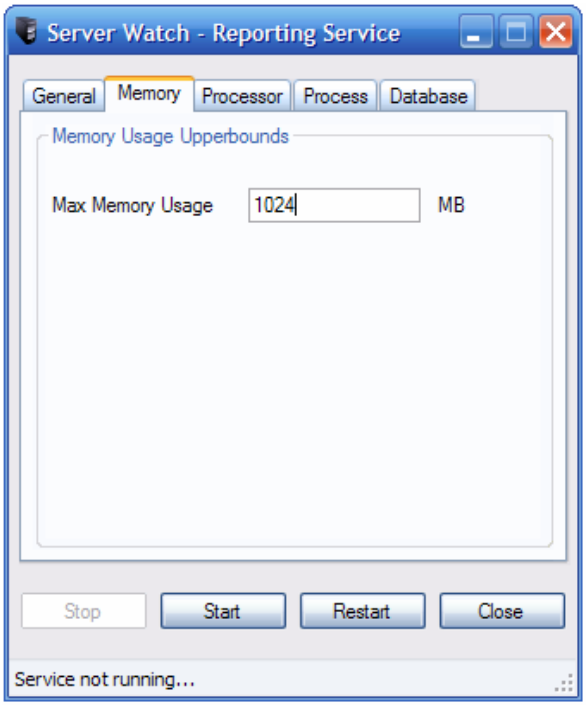


Figure 14: Minimized reporting service

The figures below show the screen shots of setting the parameters for the agents.



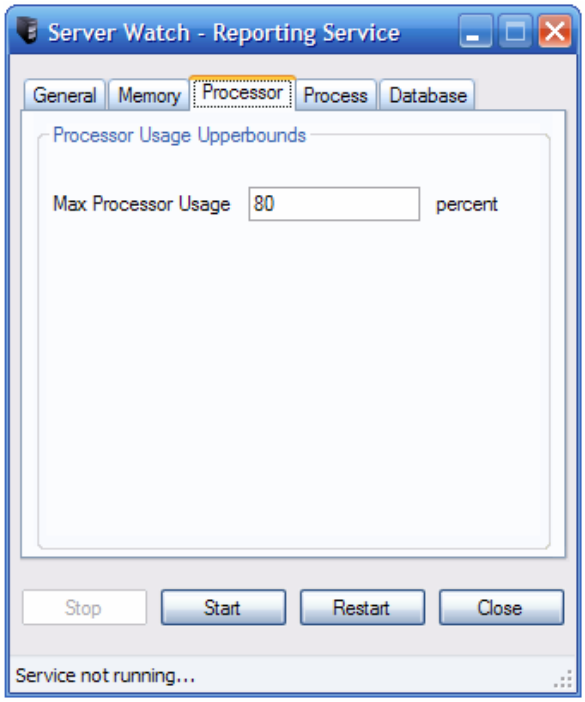
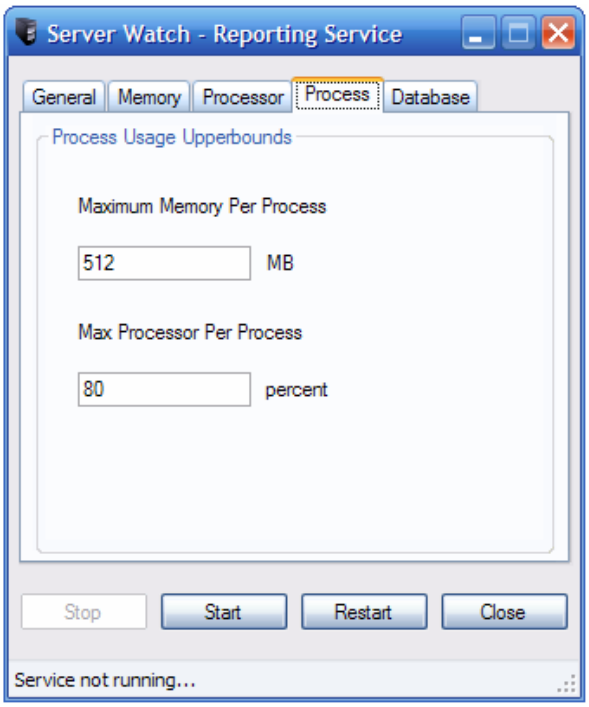


Figure 15 and 16: Configuring maximum memory usage and processor usage



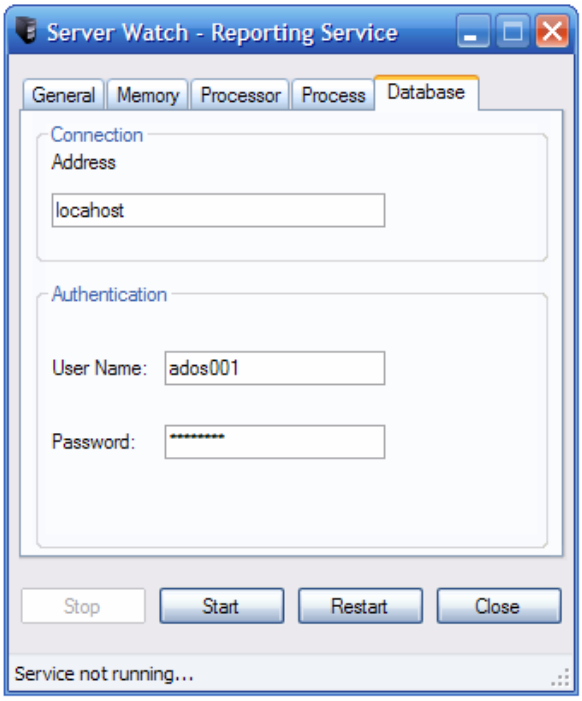
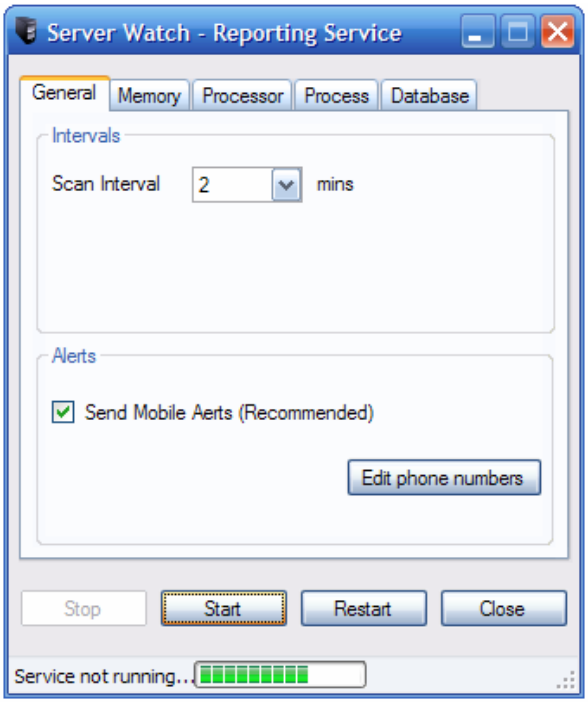


Figure 17 and 18: Configuring the Max memory and CPU usage per process and database settings



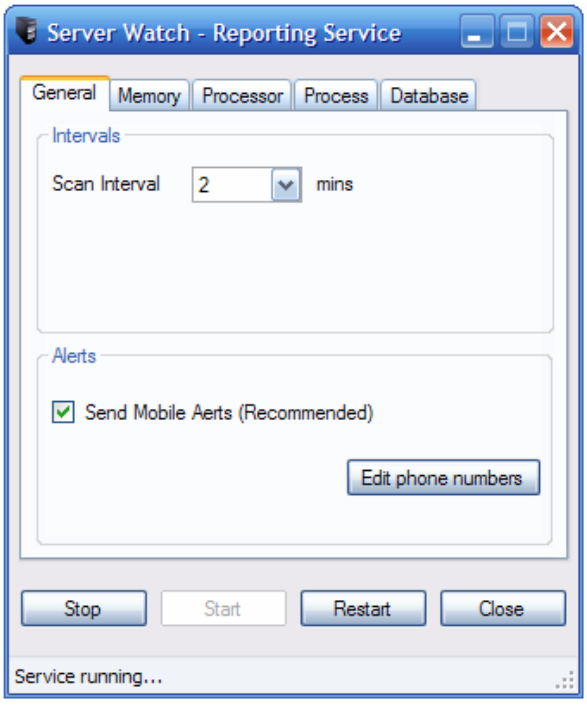


Figure 19 and 20: While and after starting the service

5.2.2. REMOTING SERVICE

The Reporting service also contains a .NET remoting service which enables the web application to communicate with the reporting service. The RemotingObject contains a reference to all the agents. It contains methods to get the data from agents by calling their ToXmlString method. The RemotingServiceProvider class exposes this object as we remoting object. It opens an HTTP channel to receive requests. It publishes an already created instance of the RemotingObject. An URL is assigned to this object which looks like http://<IP Aaddress>:45454/RemoteService.rem. Now we can call methods on this object by creating .NET remoting client. A part of the WSDL of this service is shown below.

```
- <definitions name="RemotingObject"
  targetNamespace="http://schemas.microsoft.com/clr/nsassem/ServerWat
  - <message name="RemotingObject.getProcessorDataInput">
      <part name="isForMobile" type="xsd:boolean"/>
    </message>
  - <message name="RemotingObject.getProcessorDataOutput">
      <part name="return" type="xsd:string"/>
    </message>
  - <message name="RemotingObject.getProcessDataInput">
      <part name="isForMobile" type="xsd:boolean"/>
    </message>
  - <message name="RemotingObject.getProcessDataOutput">
      <part name="return" type="xsd:string"/>
    </message>
  - <message name="RemotingObject.getMemoryDataInput">
      <part name="isForMobile" type="xsd:boolean"/>
    </message>
  - <message name="RemotingObject.getMemoryDataOutput">
      <part name="return" type="xsd:string"/>
    </message>
```

Figure 21: WSDL for the RemotingObject

5.2.3. POLLING

Polling is used to indicate the external device that the server is still responsive. When the external device stops receiving the polling message, it sends an alert SMS to the user saying that the server is not responding.

The TINI runs a HTTP listener that listens to the requests. When is receives a request, it checks for the data inside it to see which server has sent the polling message. Then it stores that server into responsive category. If it doesn't receive a request from certain server for 5 minutes, the server qualifies as non-responsive. An alert message is sent to the user via SMS. This part of the system was developed by Tom at MCS.

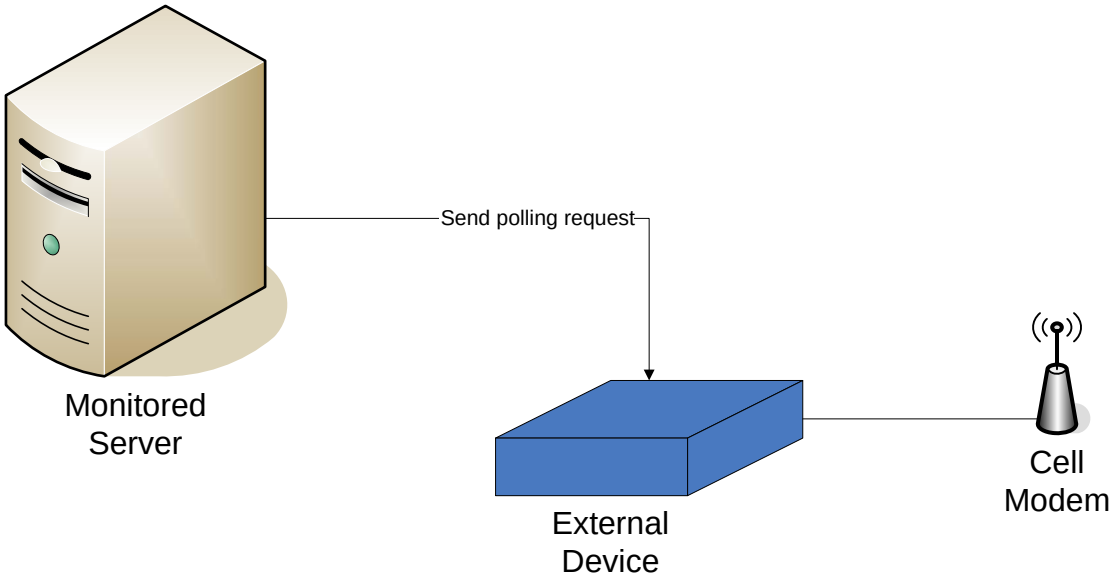


Figure 22: The external device receives polling requests when the server is responsive

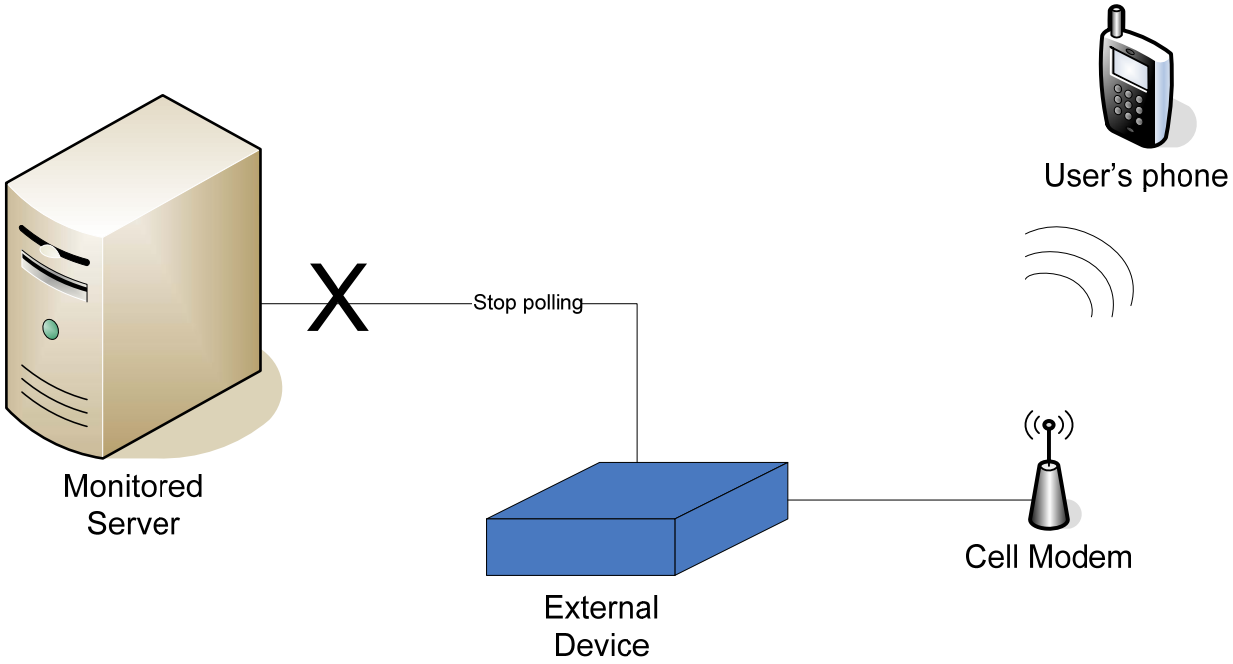


Figure 23: SMS sent when the polling device stops receiving requests

5.2.4. Fetching the history information

In the later stages of the project, it was decided to display the history of average usages for memory and processor as well as the number of processes running on the server for past 24 hours graphically on the mobile application. So this had to be done on the server-side. So it was decided to perform this operation in the reporting service so as each reporting service can fetch history information logged by its own agents. The RemoteObject contains the methods that execute a query to fetch history information and serializes it into an xml string. The query looks like the following code snippet.

```
SELECT AVG(pd.NumberOfProcesses)
from Total_Memory_Data pd
WHERE pd.TimeStamp < @beforeTime
AND pd.TimeStamp > @afterTime
```

Code snippet 16: Query to fetch history data

All the data that is fetched as a result of executing the query is then serialized into a xml string and passed to the requesting web application.

5.3. Mobile Channel

Mobile channel is an intermediary component that handles requests from the mobile application and requests this information from the corresponding reporting service and sends back the response to the requesting mobile application. It is hosted on an IIS web server.

5.3.1. Handlers

The mobile channel is composed of ASP .NET generic handlers. These handlers have a method called “ProcessRequest”. This method is responsible for handling the requests. These handlers process the request and ask the remoting client to call methods on the reporting service. Then it processes this information and responds to the mobile client accordingly.

5.3.2. REMOTING SERVICE CONSUMER

Mobile channel communicates with the reporting service through a .NET remoting client. There is a RemotingServiceConsumer class on this mobile channel which is responsible for the communication between the reporting service and the mobile channel. The handler processes the request and then calls the methods accordingly on the RemotingObject present on the reporting service that is published by the reporting service.

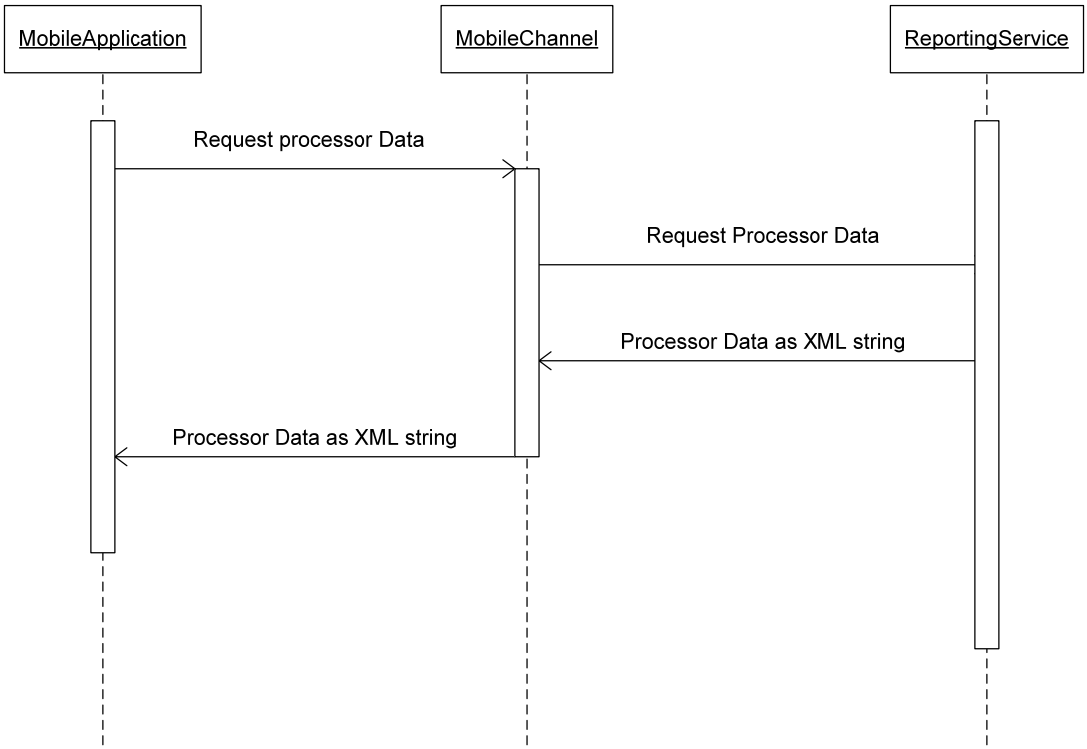


Figure 24: Flow of requests and responses

Figure 24 summarizes the flow of data between these components.

5.4 Mobile Application

A mobile application will be typically installed on the server administrator's mobile phone. The aim for this component was to develop a system has a low response time as well as professional user interface.

5.4.1. J2ME Polish To The Rescue

J2ME applications build for CLDC 1.0 and MIDP 2.0 usually do not have a very good user interface unless a lot of work has been put into it. This makes the development process slow. This makes us look for an alternative way to develop using a tool which can do the complex work for us and we do not have to go an extra mile to create a good user interface. J2ME Polish does this for us. Creating a good interface with desired layouts and colors becomes a trivial task when J2ME Polish is used. It allows the developer to concentrate on the parts that need more attention and it takes care of the interface. All the developer needs to do is create a css (cascading style sheet) file with the styles that we desire and add the **//#style <styleID>** directive on top of the creation of the GUI component and Polish takes care of the rest. Below is an example of a J2ME application build with and without Polish.

Here, is the code and screen shot of a J2ME application without using Polish.

```
private List get_MainScreen()    {
    if(MainScreen == null){

        MainScreen = new List("Main Screen",List.IMPLICIT);
        try {
            MainScreen.append("Processor", Image.createImage("/processor.png"));
            MainScreen.append("Process", Image.createImage("/process.png"));
            MainScreen.append("Memory",Image.createImage("/memory.png"));
            MainScreen.append("Hard Disk",Image.createImage("/harddisk.png"));
            MainScreen.append("Restart",Image.createImage("/restart.png"));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        MainScreen.addCommand(get_exitCommand());
    }
    return MainScreen;
}
```

Figure 25: J2ME Code without Polish

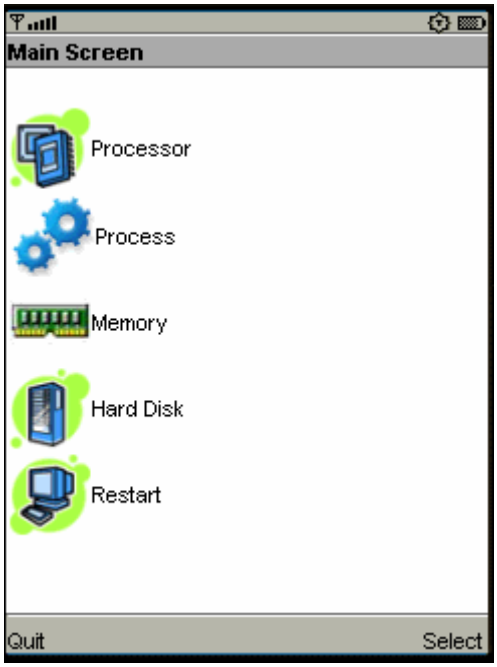


Figure 26: J2ME application screen without Polish

Below are the J2ME and Polish CSS code and the screen shot of an application screen using J2ME Polish.

```
private List get_MainScreen()   {
    if(MainScreen == null){
        //#style mainMenu
        MainScreen = new List("Main Screen",List.IMPLICIT);
        try {
            //#style mainMenuItem
            MainScreen.append("Processor", Image.createImage("/processor.png"));
            //#style mainMenuItem
            MainScreen.append("Process", Image.createImage("/process.png"));
            //#style mainMenuItem
            MainScreen.append("Memory",Image.createImage("/memory.png"));
            //#style mainMenuItem
            MainScreen.append("Hard Disk",Image.createImage("/harddisk.png"));
            //#style mainMenuItem
            MainScreen.append("Restart",Image.createImage("/restart.png"));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        MainScreen.addCommand(get_exitCommand());
        this.MainScreen.setCommandListener(this);
    }
    return MainScreen;
}
```

Figure 27: J2ME Code with Polish

```
.mainMenu   {
        padding: 5;
        padding-left: 5;
        padding-right: 5;
        background-color: bgColor;
        border-color: highLightedBgColor;
        layout: horizontal-center | vertical-center;
        columns: 2;
        columns-width: equal;
}
.mainMenuItem   {
        margin-left: 2;
        margin-right: 2;
        padding: 2;
        font-color: fontColor;
        font-size: medium;
        layout: center;
        icon-image-align: top;
        focused-style: .mainMenuItemFocused;
}
focused {
        border  {
            type: round-rect;
            arc: 10;
            color: silver;
            border-width: 2;
        }
}
```

focused is called when an item is focused. I applied to the entire MIDlet.

Figure 28 and 29: Polish CSS code and J2ME application screen with Polish

Here we can clearly see how much of a difference does J2ME Polish makes. The difference is phenomenal when ease of development is added to the list of advantages of using J2ME Polish which already includes good interface design.

5.4.2. SCREEN SHOTS AND WORKING

The mobile application is built on J2ME platform. This has a lot of benefit including platform-independence. As a result, the application can be used on a very wide variety of mobile phones as well as other devices. But it has its own limitations. There are no Panels or Frames in J2ME as we have in Java SE. Also, the numbers of GUI components that can be used to create a form are limited. Developing a mobile application in J2ME can be a very promising task. But the benefits of using this platform easily outweigh its challenging behavior.

Every J2ME application is called a MIDlet when it is developed for MIDP profiles. Every class that displays UI components on the screen needs to extend MIDlet class. For creating windows or forms in MIDP, there are two components that can be used. A form can be either a Form or a List. A Form can have different types components such as a StringItem, Gauge, TextField, etc. But a List can only be composed of a list of strings (can have image icons for these strings). It represents a list of options to choose from. Both, Form and List, have command listeners. We need to pass in a CommandListener object to the setCommandListener method. A CommandListener object has a CommandAction method in it to process this command.

In this project I have created only one MIDlet for the mobile application. This MIDlet will enable administrator to log into the mobile application and receive latest information regarding the server it needs to monitor. Also, the administrator can send command action such as kill a process etc. The mobile application is Polish-enabled. It uses Polish to help design the user interface.

The network communication in J2ME has to take place in a separate thread. When we make a HTTP request, a separate thread needs to be created to make such a request. I have created a class called MobileHttpConnector which takes care of the generating a HTTP request and a method called performResponseTask takes care of processing the response. If we want that the user do not press any other key while a request is being made, we need to replace the display screen. This avoids making multiple requests at the same time. This makes our application sequential. After getting the required data, its xml is parsed using the kxml2-lite library and appropriate screen is loaded. Below are the screen shots of the mobile application in action.

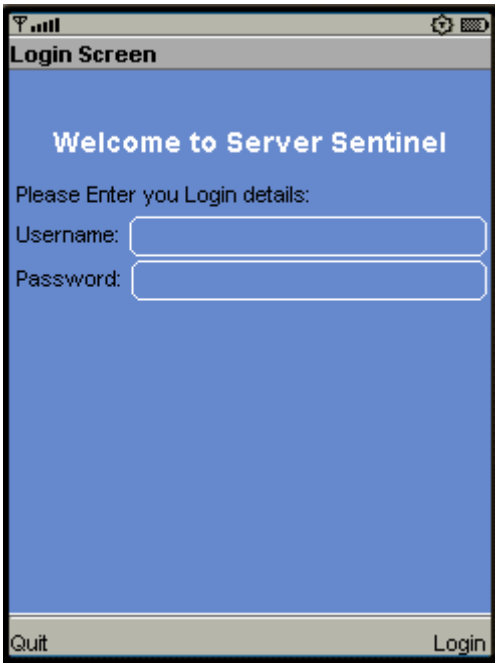


Figure30: Login Screen

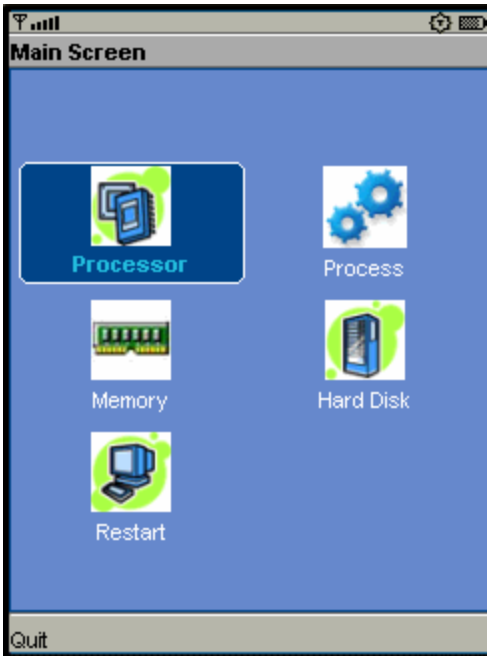


Figure 31: Main Screen

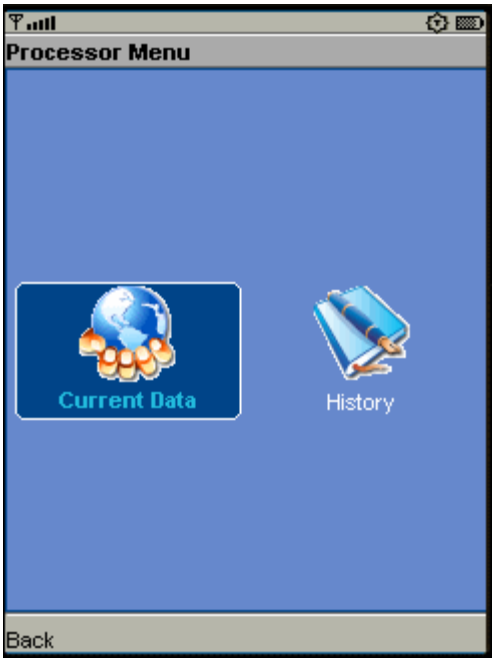


Figure 33: Menus for Processor, Hard Disk and Memory

Figure 33: Process Menu Screen

Figure 34: Current Processor Data Screen

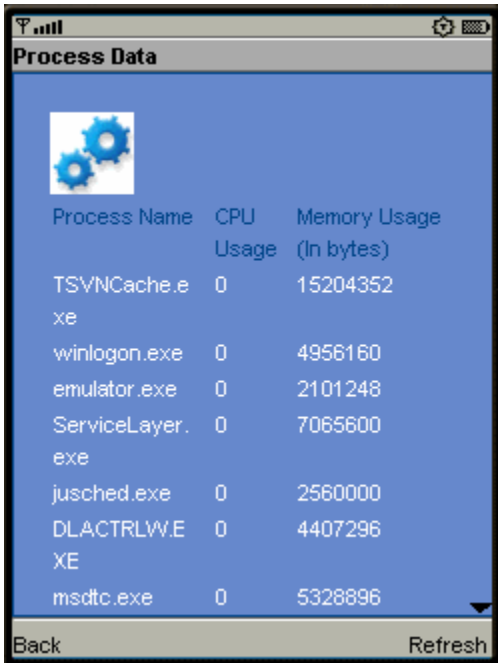


Figure 35: Current Process Data Screen

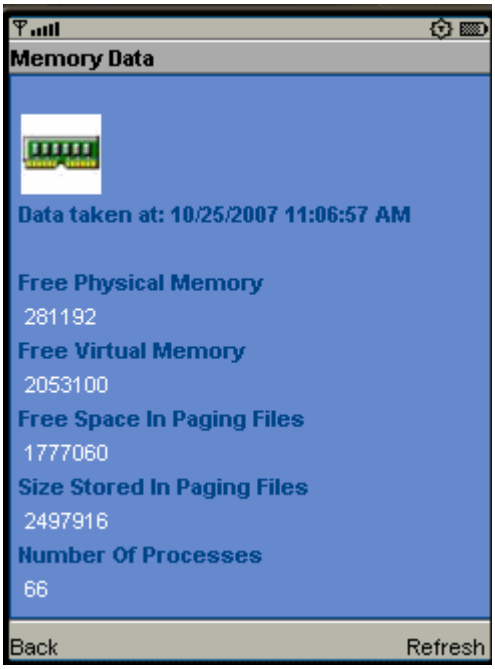


Figure 36: Current Memory Data Screen

Figure 37: Current Hard Disk Data Screen

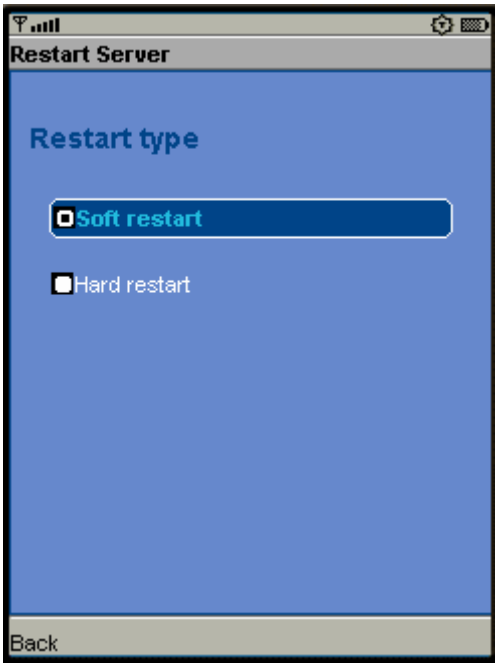


Figure 38: Restart Server Screen

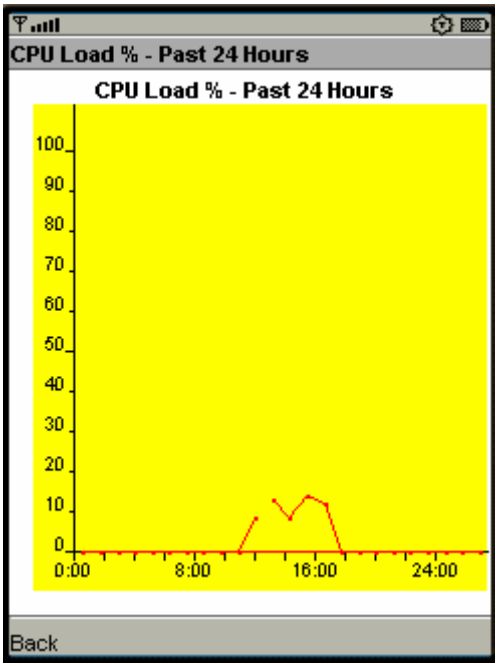


Figure 39: Processor History Screen

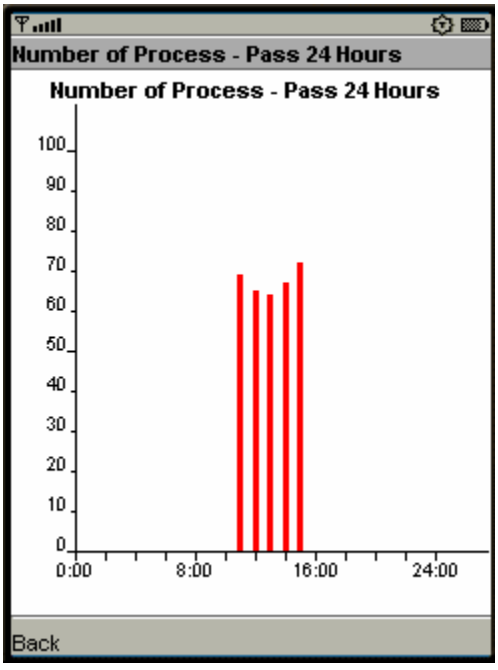


Figure 40: Process Histroy Screen

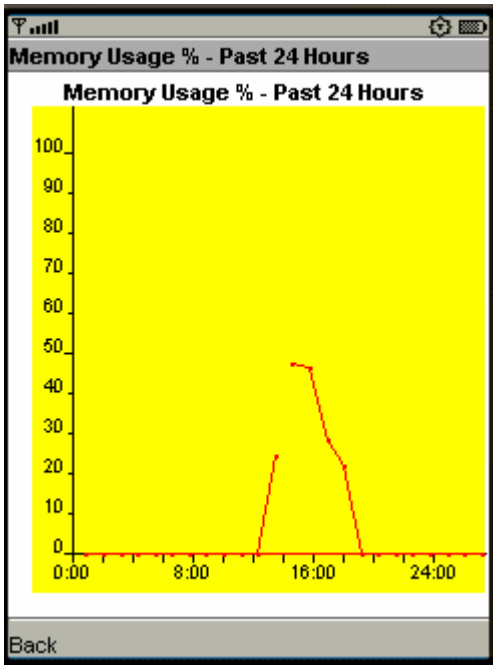


Figure 41: Memory usage History screen

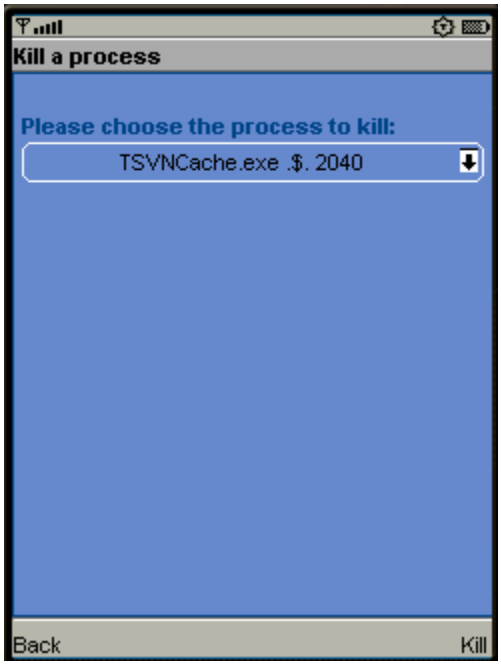


Figure 42: Kill Process Screen

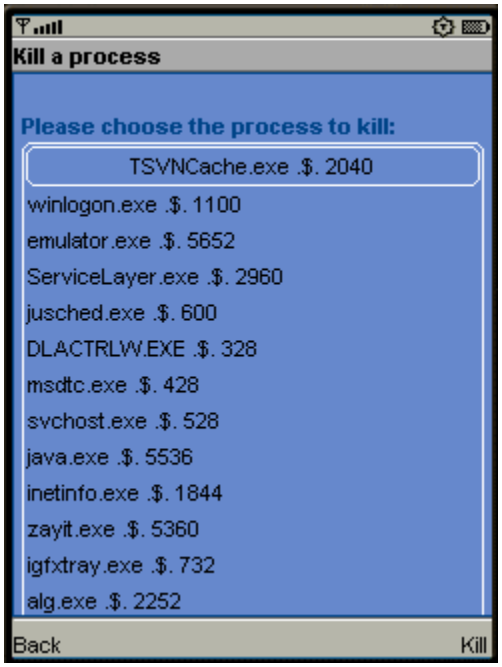


Figure 43: Kill Process Drop-Down menu

Figure 44: Run a process

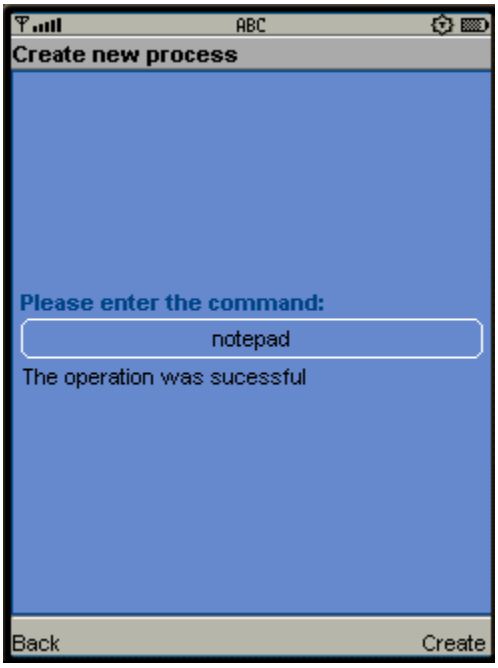


Figure 45: Run - Successful operation

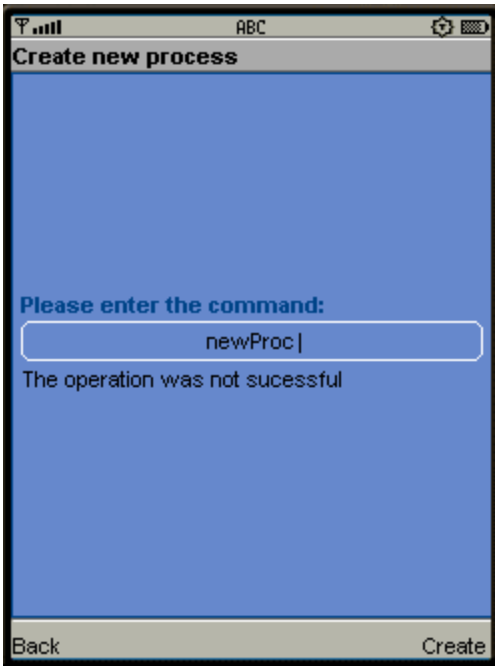


Figure 46: Run - Unsuccessful operation

6. PROSPECTIVE DEVELOPMENTS

Due to time constraints, there are a few parts of the server watch system that could not be completed. Also, there are a few things that are out of the scope of this project that should be added to the system. The following is the list of such developments needed and a brief explanation for it.

- Web application
  This was supposed to a part of the system but due to time constraints, it could not be completed.

- Compression
  The data has to be sent over the wire needs to be compressed. This is because the data being sent is in Xml format. Xml is very verbose. And compressing this data will reduce the data transfer by a significant amount. There are many places in the xml data where you there is a lot of repetition in the names of the tags. Compression algorithms such as Binary XML can reduce this quite significantly. A Binary XML compressor was developed at MCS but it works with a custom xml parser only. There were a few bugs found in the parser so I had to go ahead with kxml to parse xml on the mobile side.

- Security
  The security feature was not added to the system as this system was just a prototype that can be used to demonstrate the working and survey market possibilities. It would be good to have a security feature in the communication layer of the mobile application and the Mobile channel. An encryption algorithm should used to accomplish this.

    - IIS's HTTPS functionality
      For authentication purpose, it would be easier to make use of IIS's HTTPS functionality. This will ensure safe login and transactions.

- External device
  This part could not be completed. Here, the external device will receive polling signals from the server. When the external device will not receive any response from a server, it will label the server as non-responsive. It will then send the user a SMS stating which server is down. This has already been implemented in the system. The part that I not implemented is where the user will request this device to restart the server. An external 1-wire device was created to serve this feature. This device will send the corresponding server's main board a signal to restart. Upon completion, this will be a core part of the system as one of the system's aim is to alert the user when a critical situation has occurred.

- Multiple servers

  This system should be able to server multiple servers in the future. It is unrealistic to create a system that will monitor a single server. In modern day distributed computing, it is hard to find a server that can independently server any request. This can be accomplished by many different ways. Firstly, the ER diagram (i.e. the relations in the database) needs to be modified to log data for different server. A separate entity for a server will be needed. And references to this in the Processor, Memory and Process_Table would be needed. Other thing needed will be for the mobile channel to be able to discover reporting service on many servers. This is currently hard coded.

- Mobile application should ask for the server address

  In the current version of the server watch system's mobile application, the application is made for a specific server. This needs to be configurable by the user. The user should be able to enter the domain name (or even the global IP address) of the server where the mobile channel is situated and the mobile application can start communicating with that server.

- Improve interface

  The interface of the mobile application needs to be improved with more colors and a good layout.

- Mobile data storage to store history

  Currently, when displaying the history data, new data is fetched every time the user wants to view it. This data can be stored the mobile storage and can be retrieved every time the user wants to view this information and the user will have an option to refresh it and then the information will be refreshed, new data will be displayed and will replace the old one in the mobile storage.

- Porting to Linux platforms

  The system on the server-side runs only on platforms that support .NET 2.0. In the future, if a new version of Mono (Main Page - Mono, n.d.) comes out that supports .NET 2.0, this system can be ported into a server running Linux operating system.

- Temperature

  A research needs to be done on monitoring the temperature of a processor and RAM. A temperature probing device needs to be installed on the servers that need this kind of monitoring. The research should check for the number of servers that already have temperature probing devices installed and how many server administrators would be willing to install such a device for monitoring the temperature remotely.

7 LEARNING OUTCOMES

- Research Skills

  I have developed good research skills through out the project. There were many parts in the projects that had to be researched on. Like the platform for the mobile application, platform for mobile channel and reporting service, etc. Also, many more granular forms of research such as the best way to get processor usage of all the processes were carried out.

- Project management skills

  This project has taught me the value of documenting, planning and scheduling a project and also a few concepts about project management. The project was carried out through out without a team. All the project management tasks were done by me. This gave a great opportunity to learn a lot of project management skills like creating my own project plan and a project road map for a real world project. These skills will be very useful in the future.

- Importance of planning phase

  The planning phase of a project is a very vital phase. If we can plan the aspects of the system in the early stages of the project, we know how the system is going to look like and understand what kind of complexities can be faced during development. Some of these complexities are avoidable during the planning phase itself. For e.g. in the planning stages few problems with modeling the storage of process data was faced. The data was to be modeled in such a way that the data for each process was to be stored along with their time of creation and time the process stopped. This seemed to be taking four entities. This was then reduced down to two with some careful observation and understanding.

- Constant communication

  Constant communication between the project owner and project developers is a very vital part of a project. Without this, the direction of the project may change from the way the project owner expects it. Then going back to the point where the changes can be made becomes difficult and is very time consuming. Throughout the project I tried and maintained good communication with my project owner and other colleagues at MCS.

- Importance of testing

  Testing each and every part of the project that is developed at the time of development makes the task of refactoring after the end of the project makes it a lot easier. Every part of the developed version of the product should be thoroughly tested and checked if it is doing what it is supposed to do. Integration testing is as important as unit testing. Parts of the system work

well in isolation but they sometimes don't when they are supposed to work with other parts of the system. If they are not taken care of early, they become very hard to spot out.

- Foresee Constraints

  There are many constraints that are placed on the system that needs to be foreseen. Like some servers give the administrator the rights to do certain things but doesn't give other users to. This can create a problem when certain information is not available. Some of the code parts worked fine at MCS computers but they didn't at the university computers. This may be because of the permissions that students get at university computers. This made me understand that when a certain rights are not available, the user should be warned about it and the system should run only when the user is in administrator mode.

- Power of mobile applications

  Applications made for mobile devices can very useful tool for our day to day life. It makes it easier for users to access and control anything from anywhere as long as the mobile device is able to communicate with a network. Different types of channels such as HTTP or SMS can be used for mobile applications. This brings in a lot of opportunity for every to make full use of this technology. This is an area of IT that is developing very rapidly. A wide applications ranging for games for entertainment to banking applications for serious banking to accessing ERP systems to controlling remote assets are being developed every day. The range of such applications spread into all dimensions where desktop computing has reached. And with mobile devices becoming more powerful with better processing power and greater communication ability, there is not limit on what mobile applications can do.

- Good user interface design

  A good user interface is a very important factor when it comes to building a mobile application. Traditionally, mobile applications do not have a user interface that is as good as that can be seen on a desktop application. But with technologies such as FlashLite and J2ME Polish taking steps to allow a developers create attractive user interface, the mobile applications now have a good user interface to make the user feel good about using a mobile application from anywhere in the world. The user likes to use an application with good user interface and reasonable functionalities that a user needs compared to an application with good functionalities but not a good user interface. I have learned this important lesson about designing the user interface of the mobile application.

BIBLIOGRAPHY

.NET Remoting Overview. (n.d.). Retrieved October 21st, 2007, from MSDN: http://msdn2.microsoft.com/en-us/library/kwdt6w2k(vs.71).aspx

About WMI. (n.d.). Retrieved March 2007, from MSDN: http://msdn2.microsoft.com/en-us/library/aa384642.aspx

Benefits of WMI in .NET Framework. (n.d.). Retrieved March 2007, from MSDN: http://msdn2.microsoft.com/en-us/library/ms186144.aspx

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM , Volume 13* (Issue 6), 377 - 387.

Debbabi, M., Saleh, M., Talhi, C., & Zhioua, S. (2005). Security analysis of mobile Java. *Sixteenth International Workshop on Database and Expert Systems Applications, 2005* (pp. 231 - 235). IEEE.

Distributed computing. (n.d.). Retrieved October 21, 2007, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Distributed_system

IIS.net : HOME : Microsoft Internet Information Services. (n.d.). Retrieved October 21st, 2007, from IIS.net: http://www.iis.net/default.aspx?tabid=1

Java Servlet Technology. (n.d.). Retrieved October 21st, 2007, from java.sun.com: http://java.sun.com/products/servlet/

Java Web Services At a Glance. (n.d.). Retrieved October 21st, 2007, from java.sun.com: http://java.sun.com/webservices/

Kolsi, O. (2004). MIDP 2.0 security enhancements. *37th Annual Hawaii International Conference on System Sciences* (p. 8 pp.). IEEE.

Koopmann, J. F. (2005, April 12th). *Embedded Database Primer.* Retrieved from DBAzine: http://www.dbazine.com/ofinterest/oi-articles/koopmann5

Main Page - Mono. (n.d.). Retrieved 21st 2007, October, from Mono: http://www.mono-project.com/Main_Page

PHP: Hypertext Preprocessor. (n.d.). Retrieved October 21st, 2007, from PHP: http://www.php.net/

Querying with WQL. (n.d.). Retrieved March 2007, from MSDN: http://msdn2.microsoft.com/en-us/library/aa392902.aspx

Quin, L. (2007, May 8th). *Extensible Markup Language (XML)*. Retrieved from World Wide Web Consortium: http://www.w3.org/XML/

Remote Method Invocation Home. (n.d.). Retrieved October 21st, 2007, from java.sun.com: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

Snavely, A., Chun, G., Casanova, H., Van der Wijngaart, R. F., & Frumkin, M. A. (2003). Benchmarks for grid computing: a review of ongoing efforts and future directions. *30*, pp. 27 - 32. ACM Press.

The .NET Show: WMI Scripting. (2006, April). Retrieved from MSDN: http://msdn.microsoft.com/theshow/episode.aspx?xml=theshow/en/episode055/manifest.xml

Welcome! - The Apache HTTP Server Project. (n.d.). Retrieved October 21st, 2007, from Apache.org: http://httpd.apache.org/

Wesson, J. L., & van der Walt, D. F. (2005). Implementing mobile services: does the platform really make a difference? *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries* (pp. 208 - 216). White River, South Africa: South African Institute for Computer Scientists and Information Technologists.

What's New in MIDP 2.0? (n.d.). Retrieved March 2007, from Sun Developer Network: http://java.sun.com/products/midp/whatsnew.html

White, J. (2001). An introduction to Java 2 micro edition (J2ME); Java in small things. *23rd International Conference on Software Engineering* (pp. 724 - 725). Toronto, Ontario, Canada: IEEE Computer Society.

Windows Management Instrumentation (WMI). (n.d.). Retrieved March 2007, from MSDN: http://msdn2.microsoft.com/en-us/library/aa394582.aspx

WMI .NET Overview. (n.d.). Retrieved March 2007, from MSDN: http://msdn2.microsoft.com/en-us/library/ms257340.aspx