Web Based Information System for Unix Servers

R Madhavan

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the University of Wales.

Supervisor: Dave Price

University of Wales, Aberystwyth

17 April 2004

DECLARATIONS

The content of this dissertation is the result of my own independent work and investigation except where otherwise stated. All sources are acknowledged by explicit references to the bibliography.

Signed: . . . . (R Madhavan)

Date . . . .

I declare that this work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed: . . . . (R Madhavan)

Date . . . .

I hereby give my consent to the dissertation, if successful, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed: . . . . (R Madhavan)

Date . . . .

Acknowledgements

I am thankful to my supervisor Dave Price for his time, help and support to me during this project. I am also grateful to my employer in allowing me to use the necessary hardware and software required for this project. Finally I would like to thank my wife and children in tolerating me during this project work and writing the dissertation.

Abstract

In today's world of Information Technology (IT) application centric organisations, there is a requirement to have a range of information about their IT infrastructure accurately available, on demand, at all times. Servers which are running large commercial applications and databases, are some of the most critical components of any enterprise's IT infrastructure. IT managers, system and database administrators, application developers and users within that organisation often demand information about the underlying environment on which applications are running. The required information is primarily about server hardware, operating system, customized configuration and so on. In most cases this information is currently neither available at the right time nor is it accurate.

This project has attempted to develop and implement a web based information system to provide an easy and single point of access for server information, especially for Unix based computer systems. The objective is to collect server related data regularly from any number of Unix servers without any manual intervention, store the collected data in a central data store, and construct a web site to provide access to this information within the organization.

While this project succeeded in providing a single point of access for server information and met the users' requirements, the system could have been improved and built to provide additional capabilities and sophistication. Possible ways to achieve further enhancement and improvements are also discussed in this report.

Contents

1. INTRODUCTION

1.1. The need for system information

In any large organisation, it is quite common to see many numbers of servers running wide variety applications and databases supporting that together, support their business operation. During entire life cycle of the application various people concerning to that system would need to know about the underlying operating environment on which that particular application and database are being run. The people concerned include IT managers, system administrators, database administrators, application development and maintenance team people and users as well. The system information sought is related to server hardware, operating system, detailed configuration and so on. Having this information accurately and at the right time, contribute to effective application development, system maintenance, troubleshooting and also timely reporting.

There are many situations in which a system administrator is required to provide the information about servers to various people. Below are few examples of those situations.

An application developer is interested in knowing the release and version of compiler available in the system. This is because, he was told that the compilation error he is currently facing with his piece of code, is due to a bug in the compiler itself and this is fixed in the new release of the compiler. The database administrator of the system would like to know, what operating system patches are currently installed in the system. This is because of the requirement from RDBMS vendor to verify whether all necessary operating system patches are installed in the computer, in order to have the database to run smoothly. An IT infrastructure manager would like to know current hardware configuration of one of the servers running in his data center to prepare a report to the company management. The system administrator himself would probably want to know about the system configurations like networking, kernel parameters, device configuration and so on.

So it is quite apparent that many different people within the organization need server related information for various reasons and purposes. And it can also been seen that these pieces of information are required to solve many day to day problems which they would be facing with their computer systems. Hence in this context, the availability of accurate server information at the right time is an essential piece of data, which help to perform their job in a more effective manner. The availability or absence of this information could also make a difference in meeting their clients or customers need on time.

1.2. Existing ways of retrieval and need for a data collection system

Typically the underlying operating environment or the system information is obtained in one of the following two ways,

1. By referring to a separate system documentation, which is generally maintained by system administrators of the organization. This document is either updated periodically or whenever there is a change in system configuration. The respective system administrator of that particular server normally does this.
2. In the second method, there is no documentation maintained, but rather whenever some one requests server related information, system administrators logon to that particular server and collect the required data and produce the necessary output.

The disadvantage of the first method is that it requires someone to constantly update the document in order to keep it as most up to date. The challenge is also to make the document available and accessible to everyone within the organization. The second method is totally human dependable and manual which relies on one person (system administrator) to provide the necessary information. It is hardly practical to expect the system administrator to be available and reachable at all time.

In this information age, it is not surprising to hear that people expect accurate information to be accessible at all times and from anywhere. This is no exception even when it comes to system related information. But it is very obvious that both these traditional methods of system information access prove very much inadequate in meeting the expectation of 'information on demand'. So it is quite natural for people like IT managers and technical administrators to be disappointed when they are not able to get the required information accurately and in time. Also in any large organisations, people sit across different geo-graphical locations working under different time zones. Hence this also makes it difficult to get information whenever they need it. Hence there is a need for a better way of providing this information whenever it is required.

It would be very good if there were an overall system that collected all the necessary data from any number of servers and provided a single point of access to retrieve that data. This means, accurate information is provided at any time and also made accessible from anywhere within the organisation.

1.3. Project aim and objectives

The primary aim of this project is to produce an information system to give a single point of access through a web interface for server information of AIX, HP-UX and Solaris operating system based computers. The system development and implementation was aimed to meet the following four major objectives,

1. The system should collect data from HP-UX, AIX and Solaris servers (according to the users requirements) automatically and regularly without any limitation of the number of servers.
2. The collected data should be stored in a single central data repository.

3. A web site to be constructed to provide a web interface to query the data store and produce the necessary output.
4. The system implementation should also make it easy to include additional servers for data collection in future.

1.4. Project management

"Effective project management focuses on people, product, process and project"[1] [3]. While the people and product factors are mainly applicable for large software development projects, I recognised that both process and project were critical factors to be considered for the successful development and delivery of this project.

The process factor is essentially about the software development life cycle (SDLC) methodology (also called as software process model) used to develop the software and make it operational to meet the customer's business requirement. On the other hand, the project factor focuses on the planning and execution of various phases by identifying the critical milestones and goals. In the next two sections I discuss the SDLC methodology that was chosen and the overall planning for this software implementation project.

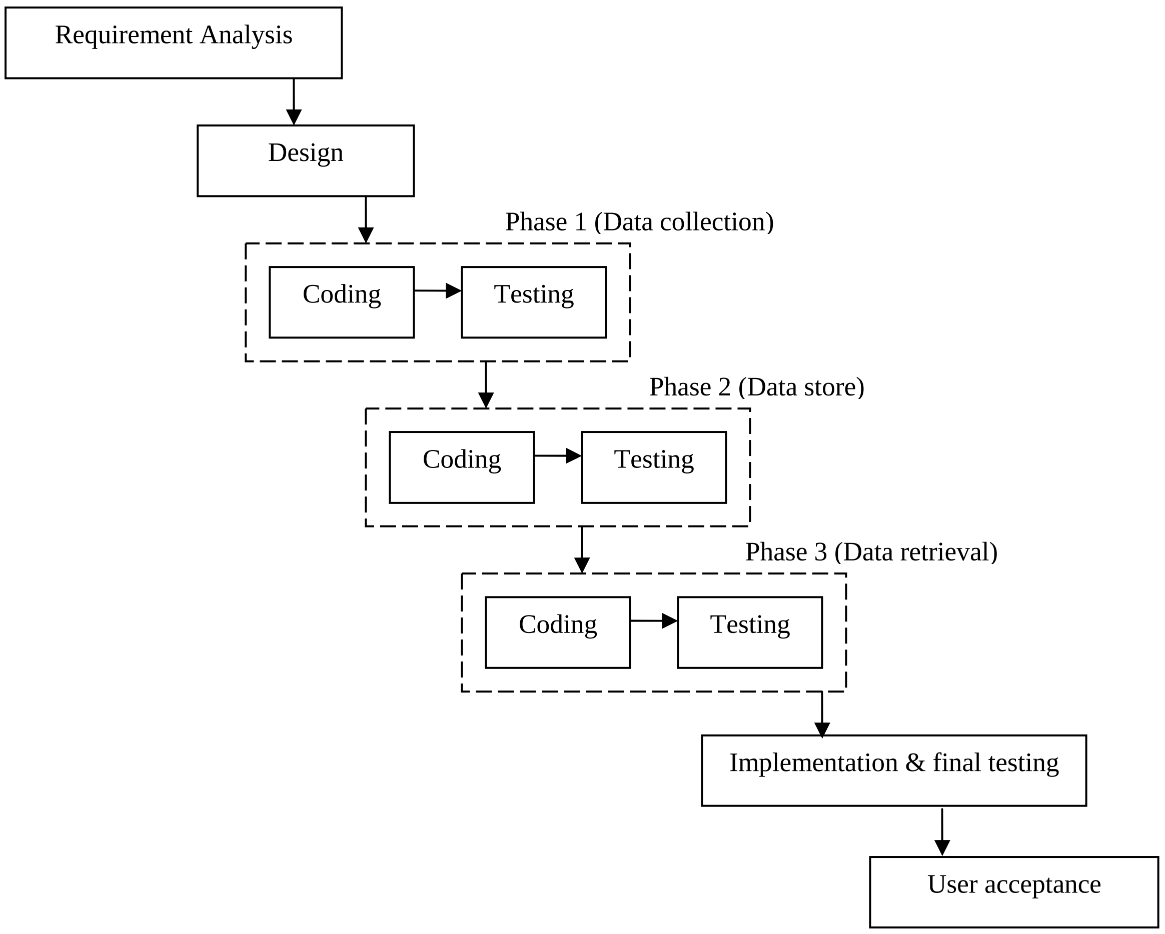
1.4.1. The software process model

Various process models available today have their own strengths and weaknesses and I thought the model I select should be an appropriate one to deal with the nature and complexity of this project. At the beginning of the project, I believed that the user requirements could be defined very clearly. I also thought that even though there would be lot of technical challenges ahead in this project, from the project management point of view this project could be considered as relatively small and not very complex. The most widely used 'Classic' or 'The waterfall model' has come under lot of criticism due to its rigid design and inflexible approach[2] [7]. However, I thought an appropriate change in the coding and testing phase in the original model would make an appropriate fit for this project. Since this project is essentially about collection, storing and retrieval of the data, I have recognized the need of three different programming languages to be chosen. This means the best approach is to code and test each layer before moving on to the other layer. This has made me to come up with the model as shown in the figure 1.1, which is a typical 'waterfall model' with a phased approach for coding and testing phases.

[1] Roger S. Pressman (2001) *SEPA, 5/e.* McGraw-Hill International edition.

[2] Centre for Technology in government, *A Survey of System Development Process Models*, http://www.ctg.albany.edu/publications/reports/survey_of_sysdev

Figure 1.1 *Software development life cycle methodology*



1.4.2. Project planning

Having decided on a software development process model, the next logical step was to draw a project plan before embarking on the next phase.

For a very large software project, the planning should essentially comprise of estimation on number of people required, estimation on cost and the drawing up of a schedule. While the first two factors are beyond the scope of this project, a proper project schedule with milestones to reach the final stage is a very important aspect to be considered for a project of any size and complexity. The project schedule should accommodate the process model that has been decided and also be reasonable enough to complete within the given time frame. The schedule was drawn up based on my own estimation of time for each phase in the life cycle model and the time I could spend on this project on a weekly basis. The project schedule with activities is presented in Appendix A.1

As part of the background study required to carry out this project, the World Wide Web was used and proved to be a very valuable resource. In addition to the traditional source of information and knowledge (mainly text books, which are mentioned in the bibliography), I have used the knowledge and resources available in the World Wide Web more extensively in order to carry out this project. These resources include online manuals, publications as well as various open source software group's (PHP, Apache and so on) web sites. Both traditional as well as web resources, which were used for this project, have been acknowledged in the bibliography explicitly.

2. REQUIREMENTS ANALYSIS

In research conducted and published by Standish group[3] [8], it was discovered that "incomplete user requirements" and "lack of user involvement" were the top two reasons for IT project failure. This clearly put the importance and emphasis on this phase for the overall success of this project. Hence it was decided that at the end of the requirements study phase, it would be necessary to have the user requirements listed to form a basis for the overall system development.

Before the requirements was finalised, I had first discussed with selected users about the scope and requirements. This phase was essentially focused on requirements gathering and finalizing.

2.1. Requirements gathering

A total of three key users were selected (one from each function namely system administration, database administration and software development) to form a group to jointly finalise the requirements. A first discussion was initiated with these users, which started with the question of 'what type of computer systems involved for data collection?' and 'what data should be collected?'. Henceforth, over a period of two weeks both informal and formal discussions led into finalising the requirements and subsequently a user requirements list for the system were drawn up. The final requirement (discussed later in chapter 2.2) was drawn out of the mutual agreement between users and myself, and the following two points are worth mentioning about compromises, which were made between us.

- It was mentioned that it would be useful to have the system resource utilisation (CPU and memory) data also collected and reported. But gathering and reporting this information means, enabling the system activity reporter (SAR) and processing the collected data. This would require additional setup and disk space management in each server involved in data collection. Hence it was explained that taking this requirement would be difficult due to the given reasonable time frame for this project to complete. Moreover the organisation[4] already had a commercial monitoring tool, which was deployed for this purpose. So if we also have to do the same then it would be redundant information within the organisation. Hence it was agreed mutually that this system would exclude the SAR data reporting as a requirement.
- We also discussed providing the maximum scalability (CPU and memory) information of the server as part of this information system. But this information is not really an internal item of data for the servers, but rather it is a characteristics decided externally by the respective hardware vendor for each server model. Hence it was decided to exclude this from the requirement and concentrate on collecting and reporting the internal information of the servers.

[3] The Standish group, *The CHAOS report*. http://www.standishgroup.com
[4] Here the organisation is referring to the one in which I carried out this project.

The project schedule was also shared with users and it was also agreed that there would be an acceptance or validation test to confirm against the agreed requirement of the system.

2.2. User requirements

This chapter produces a summary of the requirement, which was discussed and agreed with the users.

Data collection requirements

- The system should collect data from AIX, HP-UX and Solaris operating system based servers.
- Each server involved in the data collection is referred as 'host'. The system should collect data about the host, hardware it is running, operating system (and it's related information) and detail about customised configuration.
- Each of these four main categories (namely host, hardware, operating system and detailed configuration) has many individual items, which should be collected and reported by the system. These detailed items are presented in Appendix A.2
- The method of data collection is not of any importance to the users and it was left to me to decide and implement.
- The data must be updated with the current information at least once in a week.

Data repository requirement

- The collected data should be stored in a central data store.
- The type of data store is not of importance to the users, however it is preferred that the central data repository should provide a search capability (this search capabilities would be used in future) and allow to modify the data easily.
- The primary purpose of the data store is to hold the current information, however it is nice to retain the old data for future reference.

Data retrieval or user interface requirements

- It was reconfirmed that a web browser would be the user interface.
- The home page or the first page of the site should list all hosts with the corresponding application name and provide an option to select the host to query for information.

- Once a host name is selected and submitted to the web server, the system should retrieve all required server information from the data store and display in one single web page.

- The web interface would be used only to query the data store and there would not be any data upload from the web browser.

- A basic user authentication mechanism is required before any users can retrieve the data from the web server.

3. OVERALL DESIGN

3.1. Design overview

For the purpose of ease of design, coding and testing, this information system has been decomposed into three main layers[5]. They are namely,

1. Data collection layer
2. Data store layer
3. Data retrieval layer

All three layers are implemented within a single Unix server. So essentially this Unix server (referred as master server) is the core of the entire solution. A HP-UX based server was chosen for this purpose and the decision to choose this was due to two main reasons. First, my work place had a HP-UX server for testing purposes and it was naturally the first thing to see whether this server could be used for this project purpose. Second, it was also found that Hewlett-Packard company bundle Apache web server and PHP programming language as part of the overall web solution under HP-UX. Because both Apache web server and PHP were considered[6] for the data retrieval layer, it was decided to use HP-UX as the master server.

Figure 3.1 illustrates the overall system flow. The detailed configuration of the master server and software packages used are listed in Appendix A.3

3.1.1. Data collection layer overview

The data collection layer is responsible to bring all necessary data from multiple hosts to the master server before it can be organised in the data store. Following two choices were considered in choosing the data collection method.

1. Each host initiates the data collection process by itself and sends the collected output to the master server (*push method*)
2. Master server initiates the data collection process and gets the collected output from all hosts (*pull method*).

The *push method* needs the collection script to be placed in each host. This means, if any changes would be made to the script or program[7] then it has to be distributed to all hosts. Failure to update any one host could make the system fail in any one of the layers.

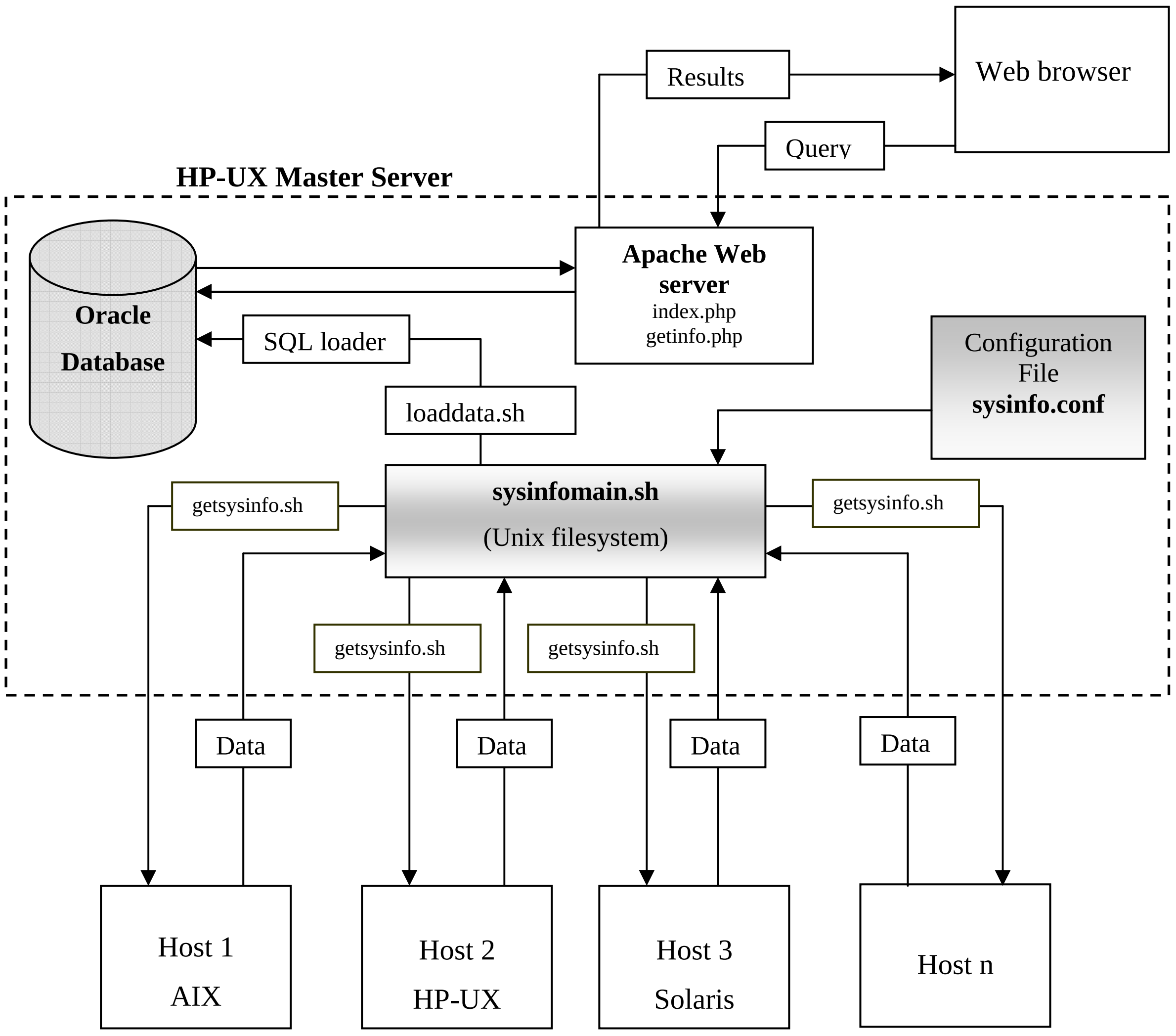
On the other hand, the *pull method*, essentially about having a main program in the master server, which begin by reading a configuration file for the hosts involved in data collection and then gets the data from each host. This will make the

[5] Henceforth, these three layers will be referred as when required in this report.
[6] The choice of tools, is discussed in detail in Section 3.2
[7] Changes to script or program normally happen due to bug fix or any enhancement(s) are to be made to the existing system.

maintenance easy, as it would be necessary to keep only one copy of the data collection program.

So it was concluded that the *pull method* would be a better choice considering the less effort involved in maintaining this environment in the future. Hence I decided that a single program consisting of different procedures for each operating system should be written and maintained in the master server. Upon initiation of the data collection process, this program is sent to the respective target host to collect necessary data. Once data is collected at the target host, the same script then sends back the data to the master server for further processing.

Figure 3.1 *Design overview and system flow diagram*



3.1.2. Data store layer overview

The data store layer is primarily responsible for organising the collected data and allow the data retrieval layer to retrieve the data. Two types of data store were considered for this project and it was decided to use a database. The other choice was to use text files stored directly in the Unix filesystem, but the decision to go with the database was based on following criteria or reasons,

- The database is a better choice in providing search capabilities.
- Using the database, data could be well organised and structured.
- Database is better at providing data security. In the case of text files, it is required to have proper file access permissions set for many files and directories.
- Database is also a preferred choice for dynamic web content applications, as most of the web applications work seamlessly with many database management systems in providing dynamic web contents.

While the decision was made to use the database, the question of which database to use, was not really a trivial decision. This is discussed later in section 3.2.

3.1.3. Data retrieval layer overview

The data retrieval layer is to receive the query from web browser, pass the query to data store and send results back to web browser. So this layer has two parts, one the web server portion and the other is a web application portion.

The web server portion is responsible in receiving query from web browser(s) and pass results back to them.

The web application portion interacts with the database to extract required content in response to the query received from web browser.

3.2. Choice of tools

3.2.1. Data collection layer

In selecting the programming language for this layer, two main criteria were defined. One, the programming language should be able to run under all three flavours of Unix operating systems (HP-UX, AIX and Solaris). Two, it should be able to support and have the ability to run Unix system administration commands which are necessary to collect data from respective hosts. With these criteria in mind, Unix shell script and Perl programming language were considered for this layer.

Perl programming language was considered to be more sophisticated and flexible in output formatting. It required me to pick up an additional skill set and hence this could have an impact on the project schedule. On the other hand, Unix shell scripts can meet the requirement for this project, considering the fact that this is being widely used by many system administrators in the industry to manage complex environment. Also my familiarity and experience with Unix shell scripting could help to shorten the overall time required to implement this system.

As I already had few new technical skills to be acquired for this project (Oracle database, PHP and web server), I decided to choose Unix shell script as the tool for this layer. Hence I used Unix Korn shell environment to develop the data collection program, which is used to collect the all necessary data to store in the database.

3.2.2. Data store layer

Having decided that a database would be used for the data repository, the next logical step was to choose the database from available choices. For this both MySQL [18] and Oracle database were considered and I decided to use Oracle database.

There were primarily two reasons to choose Oracle database instead of MySQL. They were,

1. I believed that Oracle's SQL*Loader utility was much superior and much more flexible tool (when compared to MySQL) to load the data into the database tables from the Unix flat files. Because the data collected was available as flat files before it could get inserted into the database, the ease and power of this utility was an important consideration for me in choosing the database.
2. Because my work environment has an Oracle environment, I thought choosing Oracle as data store for this project, would help me to pick up an additional new skill set. The experience and knowledge I would gain with Oracle from this project could also be used in my work place. Since both MySQL and Oracle were going to be a new journey to me, I found it would be more advantageous to use Oracle than MySQL.

However, I also searched and verified that the web application language (in this case PHP) was fully supported and had interface calls to communicate with Oracle RDBMS.

3.2.3. Data retrieval layer

As was already mentioned HP-UX is supplied along with a web solution, which contains both Apache web server [16] and the PHP programming language [5,9]. Hence it was naturally the first choice to use this combination unless otherwise it would be totally unwise to do so.

Both Apache and PHP go hand in hand very nicely. The Apache web server ability to parses PHP code embedded inside a HTML page, makes PHP language's popularity justified in the dynamic web applications area. PHP's wide support of a large number of backend databases including Oracle database made a comfort feeling in choosing this combination.

So I have decided to use Apache web server and PHP as the tools for this layer.

4. DETAILED DESIGN AND CODING

4.1. Design of each layer

4.1.1. Data collection layer design

The data collection layer design, was aimed to meet the following two main goals.

1. There should be only one script (with multiple procedures inside for each operating system) in the master server to collect the data from all the hosts.
2. It should make it easy and less effort to add any new host for the data collection.

The data collection layer has a single configuration file, which contains the names of all the hosts from which the data is collected. The desired hosts are added or removed as and when required and hence no hostnames are hard coded or pre-configured in any of the scripts or programs. The main script (which initiates the data collection from the master server) reads the configuration file and sends the actual collection script to the respective hosts to collect the data. The same collection script, which runs on respective host, sends back the collected data to the master server. The collected data is stored in the master server as text files before it is loaded into the Oracle database.

So essentially below three components constitute the data collection layer.

1. A data collection script *(getsysinfo.sh),* which actually runs on each hosts, contains the underlying Unix commands to collect the data and send it back to the master server.
2. A configuration file *(sysinfo.conf),* which has a strict format and contains names of all the hosts, from which the data should be collected.
3. A main script *(sysinfomain.sh)*, which initiates the data collection process reads the *sysinfo.conf* file and sends the *getsysinfo.sh* script to respective hosts. Once the data is successfully collected, this main script also executes necessary data loading scripts in the master server to load the data into the database.

The *getsysinfo.sh* script has all the Unix shell procedures for three operating systems. After checking the type of the operating system in which the script is run, the respective procedure is selected to run on that particular host. A final procedure in the same script sends all collected data from that particular host to the master server. The collected files are sent back to the respective sub-directory for that host in the master server.

The *sysinfo.conf* configuration file has three fields with a field delimiter separated between them. The first field holds the hostname of the host from which data should be collected. The second and third fields have the name and type of the application. The content from this file is essentially used to build the index page dynamically by PHP, so that the user can choose the hostname of the server to view the information.

The *sysinfomain.sh* script reads the *sysinfo.conf* file line by line and sends the *getsysinfo.sh* script to each host. This *sysinfomain.sh* also creates one sub-directory (in the master server) for each host in order to receive and store all collected data files from that particular host. This main script sends the *getsysinfo.sh* script to host by using the remote copy command (*rcp*) from the master server. After sending the *getsysinfo.sh* to respective host, it is run by remote execution command (*remsh)* from the master server.

Hence all target hosts are configured to allow the remote command execution from the master server. This is one of the pre-requisite to collect the data successfully by the master server. However, in order to avoid opening up of remote command execution from the host systems to the master server, the collected data are sent back to the master server by the *getsysinfo.sh* script using the file transfer protocol (*ftp)* command.

Hence at this stage it was decided to start with the coding of *getsysinfo.sh* script. But before starting this task, it was apparent that (from Software Development Life Cycle method chosen as well as from the feasibility point of view) the Oracle database table structure and the PHP application design must be completed and understood fully before start developing the *getsysinfo.sh* script. So the detailed design of both data store and data retrieval layers were completed before coding the *getsysinfo.sh* script.

4.1.2. Data store layer design

The Oracle database design [4] basically governs both the data collection script as well the web application (PHP) coding. At the end of this database design phase, the Oracle database table structure was decided so that the coding of *getsysinfo.sh* collection script could be started. The input data file format was also decided at the end of this stage. This input data file is used by the SQL*Loader utility to load the data from the flat file to the Oracle database.

The discussion with the users was helpful to derive the 'Enterprise rules' to form the basis for database design. Based on these 'Enterprise rules', the entities, attributes and their relationship were identified. Hence the database design was begun by identifying the entities that must be represented in the Entity-Relationship (ER) modeling. The ER diagram is presented in the Appendix A.4.

There were totally four entities identified. They were namely Host, Hardware, Operating System and Detail Configuration. The attributes of each entity were basically the data to be captured under the each category. But for the historical data (to be used by the application program in future and for further enhancement to the information system) one more additional attribute was identified for each entity. It was noted that there would be a necessity for a date attribute to be associated with each row in storing the historical data. This was due to the reason that there would be duplicate records for the same hostname and only distinguished by the date in which the data was captured. Hence there were basically two choices considered in finalising the entities and arrive the database schema.

The first choice, was to have the same entity (for example, Hardware) also to hold the date attribute, so that the same database table would also hold both current as well as history data. In this way it would be totally four entities and which would be translated into four tables in the database. Since the primary objective of this information system is to provide the current data, the application program logic should take care of identifying the most recently updated record and extract that particular row accordingly. So this would make PHP application program to build additional comparison and filtering logic inside the current data extraction program.

On the other hand, the second choice was to have two different sets of tables, one set would hold the current data and the other set would hold the history data. In this case, it would make totally eight tables (four tables holding current data and four more tables holding the history data) in the database schema. The Entity-Relationship is same for both sets of tables with one additional attribute (date) in each entity for the history information. It was also informed to me by some experienced Oracle database administrators that, whenever possible it would always be a better idea from performance point of view to separate history records and current records into different tables. Also it was understood from the users requirements that the frequency of reading the current data records would be much more, than reading the history records. Also choosing this design would make the application program to assume safely that the current tables has no duplicate records for the given hostname and hence could avoid building any additional comparison logic in order to display the most recently updated data to the user interface.

Hence it was decided to have totally eight tables in the database, four tables for holding the current data and four more tables for holding the history data. The names of the tables holding the current data are HOSTINFO, HWINFO, OSINFO and DETAILCFGINFO. The corresponding history data tables are HOSTINFO_HIST, HWINFO_HIST, OSINFO_HIST and DETAILCFGINFO_HIST. The data loading program takes care of copying the current data to the history tables before updating the current data tables. Even though there were four more additional tables designed to just hold the history data, it could be seen that there is no data redundancy for the same record within this two sets of tables. This is due to the reason that the each row in the history data tables is inserted only before the current data tables are updated.

The foreign key hostname (referencing HOSTINFO table) in HWINFO, OSINFO and DETAILCFGINFO tables ensured that the 1..1 relationship between entities was satisfied. However this would only ensure the maximum one participation but would not guarantee the minimum one participation. But this is taken care during the data loading. When data is loaded from the Unix text files to the database, it is ensured that each rows meant for that particular host is loaded into all four tables. There is also a log file generated during the data loading which would log if any rows were not loaded into the database.

The next important aspect of designing the table structure was, to choose the right data type for each attribute. In addition to primitive data types to store characters, numbers and date so on, Oracle also offers another important data type called CLOB (Character Large Objects). The CLOB data type was chosen to hold the free format attributes (like /etc/hosts file, /etc/services file, Software list etc) inside the database. However it was found out during the coding phase that the storing and retrieving of the CLOB objects are much more complex than handling the primitive data types.

Considerable amount of time was spent in finding out how to load this CLOB type data into the database using the Oracle SQL*Loader utility and as well retrieving using the PHP application program.

The next aspect as part of this layer is the SQL*Loader control file structure and the input data file format. One input data file and one control file is created by the data loading script for each table during the run time. One data loading script per table is written and that in turn creates the control file and the input data file for that particular table. So there are totally four data loading scripts to load data into four current data tables. Each control file is created during the execution of the script with SQL*Loader control commands. Before the SQL*Loader utility is called to load the data, the input data file is created by combining all the host data into one single file for that particular table. This approach is also made possible due to temporarily storing the collected data files from each host under separate sub-directories in the master server.

So totally four Unix scripts were written (*loadhostinfo.sh, loadosinfo.sh, loadnhwinfo.sh and loaddetailcfg.sh*) to load the data into the four current data tables. Each script creates its own SQL*Loader session and produce its own log file and bad file to trouble shoot any data uploading problems. But before this four data loading scripts are called, another SQL plus script (*copytables.sh*) is called to insert the data into the history tables by selecting all rows from the respective current data tables.

As soon as the data collection process is completed successfully, all four current data loading scripts are called from the *sysinfomain.sh* script. This approach ensured that the minimum one participation of entity is guaranteed. By inspecting the log files after the data loading, any errors or problems during the loading could also be identified easily.

As mentioned earlier both the E-R diagram and table structure are presented in the Appendix A.4 for reference

4.1.3. Data retrieval layer design

The first page or the home page of the web site is served by a single program, which has both HTML and PHP code inside. The PHP code embedded in the same file interacts with the Oracle database and extracts all the rows from HOSTINFO table of the Oracle database. The extracted data is then passed to the web server to serve to web browsers. The home page has a drop down list html form to choose one of the available hosts listed in that web page. So essentially the index page (*index.php*) of the web site has a dynamic content, which depends on the Oracle database HOSTINFO table. Once the name of the host is selected the selection is assigned to a variable and passed to the next PHP program (*getinfo.php*).

The second page is the result of *getinfo.php* PHP program, which takes the selection of the hostname from the first page and queries the HWINFO, OSINFO and DETAILCFGINFO tables of the database. The result is used to construct one html table for each database table. However the CLOB objects are not displayed in the same page, instead a hyper link is provided to view that particular object.

Hence it was found out that in addition to one primary page to display all the information for a particular selection in the first page, each CLOB object needed to be displayed in each separate page so as to make the data presentation much more elegant.

However, it was also understood that there could not be too many CLOB object retrieval programs, but rather it should be only one common program to retrieve all CLOB data type attributes in all tables. This would make this layer design much more simple but posed a bit of challenge to the coding phase as HTML does not give much choices in passing variables between pages. In the second page once a particular HTML link is selected for the retrieval of CLOB attribute, then the corresponding table name and the column name are passed as variables to another PHP program. Hence only one CLOB object retrieval program is written to display the contents of all CLOB data types from all three tables.

So this layer has totally three PHP programs (*index.php, getinfo.php, getclob.php*) irrespective of any number of hosts configured in the *sysinfo.conf* file. The first program (*index.php*) is to build the home or main page, the second program (*getinfo.php*) is to display all attributes for that particular host (except the CLOB objects) and the third program (*getclob.php*) is to display the selected CLOB object for that particular host.

The user authentication was another aspect for this layer. This was implemented by using the Apache web server's default mod_auth authentication module. The users were created using the Unix *useradd* command and password was set using the Apache's *htpasswd* command.

4.2. Coding of each layer

4.2.1. Data collection layer coding

Out of the three components (*getsysinfo.sh, sysinfomain.sh, sysinfo.conf*) for this layer, the core of this layer is the *getsysinfo.sh* script. This script, which is written in Korn shell, essentially comprises of AIX, HP-UX and Solaris operating systems system administration commands. The script is written to satisfy two conditions. The first, is to collect all parameters identified during the user requirement and the second, is to match the output format to the input data file of SQL*Loader utility. The script also ensures that the output format matches the data type defined in the database.

The *getsysinfo.sh* script is written to capture data for three tables namely HWINFO, OSINFO and DETAILCFGINFO. The data for the HOSTINFO table is loaded using the *sysinfo.conf* file (as this file has all necessary data for the HOSTINFO table). Presented below is the sample of the *sysinfo.conf* file.

Sample *sysinfo.conf* file

```
uxd07+Oracle+Test
uxp18+Finance+Production
uxd02+Billing+Development
uxd18+SAP+Development
uxd15+Engineering+Development
```

Before I completed the coding for all three operating systems, I decided to complete the *getsysinfo.sh* script for one operating system flavour first. The script was first developed on AIX operating system [15]. Totally five procedures for one type of operating system were written. Out of five procedures two procedures are common to all three operating systems. One of the common procedure set few shell variables and execute non-operating system specific commands. The second common procedure sends the collected data to the master server. The other three procedures are unique to each flavour of operating system, which are meant to collect data for OSINFO, HWINFO and DETAILINFO tables. Once the five procedures were completed, it was tested in that particular AIX host to confirm that it had collected all necessary data according to the specification and the output format conform to the input data file requirement.

Once *getsysinfo.sh* shell script was fully tested and confirmed its desired functionality, the same script was re-used next in a HP-UX server. The two common procedures were re-used without any changes, but the three other Operating System specific procedures were modified for HP-UX commands [13] in place of AIX commands to capture the same information. But the large portion of the code was totally untouched. This approach has made me to save lot of time, as I had only spent far less time in getting the script to work on HP-UX than on AIX. This approach also ensured that I was able to get the desired output format for the HP-UX at the first time itself as the code was fully tested in AIX before.

The same approach, which was taken for HP-UX had been followed for Solaris operating system as well. Again as expected, it only took much less time in getting the script to work and test fully in Solaris than on AIX operating system.

Following are two examples of changes made in coding to suit the individual operating system specific commands.

To identify whether the Unix kernel is running in 32 bit or 64 bit mode, in AIX *bootinfo –K* command is used. The same line of code is changed as *getconf KERNEL_BITS* for HP-UX and for Solaris [14] *isainfo –v* command is used.

Similarly the installed software list is obtained by using the *lslpp* command in AIX, in HP-UX it is obtained by using the *swlist* command and in Solaris it is required to use *pkginfo* command.

Once all the data collection procedures were tested fully, they were combined into one single script to get the final *getsysinfo.sh* script. The type of the operating system is checked first before branching to select the respective procedures for that operating system.

The script was made to ensure that following coding standards were followed.

1. Any hard coding of name of log files, directories or any other values inside the program should be avoided and instead variables should be used and defined at the beginning of the script.
2. All variables must be in upper case and alpha numeric. But the variables to hold the values of the database columns must be distinguished by beginning with upper case and followed by lower case letters or numeric.
3. Name of the variables should be appropriate to describe its purpose.

The *sysinfomain.sh* script begins by reading the *sysinfo.conf* file and copy the *getsysinfo.sh* to the first host specified in the configuration file. Once copied the next line of code, runs the *getsysinfo.s*h script on that particular host using the *remsh* command. Once this script is successfully completed in running on that particular host, all necessary data files are available in the master server under the specified directory for that host. This process is repeated for all the hosts using a single *for…do…done* loop. Once data collection is completed for all the hosts, then the same *sysinfomain.sh* scripts executes the data copying script (*copytables.sh*) which insert all rows from the current tables to the history data tables. Once this is completed, the *sysinfomain.sh* script executes the each data loading script for HOSTINFO, OSINFO, HWINFO and DETAILCFGINFO tables.

So at the end of the successful completion of *sysinfomain.sh* script, the data from all the hosts are collected, sent to the master server and loaded into the Oracle database.

Hence as mentioned earlier, this is the only script, which initiates the data collection and loading. This script was scheduled in the master server's *cron* scheduler to run initially once in day. After observing for about five cycles of successful run, the frequency of the run was later reduced to once in week.

The selected portion of code for *sysinfomain.sh* and *getsysinfo.sh* and are presented in Appendix A.5

4.2.2. Data store layer coding

First the Oracle database and the database schema [6] was created as per the table structure designed. There were totally four scripts (*loadhostinfo.sh, loadosinfo.sh, loadnhwinfo.sh and loaddetailcfg.sh*) developed to load the data from text files to the current data tables. Each script does two main tasks. The first task is to create the SQL*Loader control file [2]. This control file controls the behaviour of the Oracle data loader command (*sqlldr*). The second task is to create the input data file from the output produced by the respective hosts. This input data file is used by the SQL*Loader control file for the INFILE parameter.

Two different programming logics are used to construct the input data file for the four current data tables. The first logic is used to load data for HOSTINFO table and second logic is used to load data for all other three tables (HWINFO, OSINFO and DETAILCFGINFO).

In the first logic the data for the HOSTINFO table is derived from the *sysinfo.conf* file. The Unix script, *loadhostinfo.sh* reads the *sysinfo.conf* file and creates a new input data file to match all columns of HOSTINFO table. Once the file is created the next task of creating the SQL*Loader control file is done by passing all control commands to a separate file. Then the script calls the SQL*Loader command (*sqlldr)* and pass the name of the control file as the argument. In turn the control file reads the input data file and all rows are loaded from the input data file.

But where as a different logic is used in building the input data file for all other three tables. This is because the data collected from any particular host are stored in separate directories. These directories are traversed in order to build the input data file for each table. So all three other data loading scripts (*loadosinfo.sh, loadhwinfo.sh, loaddetailcfg.sh)* have the same *for…do…don*e loop commands to build the input data file. The final input data file has one line for each host. A delimiter character separates the value of each column in the database. Once the input data file is constructed, the script creates the SQL*Loader control file for that particular table. As mentioned, passing all control commands to another separate temporary file does this part. The control file is passed as an argument for the *sqlldr* command and that in turn use the input data file to source the data.

But before these four data loading scripts are called to load the data, another script *copytables.sh* is executed to copy the data from current tables to the history tables. This simple script which calls the *sqlplus* command and pass the *insert* statement to insert the corresponding row in all four history tables by reading the respective current data tables.

The selected portion of code for *loadhostinfo.sh* and *loadosinfo.sh* are presented in Appendix A.5

4.2.3. Data retrieval layer

Once Apache web server [1] authenticates a user, the *index.php* program serves the first page to the web browser. Since the same program does both the presentation as well database connectivity tasks, it has both PHP code [10] as well as the HTML code [17] inside. The first or index page of the web site is basically the content extracted from the HOSTINFO table. The Oracle database connectivity is established by using the PHP Oracle 8 functions [11,12], which basically use the Oracle 8 Call-Interface (OCI8). The *OCIExecute* function of PHP executes the required SQL statement against the connected database and *OCIFetchInto* function call gets the results into an array variable (host). Once the HOSINFO table column values are fetched, using the HTML Form with *select* and *option* tags, a drop down list menu is generated. The selected value for the hostname variable is sent to the *getinfo.php* program. The HTML form uses the POST method to pass the hostname value to the *getinfo.php* program. The *getinfo.php* program uses the value of the hostname to extract all the information for that particular host.

The *getinfo.php* uses the same above-mentioned two OCI8 function calls (*OCIExecute*, *OCIFetchInto)* to query and store the results from HWINFO, OSINFO and DETAILCFGINFO tables. The extracted data from the database is presented in three HTML tables, which correspond to Hardware, Operating System and Detailed configuration information. This single program, which essentially displays all information about the selected host, is grouped into three different stages.

The first stage queries and fetches the data from the HWINFO table. A PHP array variable (hwinfo) is used to hold the result of the query obtained by the *OCIFetchInto* function call. The content of this variable is used to build and present the Hardware Information HTML table.

The second stage queries and fetches the data from the OSIFNO table. A PHP variable (osinfo) is used to hold the result of the query obtained again by the *OCIFetchInto* function call. The Operating System information is presented by using the content of this variable.

The third and final stage queries and fetches the data from the DETAILCFGINFO table. A PHP variable (detailinfo) is used to hold the result of the query. The Detailed Configuration Information HTML table is presented by using the content of this variable.

In all three HTML tables, the CLOB data type column has a hyper link provided against its corresponding heading. This hyper link calls another program *getclob.php* by passing values for three variables (hostname, table name and column name) as part of the Uniform Resource Locator (URL). Every hyper link passes different values for hostname, the database table name and the corresponding database column name for that particular heading. So the same *getclob.php* is used with different input values according to the table and column name selected.

So in summary, totally these three (*index.php*, *getinfo.php, getclob.php*) PHP programs (or rather it may be called as, HTML script with PHP code embedded) do the data extraction from the database and presentation to the users web browser for viewing the current data about the particular host, which is selected in the first page.

As part of the coding standard, all the PHP variables are used as lower case and the database columns are distinguished with upper case characters. Comments are inserted whenever necessary.

A sample portion of the code is presented in the Appendix A.5 for the *index.php*, *getinfo.php* and *getclob.php* programs.

5. TESTING

5.1. Testing approach

As decided while choosing the process model, the testing of each layer was done before starting the coding of next layer.

The other approach was to finish the coding of all three layers and start testing each program and scripts. But I believed that this approach could make the trouble shooting more difficult and any change of code in one of the layer would have a bigger impact on another layer programs. So hence I have decided that each layer must be tested first to confirm the proper functioning before proceeding to the next layer.

Each layer of coding was tested with the objective of making sure that it could deliver the required output to the next layer. Testing criteria were worked out for each layer and it was necessary that the each program to pass the criteria successfully without any errors.

All Unix shell scripts which were written for data collection and data store layers, were first run under shell debugging and verbose mode. This was to visually see and ensure that all statement in the code was executed at least one time without any errors. This approach was to do a 'white box' testing in addition to test for the desired data output produced by each script.

Once the individual layer testing was completed a complete integrated testing was done. In this test it was ensured that the overall system could function without any errors while meeting the test criteria. This integrated test was also to confirm that the system would be ready for a validation test or user acceptance test.

5.2. Testing and test results

5.2.1. Data collection layer

Testing objective of this layer was to confirm that the data were collected from the Unix servers and sent to the master server as per the format designed. Following table shows test conditions and results of testing for this layer.

Sl. No	Test	Expected result	Achieved result	Remarks
1	Run the *getsysinfo.sh* script in AIX, HP-UX and Solaris servers and verify it can produce proper output and conform to input file	Respective shell procedures should be selected for each OS and all necessary data are collected.	**PASS** Each server's output produced the desired format and all	

	format of Oracle SQL*Loader utility.		necessary data were collected without any omission.	
2	Run *sysinfomain.sh* script in master server.	*getsysinfo.sh* script should be sent to all three servers specified in the configuration file and should run the script from master server.	**PARTIALLY SUCCESSFUL** Could not send the *getsysinfo.sh* script to AIX server and collect output	It was found out that the AIX server did not allow the remote execution. Problem corrected after making right setup in the AIX server
3	Send the collected data to the master server to the respective directories	All three servers data should be available in the master server under the respective directories	**PASS** All collected data for three servers were available under the respective directories as per the data loading input file requirement.	Successful result of this test confirmed proper collection of data

5.2.2. Data store layer

The objective of testing this layer was to confirm that the collected data could be successfully loaded into the Oracle database.

Sl. no	Test	Expected result	Achieved result	Remarks
1	Read *sysinfo.conf* file and load into HOSTINFO table	All three samples records were to be loaded into database. No discard records or bad records should be reported.	**PASS** *sqlldr* utility successfully loaded all three records and checked with *sqlplus* utility	

2	Read collected hardware related data and load into HWINFO table	All three sample records to be loaded into database. No bad or discard records	**PASS** *sqlldr* utility successfully loaded all three records and checked with *sqlplus* utility	
3	Read collected OS related data and load into OSINFO table	All three sample records to be loaded into database. No bad or discard records	**PARTIALLY SUCCESSFUL** *sqlldr* utility successfully loaded all three records and checked with *sqlplus* utility	The data type defined in the database for one of the column did not match with the input data (column name LASTBOOT.) The data type was changed from DATE to VARCHAR2 and tested ok.
4	Read collected data and load into DETAILCFGINFO table	All three records to be loaded into database. No bad or discard records	**PASS** *sqlldr* utility successfully loaded all three records and checked with *sqlplus* utility	

5.2.3. Data retrieval layer

The objective of testing this layer was to confirm that the data stored in Oracle database could be successfully retrieved by the web application. This would also ensure that the Apache web server was configured properly.

Sl. No	**Test**	**Expected result**	**Achieved result**	**Remarks**
1	Apache web server user authentication	Web site to prompt user name and password	**PASS** Prompted the user name and accepted the set password	
2	List the available servers in the database and accept the user input.	Index page to show three servers details with a drop down menu to choose the server name to query	**PASS**	
3	Query the database with selected server name	Once the selected server name has been submitted, the next page should display the hardware, software and detailed information.	**PASS** In a single page all required data were shown.	
4	Check all the links provided for the detailed information.	The information displayed with respect to each items must correspond to the heading shown.	**PARTIALLY SUCCESSFUL**	It was noticed that one of the CLOB data type was not retrieved and. This was due to a typo error in PHP program and it was corrected

5.3. Validation

The validation test was performed to confirm that the system meet the users requirements. Once the final testing was over, the list of criteria for the validation test were finalised with users. The criteria were basically the sub-set of stated requirements at the beginning of the project

There were two users involved in validating the system against the initial requirement stated at the beginning of the project. The system was used and tested by them for about two to three days before a formal acceptance was carried out. After the three days of testing and verification by them separately, I have re-run and verified the below tests together with them to confirm the requirements are met.

During the test I also sought suggestions and feedback for improvement and their comments about the system.

Following table gives the user acceptance test criteria with results and feedback from users.

Sl. no	**Test**	**Expected result**	**Achieved result**	**Remarks/ User feedback**
1	User authentication a) Enter valid user name b) Enter in-valid user name	Should show the index page, if correct user name and password are entered. If not deny the access.	**PASS**	It was mentioned that it would be nice if user would be able to change the login password.
2	Add five servers in *sysinfo.conf* file and Index page should list all servers.	Should show one server in each row with all columns filled up.	**PASS**	
3	Proper values were displayed in the pull-down menu for servers list	All five servers should be shown in the pull-down menu	**PASS**	
4	Select any one server and view all information	To display hardware, OS and detailed configuration information in one page	**PASS**	
5	Check every item and verify it was the correct data for that item (example: number CPUs were correct, IP address was correct)	Should show the corresponding data for each item	**PASS**	As per the user feedback, the Last boot column data type was changed to accommodate number days since last boot.
6	Check each and every hyper links displayed and verify the details displayed were correct	Each hyper link should show corresponding CLOB data type value.	**PASS**	

7	Repeat step 4-6 for all five servers	Obtain the same results as in step 4-6	**PASS**	
8	Add one more server in *sysinfo.conf* file and check the data was collected for that server.	Data for the newly added server should be collected, stored and display in the web page	**PASS**	
9	Remove three servers from the configuration file and check ensure those servers data were removed from database	After re-running the collection script, the web site index page should not show the removed three servers	**PASS**	
10	Change one of the servers '/etc/hosts' file and check the change was seen in the web browser	After the change and data collection, the '/etc/hosts' file link should show new content.	**PASS**	

6. CRITICAL EVALUATION AND CONCLUSIONS

6.1. Project management

While I was able to complete the project close to originally planned time, but I must state that the coding and testing phase over ran by almost 25% than the originally planned 150 hours. This was purely due to my under estimation of time needed to learn both Oracle (including its utilities) as well as PHP. These two were totally new skill sets acquired by me during this project. However this overrun was compensated by earlier than expected time of completion of the 'Solution Decision' phase. I had realised that any new skill sets to be acquired or unfamiliar technologies introduced in a project could greatly affect the over all project schedule. Hence, if I have to re-draw this or draw another project schedule I would definitely take this as an important aspect for the overall project planning.

The software development life cycle model I have chosen had helped me to implement this project in a systematic way, but at end of the project I had realised that I could have chosen a different process model which might be slightly more appropriate for this project. The Waterfall model I had used did not explicitly provide a feedback loop[8] from one phase to other phase. During the coding and testing phase, I have faced few problems with respect to the database column lengths and data type selected. While I have fixed those problems but strictly speaking, this could be possible only with the provisioning of feedback loop from one phase to another phase. In this case, a change in the database design as the result of integrated testing had no provision according to the linear model. Also as per my approach of individual unit testing followed by the integrated testing deemed to fit very closely in the V model than the Linear model. Hence for both these reasons, I think I should have chosen the V model rather the linear model.

6.2. Design and choice of tools

I had realised that the idea of using remote shell (*remsh*) from the master server to send and run the data collection script had created a major weakness in terms of security to the systems. While I still believe it is a good idea to have the data collection script in the master server (as I did so), but I think I should have at least used secure shell (*ssh*) environment to implement the data collection process. But it could also be said that if any host wants to participate in this information system, then it should trust the master server. Hence in my view the idea of choosing the *remsh* was not necessarily a wrong idea but if I have to do it again, I would probably do it with secure shell environment.

As regard to the data store layer, I still believe that the Oracle database as the data repository had worked well for me and at no time during this project I regretted

[8] "Although the original waterfall model proposed by Winston Royce made provision for feedback loops, but in practice this process model treats it as if it were strictly linear". (Roger S. Pressman (2001) *SEPA, 5/e.* McGraw-Hill International edition.)

in choosing Oracle database. However, I had observed during the course of this project work that the resource and knowledge available for PHP-MySql combination to be very rich and plenty, when compared to PHP-Oracle combination. But Oracle's SQL*Loader utility's flexibility and power was one of the strong point which had helped me to make the data loading into the database much more ease and robust. On this aspect I still doubt whether I could have the same functionality with MySql. However I must acknowledge that I had not done very detailed study at the beginning of the project to make a right and fair conclusion on data loading options and utilities between these two database systems. Hence if I have to do it again, I would first give a deeper look at MySql data loading options and LOB data type support, and if it is found equal to Oracle and then would probably choose MySql instead of Oracle.

As far as the data retrieval layer is concerned, I totally believe the Apache and PHP was a right combination and I do not think I would want to choose any other combination or programming language. But there was one major weakness in user authentication implementation. While I was aware of the sensitivity of nature of the data carried by the web site and I have implemented the default Apache's user authentication mechanism, but this proved to be practically far from perfect. This authentication mechanism required the web administrator to create the login account and set the password. I have not catered or designed to provide some mechanism to allow users to change their passwords. While the creation of account could be still done by the administrator, the users wanted that the account password is set or changed by them. In that case, PHP and database based authentication with the options of password changing feature would be a better choice than the basic HTTP authentication.

6.3. Testing and validation

I believe the testing approach I had taken, has worked well for me. This was evident from the fact that I was able to fix most of the problems during my unit testing and hence the integrated testing was by and large went without major problems. This approach also helped to pass the validation test the users without much issues.

Even though I was able to deliver the system as per initial agreed requirement and met the user expectation on the out come of the system, but this system has failed to meet the users expectation on the user authentication mechanism (as discussed in chapter 6.2). This could have been clearly avoided if I had discussed in detail about this requirement initially and taken care during the design. The part of the problem also due to my assumption that the basic authentication would be enough to restrict the access to the data stored in the master server. I have realized that any assumptions in user requirement should be avoided or otherwise to be clarified, to avoid this kind of situations.

6.4. Overall conclusions

Overall this project achieved its primary aim of providing the Unix servers information over the web, which was found very useful to the intended users of this information system.

But this system could have been improved at least in following few ways and provide more functionalities than just providing the current system configuration. First, a search function to look for specific pattern for few columns like type of operating system, version of operating system and vendor name to filter based on user's criteria to view the systems. Second, provide some means to query the historical data in addition to the current data. Third as a way of tracking changes happened in the system configuration over the period of time.

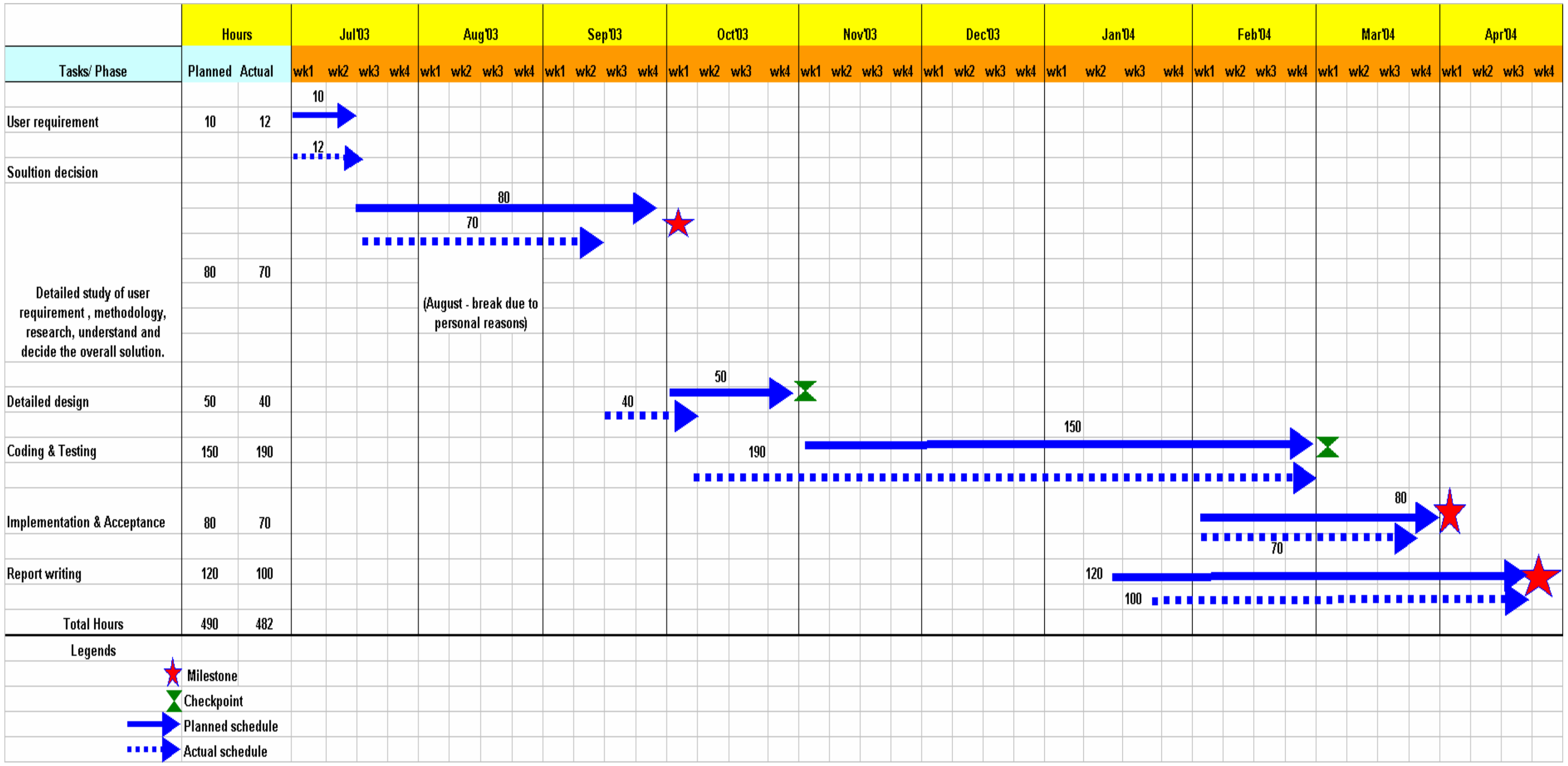
However, considering the given reasonable time for this project and having met the users requirement and expectations, this system can be considered successful and serves its purpose.

Bibliography

1. Mohammed J. Kabir (2002) *Apache Server 2 Bible*. Hungry Minds, Inc. USA.
2. Oracle 8i Server and SQL*Plus, *Documentation CD-ROM.* Oracle Corporation.
3. Roger S. Pressman, *Software Engineering (SEPA, 5/e)*, McGraw-Hill International edition.
4. Thomas Connolly & Carolyn Begg (2000) *Database Solutions*. Addison-Wesley
5. Ramus Lerdorf & Kevin Tatroe (2002) *Programming PHP*. O'Reilly & Associates
6. Michael Wessler (2001) *Oracle DBA on Unix and Linux*. SAMS
7. Centre for Technology in government, *A Survey of System Development Process Models*, http://www.ctg.albany.edu/publications/reports/survey_of_sysdev
8. The Standish group, http://www.standishgroup.com
9. Word Wide Web: *PHP: Hypertext Preprocessor*, http://www.php.net/
10. World Wide Web: *PHP Help*:*PHP Freaks.com* http://www.phpfreaks.com/
11. World Wide Web: *PHP Conference Material Site*, http://conf.php.net/
12. World Wide Web: *Webmonkey Programming PHP & Using Oracle with PHP*, http://hotwired.lycos.com/webmonkey/programming/php/
13. World Wide Web: *HP Technical documentation*, http://docs.hp.com/
14. World Wide Web: *Sun Product Documentation*, http://docs.sun.com/
15. World Wide Web*: IBM AIX operating system: Library*, http://www-1.ibm.com/servers/aix/library/index.html
16. World Wide Web: *Apache HTTP Server Project*, http://httpd.apache.org/
17. World Wide Web: *HTML Tutorial*, http://www.2kweb.net/html-tutorial/
18. World Wide Web: *MySQL*, http://www.mysql.com/

Appendix

A.1 Project plan



Tasks/ Phase	Planned	Actual
User requirement	10	12
Soultion decision		
Detailed study of user requirement , methodology, research, understand and decide the overall solution.	80	70
Detailed design	50	40
Coding & Testing	150	190
Implementation & Acceptance	80	70
Report writing	120	100
Total Hours	490	482

A.2 User requirements – Details of data captured from servers

Following are the details of each data items captured under each category. Each category listed here corresponds to one database table in the Oracle database.

- **Host information**
  1. Hostname
  2. Application name
  3. Application type
- **Operating system information**
  1. uname
  2. OS name
  3. Version
  4. OS mode (32 or 64 bit)
  5. Installed patches list (file)
  6. Last boot
  7. Installed Software list (file)
- **Hardware information**
  1. Vendor
  2. Model
  3. Serial number
  4. 64bit capable
  5. CPU type
  6. Total number of CPU
  7. Physical memory
  8. Total disk space
  9. Tape drives (file)
  10. IO adapters (file)
  11. Hardware scan output (file)
  12. External storage Connection
- **Detail Configuration information**
  1. IP address
  2. Networking details (file)
  3. Bootlist info (file)
  4. Filesystem details (file)
  5. LVM details (file)
  6. Fstab(file)
  7. Hosts(file)
  8. Services(file)
  9. Inetd.conf(file)
  10. Passwd(file)
  11. Group(file)

A.3 System hardware and software packages used for this project

One HP-UX master server was used for this project with following hardware and software packages. The operating system was re-installed and additional software packages were installed using the vendor supplied software distribution media.

- Hewlett-Packard PA-RISC based K-class HP-UX server
- HP-UX version 11.00
- Korn Shell environment
- HP Apache based web server with PHP – Product part number B9416AA
- Oracle server – version 8.1.7

There were totally three types of client Unix servers (one for each operating system) were used to collect the data and test the information system. Following are the operating system detail of each host, which were used to test and implement this information system.

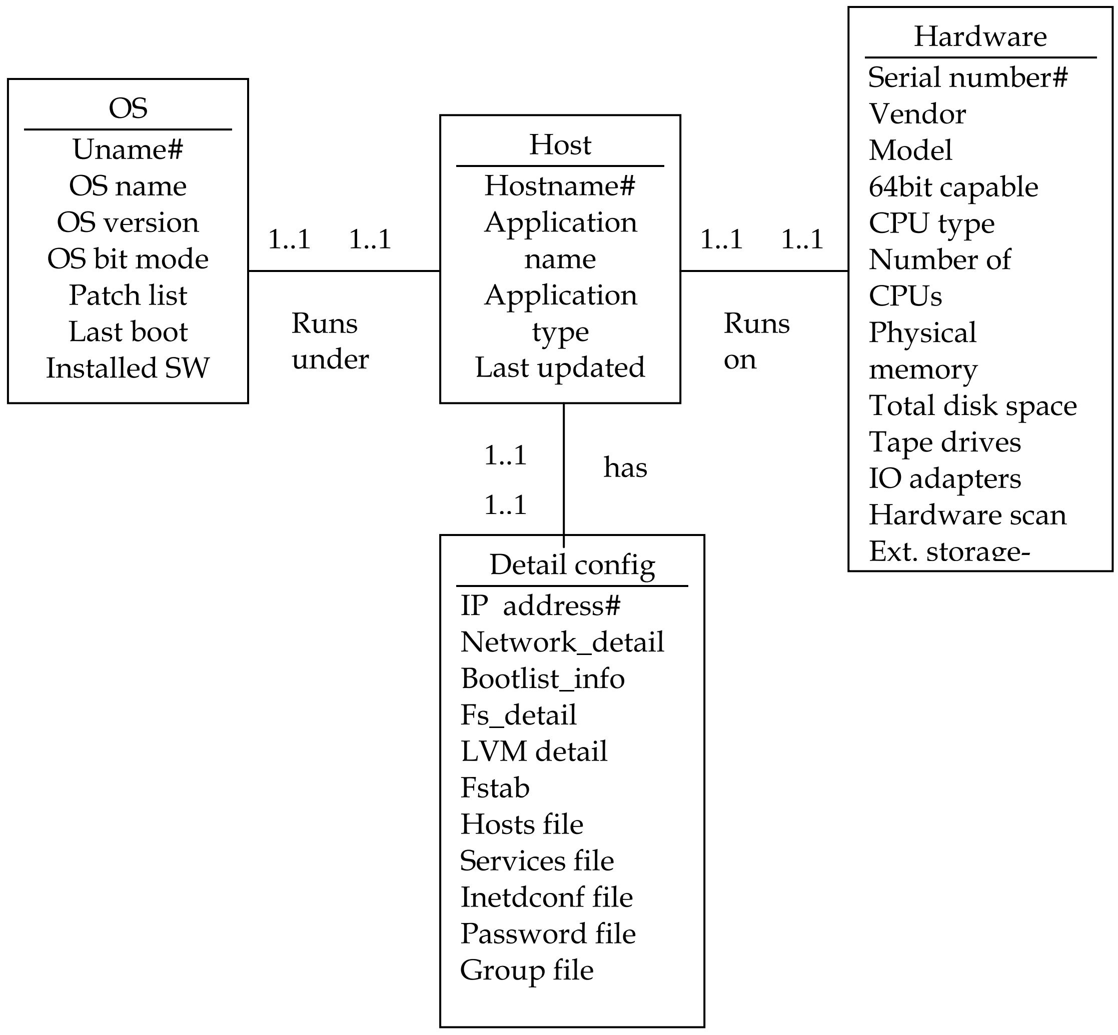
- IBM AIX version 5.2
- Hewlett-Packard HP-UX 11.00
- SUN Solaris 8 (SunOS 2.8)

A.4 Database design

Entity Relationship diagram

Below diagram shows entity relationship modeling used to arrive the database schema.

Each computer system called host, runs on only one hardware and under only one operating system. The host has detailed configuration information in order to support the application environment. The host, hardware, operating system and detailed configuration have various items or properties as mentioned in the below diagram. The properties under each entity correspond to the each data item for that category (this is as per the users requirement, which is presented in Appendix A.1)



Database design (continued…)

Oracle Database tables structure

a) Current data tables

Table name : HOSTINFO (Represents Host entity)

Column name	Data type	Remarks
HOSTNAME#	VARCHAR2(8)	PK
APPLNAME	VARCHAR2(15)	
APPLTYPE	VARCHAR2(15)	
UPDATED	DATE	

Table name : HWINFO (Represents Hardware entity)

Column name	Data type	Remarks
SERIALNUMBER#	VARCHAR2(15)	PK
HOSTNAME#*	VARCHAR2(8)	FK (Not null and no duplicate) References hostinfo
VENDOR	VARCHAR2(8)	
MODEL	VARCHAR2(10)	
CAPABLE64BIT	VARCHAR2(3)	
CPUTYPE	VARCHAR2(8)	
NUMBEROFCPU	NUMBER(2)	
TOTALMEMORYMB	NUMBER(6)	
TOTALDISKSPACEGB	NUMBER(4)	
TAPEDRIVES	CLOB	
IOADAPTERS	CLOB	
HARDWARESCANOUTPUT	CLOB	
EXTSTORAGECONN	VARCHAR2(8)	

Table name : OSINFO (Represents OS entity)

Column name	Data type	Remarks
UNAME#	VARCHAR(8)	PK
HOSTNAME#*	VARCHAR(8)	FK (Not null and no duplicate) References hostinfo
OSNAME	VARCHAR2(8)	
OSVERSION	VARCHAR2(8)	
OSBITMODE	CHAR(2)	
PATCHLIST	CLOB	
LASTBOOT	VARCHAR2(15)	
INSTALLEDSOFTWARELIST	CLOB	

Table name : DETAILCFGINFO (Represents Detailed Configuration entity)

Column name	Data type	Remarks
IPADDRESS#	VARCHAR(15)	PK
HOSTNAME#*	VARCHAR(8)	FK(Not null and no duplicate) References hostinfo
NETWORKDETAIL	CLOB	
BOOTLISTINFO	CLOB	
FSDETAIL	CLOB	
LVMDETAIL	CLOB	
FSTABFILE	CLOB	
HOSTSFILE	CLOB	
SERVICESFILE	CLOB	
INETDCONFFILE	CLOB	
PASSWDFILE	CLOB	
GROUPFILE	CLOB	

b) History Data tables

Table name : HOSTINFO_HIST (Corresponds to HOSTINFO table)

Column name	Data type	Remarks
HOSTNAME#	VARCHAR2(8)	Composite key
DATED#	DATE	Composite key
APPLNAME	VARCHAR2(15)	
APPLTYPE	VARCHAR2(15)	

Table name : HWINFO_HIST (Corresponds to HWINFO table)

Column name	Data type	Remarks
SERIALNUMBER#	VARCHAR2(15)	Composite key
DATED#	DATE	Composite key
HOSTNAME#*	VARCHAR2(8)	FK (Not null and no duplicate) References hostinfo_hist
VENDOR	VARCHAR2(8)	
MODEL	VARCHAR2(10)	
CAPABLE64BIT	VARCHAR2(3)	
CPUTYPE	VARCHAR2(8)	
NUMBEROFCPU	NUMBER(2)	
TOTALMEMORYMB	NUMBER(6)	
TOTALDISKSPACEGB	NUMBER(4)	
TAPEDRIVES	CLOB	
IOADAPTERS	CLOB	
HARDWARESCANOUTPUT	CLOB	
EXTSTORAGECONN	VARCHAR2(8)	

Table name : OSINFO_HIST (corresponds to OSINFO table)

Column name	Data type	Remarks
UNAME#	VARCHAR(8)	Composite key
DATED#	DATE	Composite key
HOSTNAME#*	VARCHAR(8)	FK (Not null and no duplicate) References hostinfo_hist
OSNAME	VARCHAR2(8)	
OSVERSION	VARCHAR2(8)	
OSBITMODE	CHAR(2)	
PATCHLIST	CLOB	
LASTBOOT	VARCHAR2(15)	
INSTALLEDSOFTWARELIST	CLOB	

Table name : DETAILCFGINFO_HIST (Corresponds to DETAILCFGINFO table)

Column name	Data type	Remarks
IPADDRESS#	VARCHAR(15)	Composite key
DATED#	DATE	Composite key
HOSTNAME#*	VARCHAR(8)	FK (Not null and no duplicate) References hostinfo_hist
NETWORKDETAIL	CLOB	
BOOTLISTINFO	CLOB	
FSDETAIL	CLOB	
LVMDETAIL	CLOB	
FSTABFILE	CLOB	
HOSTSFILE	CLOB	
SERVICESFILE	CLOB	
INETDCONFFILE	CLOB	
PASSWDFILE	CLOB	
GROUPFILE	CLOB	

A.5 Listing of selected portion of code from various programs

sysinfomain.sh : Data collection Layer – Main script

```
################## Begin data collection from all hosts
      for HOST in `cat $CONFFILE | awk -F+ '{print $1}'`
        do
          rm -r $DATADIR/$HOST
          mkdir -p $DATADIR/$HOST
          chmod 777 $DATADIR/$HOST
          remsh $HOST "mkdir -p $REMOTEDIR"
          rcp $LOCALDIR/$COLLECTCMD $HOST:$REMOTEDIR/$COLLECTCMD
          remsh $HOST "chmod 700 $REMOTEDIR/$COLLECTCMD"
          remsh $HOST "$REMOTEDIR/$COLLECTCMD"
        done
############### End of data collection
```

getsysinfo.sh : Data collection layer – Actual data collection script

```
AIX5hwinfo()
{
Hostname=`hostname`
Vendor=`uname -M | awk 'BEGIN {FS=","} {print $1}'`
Model=`uname -M | awk 'BEGIN {FS=","} {print $2}'`
Serial=`uname -u | awk 'BEGIN {FS=","} {print $2}'`
CPUTYPE=`getsystype -y`

if [ $CPUTYPE = 64 ];
 then
   Cputype64=Yes
 else
   Cputype64=No
fi
…
…
DSPACE=0
for i in `lsvg -o | grep -v rootvg`
 do
      ((DSPACE=DSPACE + `lsvg $i | grep "TOTAL PPs" | awk '{print $7}' | cut -d "("-f 2`))
 done

…
…

TotalDiskspace=`expr $DSPACE / 1024`
print "$Hostname | $Vendor | $Model | $Serial | $Cputype64 | $Proctype | $TotalCPU | $Physical
memory | $TotalDiskspace | $Tapedrives | $IOAdapters | $HardwareScanoutput | $ExtStorageCo
nn" > $OUTPUTDIR/hwinfo.out
}
```

loadhostinfo.sh - Data loading script for uploading the data into HOSTINFO table

```
…
…
cat $INPUTFILE  |while read LINE
 do
   echo $LINE+`date "+%d %b %y"` >>$OUTFILE
 done

# Constract the control file now

echo "
load data
infile '$OUTFILE'
badfile '$SQLBAD'
replace
into table Hostinfo
FIELDS TERMINATED BY \"+\"
(HOSTNAME,APPLNAME,APPLTYPE,UPDATED) " >$SQLFILE

# Now call the sql loader

su - oracle -c "sqlldr userid=$USER/$PASS control=$SQLFILE log=$SQLLOG"  >/dev/null

# End of script
```

loadosinfo.sh - Data loading script for uploading the data into OSINFO table

```
…
…
# Contruct the infile for the sql loader

>$OUTFILE
>$SQLLOG
>$SQLBAD

for HOST in `cat $CONFFILE|awk -F+ '{print $1}'`
 do
    cat $DATADIR/$HOST/$INDATFILE >>$OUTFILE
 done

# Construct the control file now

echo "
load data
infile '$OUTFILE'
badfile '$SQLBAD'
replace
INTO TABLE OSINFO
FIELDS TERMINATED BY \"|\"
TRAILING NULLCOLS
(
HOSTNAME,UNAME,OSNAME,OSVERSION,OSBITMODE,
EXT_FILE1  FILLER CHAR(100),
PATCHLIST lobfile(EXT_FILE1) terminated by EOF,
LASTBOOT,
EXT_FILE2  FILLER CHAR(100),
INSTALLEDSOFTWARELIST lobfile(EXT_FILE2) terminated by EOF
)  " >$SQLFILE

# Now call the sql loader

su - oracle -c "sqlldr userid=$USER/$PASS control=$SQLFILE log=$SQLLOG" >/dev/null

# End of script
```

index.php – Data retrieval layer – Index page

```
…
…
<?
require('./dbinfo.inc');
$connection = OCILogon(USER,PASS,DB);
$query="select * from hostinfo order by hostname";
$stmt = OCIParse($connection, $query);
OCIExecute($stmt);
?>
…
…
<form ACTION = "getinfo.php" method=POST >
<select name="hostname">
<?
while(OCIFetchInto($stmt, &$host, OCI_ASSOC))
  {
?>
<option value=<?= $host['HOSTNAME'] ?> > <?= $host['HOSTNAME'] ?> </option>
<? }
?>
</select>
<input TYPE="Submit" Value="Submit">
</form>
```

getinfo.php – Data retrieval layer – Retrieve data from Oracle database tables

```
<? echo $hostname;

/// Query and Display the HW info

require('./dbinfo.inc');
$connection = OCILogon(USER,PASS,DB);
$query="select * from hwinfo where hostname='$hostname'";
$stmt = OCIParse($connection, $query);
OCIExecute($stmt);
OCIFetchInto($stmt, &$hwinfo, OCI_ASSOC)
?>
<br><p align=center><font color="blue">Hardware Information</font><br>
<table border="4" width=600 align=center cellspacing="0">
<tr>
<td width=65%><font color="red"><b>Vendor</b></font></td>
<td width=35%><b><?= $hwinfo['VENDOR'] ?></b></td></tr>
<tr>
<td ><font color="red"><b>Model</b></font></td>
<td><b><?= $hwinfo['MODEL'] ?></b></td></tr>
<tr>
…
…
  <tr>
<td ><font color="red"><b>Hardware Scan output</b></td>
<td><b><a href="getclob.php?host=<? echo $hostname
?>&tab=hwinfo&col=HARDWARESCANOUTPUT">Click here</a></td></tr>
…
…
```

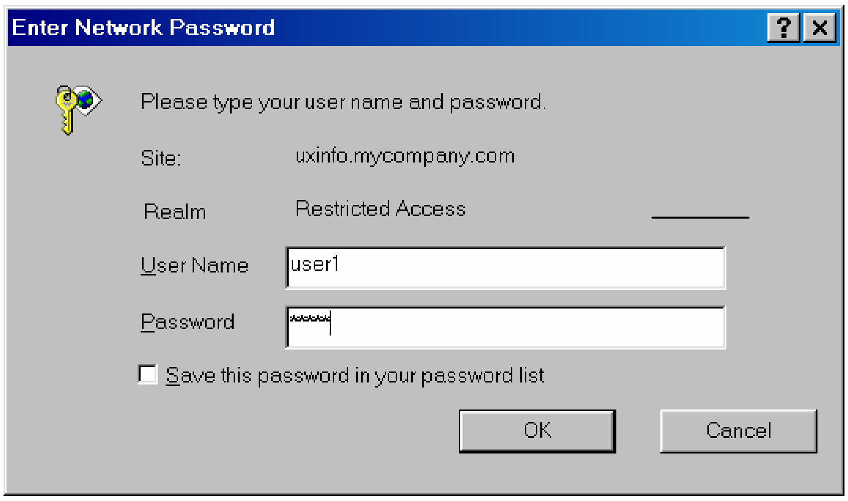
getclob.php – Data collection layer: Retrieve the CLOB data type

```
<html>
<body>
<title>Unix Servers Information</title>
<body background="images/pink_fabric.gif">
<p align=center><font color="red" size="+1"><b>Hostname:</b></font>
<b><? echo $host ; ?></b>
</p>
<?
require('./dbinfo.inc');
$connection = OCILogon(USER,PASS,DB);
$query = "select $col from $tab where hostname='$host'";
$stmt = OCIParse($connection, $query);
OCIExecute($stmt);
OCIFetchInto($stmt, &$clobinfo, OCI_ASSOC);
$CLOBDATA=($clobinfo["$col"]->load());
?>

<PRE><? echo $CLOBDATA ?></PRE>
</body>
</html>
```

A.6 Selected screen shots of web browser

(**Note**: Fictitious company name and URL is used to maintain the confidentiality of the organization. Also the server serial number and IP address have been masked to maintain the confidentiality)



Please type your user name and password.

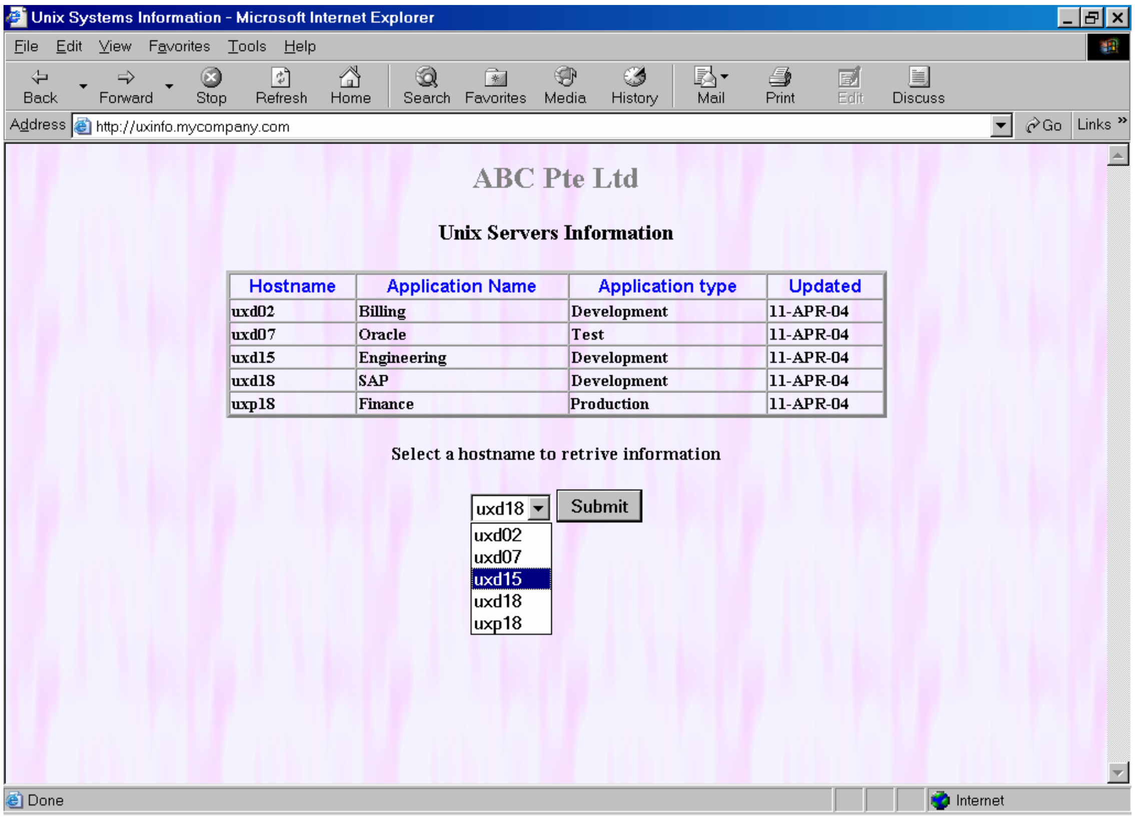
Site: uxinfo.mycompany.com

Realm Restricted Access

User Name user1

Password *****

☐ Save this password in your password list


Unix Systems Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit Discuss
Address http://uxinfo.mycompany.com
Go Links
Done
Internet

ABC Pte Ltd

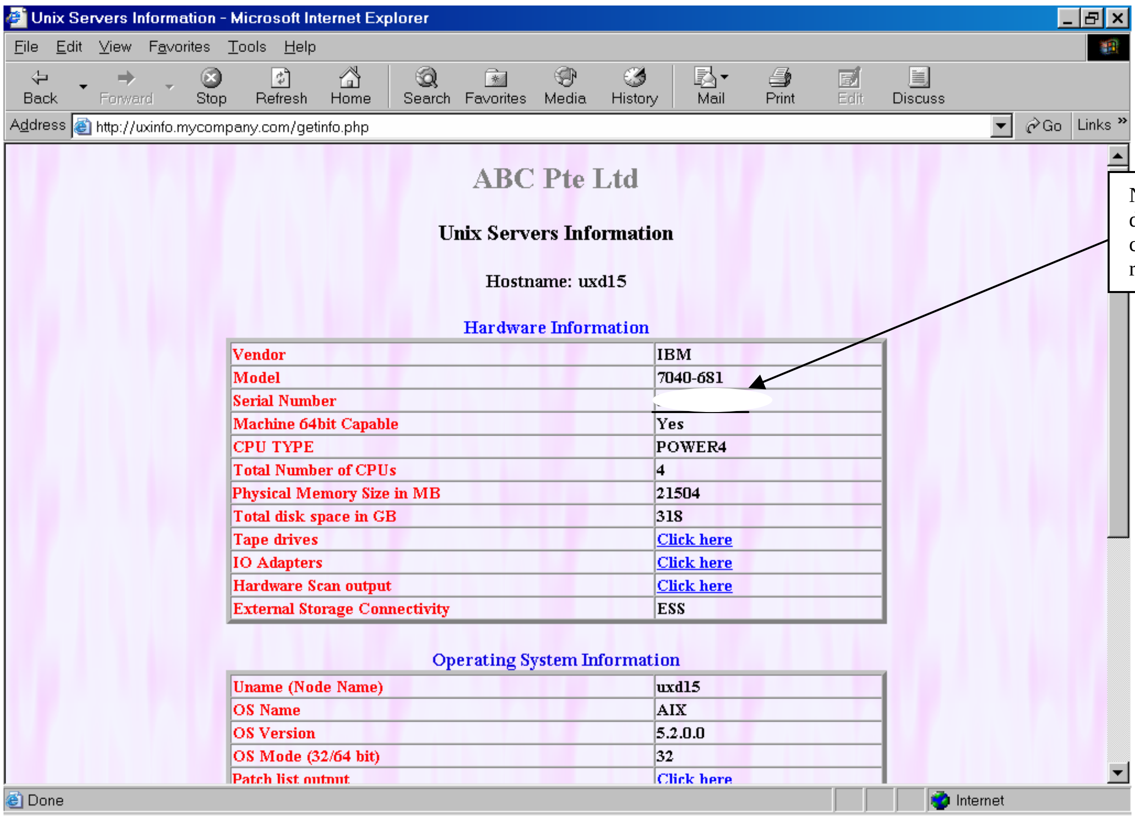
Unix Servers Information

Hostname	Application Name	Application type	Updated
uxd02	Billing	Development	11-APR-04
uxd07	Oracle	Test	11-APR-04
uxd15	Engineering	Development	11-APR-04
uxd18	SAP	Development	11-APR-04
uxp18	Finance	Production	11-APR-04

Select a hostname to retrive information

uxd18 ▾ Submit

uxd02
uxd07
uxd15
uxd18
uxp18


Unix Servers Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit Discuss
Address http://uxinfo.mycompany.com/getinfo.php
Go Links
Done
Internet

ABC Pte Ltd

Unix Servers Information

Hostname: uxd15

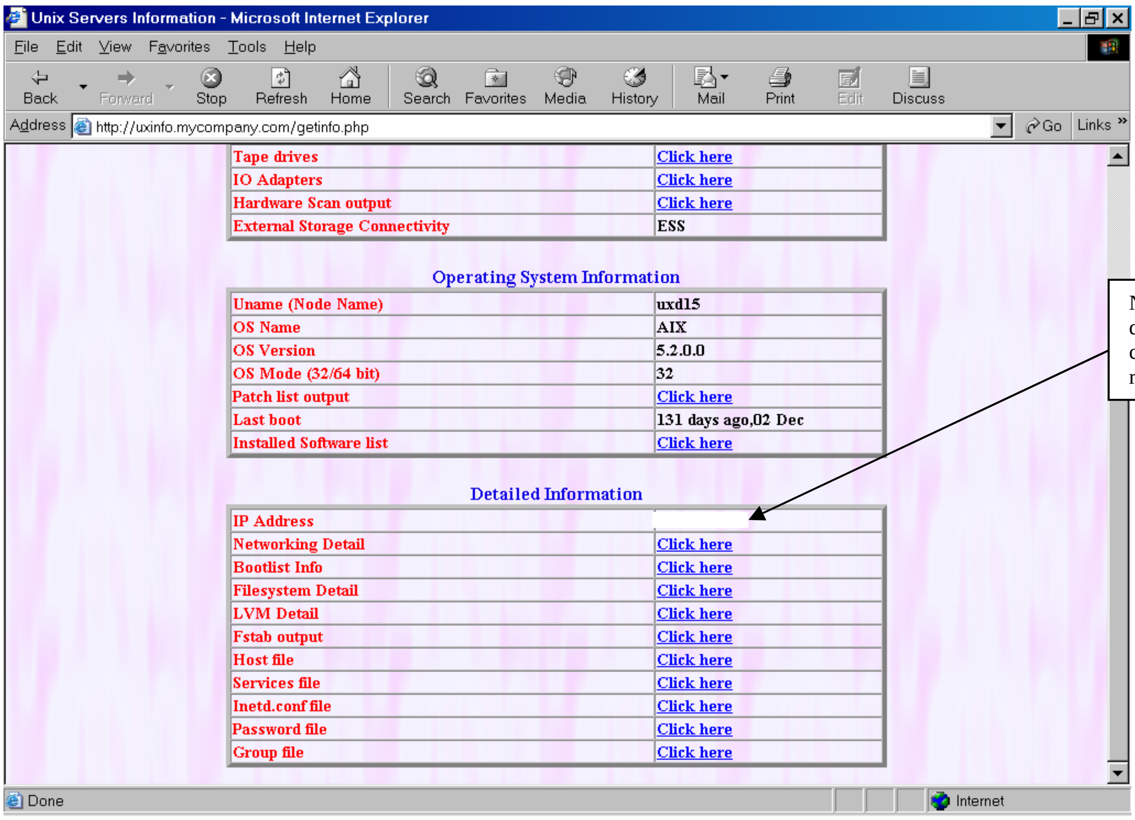
Hardware Information

Vendor	IBM
Model	7040-681
Serial Number	
Machine 64bit Capable	Yes
CPU TYPE	POWER4
Total Number of CPUs	4
Physical Memory Size in MB	21504
Total disk space in GB	318
Tape drives	Click here
IO Adapters	Click here
Hardware Scan output	Click here
External Storage Connectivity	ESS

Operating System Information

Uname (Node Name)	uxd15
OS Name	AIX
OS Version	5.2.0.0
OS Mode (32/64 bit)	32
Patch list output	Click here

Not disclosed due to confidentiality reason


Unix Servers Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit Discuss
Address http://uxinfo.mycompany.com/getinfo.php
Go Links
Done
Internet

Tape drives	Click here
IO Adapters	Click here
Hardware Scan output	Click here
External Storage Connectivity	ESS

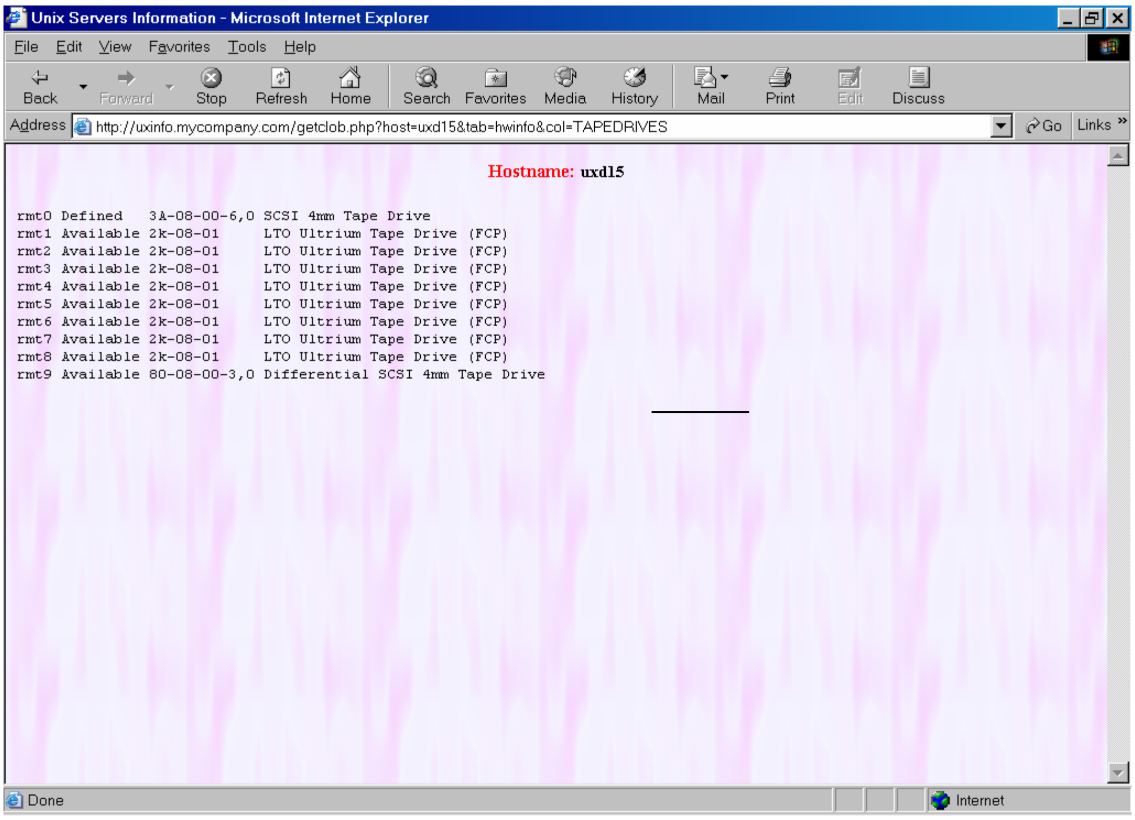
Operating System Information

Uname (Node Name)	uxd15
OS Name	AIX
OS Version	5.2.0.0
OS Mode (32/64 bit)	32
Patch list output	Click here
Last boot	131 days ago,02 Dec
Installed Software list	Click here

Detailed Information

IP Address	
Networking Detail	Click here
Bootlist Info	Click here
Filesystem Detail	Click here
LVM Detail	Click here
Fstab output	Click here
Host file	Click here
Services file	Click here
Inetd.conf file	Click here
Password file	Click here
Group file	Click here

Not disclosed due to confidentiality reason


Unix Servers Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit Discuss
Address http://uxinfo.mycompany.com/getclob.php?host=uxd15&tab=hwinfo&col=TAPEDRIVES
Go Links
Done
Internet

Hostname: **uxd15**

```
rmt0 Defined   3A-08-00-6,0 SCSI 4mm Tape Drive
rmt1 Available 2k-08-01     LTO Ultrium Tape Drive (FCP)
rmt2 Available 2k-08-01     LTO Ultrium Tape Drive (FCP)
rmt3 Available 2k-08-01     LTO Ultrium Tape Drive (FCP)
rmt4 Available 2k-08-01     LTO Ultrium Tape Drive (FCP)
rmt5 Available 2k-08-01     LTO Ultrium Tape Drive (FCP)
rmt6 Available 2k-08-01     LTO Ultrium Tape Drive (FCP)
rmt7 Available 2k-08-01     LTO Ultrium Tape Drive (FCP)
rmt8 Available 2k-08-01     LTO Ultrium Tape Drive (FCP)
rmt9 Available 80-08-00-3,0 Differential SCSI 4mm Tape Drive
```