University of Portsmouth

Computing & Mathematics Programme Area

Final year project undertaken in partial fulfilment of the requirements
for the BSc (Honours) Degree in Business Information Systems

Grid Monitoring: An Extendable and Secure Approach

by

Matthew Grove

Supervisor: Dr. Mark Baker

Project unit: PJE40

May 2003

Abstract

Projects are underway to provide distributed processing over a wide area. They are aiming toward allowing seamless access to globally distributed resources.

In any distributed infrastructure, there is a need to monitor and manage the resources that exist within the system. While the monitoring needs of the Grid can be specified, the kinds of information being monitored can be diverse - from streaming data that indicates the state of a running process, to providing meta-data about the use of a Grid site. This information must be delivered in a, secure, timely manor across heterogeneous systems.

The project explores the development of an extendable system, leveraging existing monitoring technologies to provide a distributed, transparent means to monitor resources within a Grid environment. A prototype is designed and developed to both prove the validity of the monitoring architecture and overcome the implementation issues with developing scalable, robust Grid software. Abstraction and encapsulation of the system components create a structured API, while the use of dynamic code loading allows for an extendable and flexible implementation.

Contents

List of Figures

Chapter 1

Introduction

1.1 The Grid and monitoring

A Grid consists of a federation of networked resources typically found in virtual groups. The Grid can provide a gateway to submit computational jobs and other work, which will be run on resources allocated by remote or local schedulers. It can be thought about as giving users access to a global virtual computer. This makes better use of the computer-based resources, which may stand idle and under utilized, and allows sharing of powerful resources (such as super computing platforms). A virtual view of one user interacting with a Grid is shown in Figure 1.1.

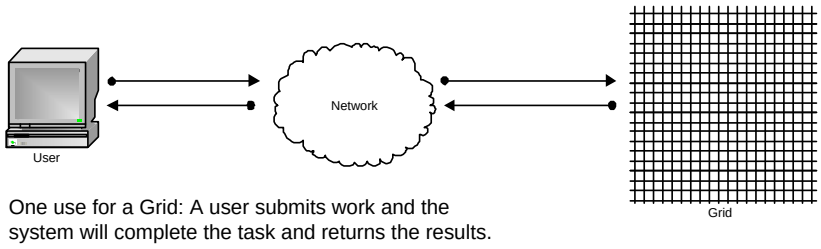

Figure 1.1: A virtual view of a user interacting with a Grid

The Grid itself needs a mechanism for monitoring. By monitoring we mean:

- A way of gathering data about the state of the Grid
- Processing this data

- Delivering it to agents and clients, which can make use this information.

In order to gather this information, we first need to decide what data is needed and for what it is used.

The Grid is basically made up of:

- A series of nodes which are capable of doing work,
- Agents which control the nodes,
- End users who want to make use of the resources,
- Networks which provide the infrastructure that link the resources together.

These components are shown as part of a physical view of Grid infrastructure in Figure 1.2.

Potentially each of these components, which can be divided into infrastructure, users and resources can generate data and each could require it: they are both producers and consumers of Grid information. The two key consumers are the control agents and the end users. The agents require information to be able to make decisions when controlling the Grid, end users might be interested in monitoring a resource running one of their jobs (perhaps in real time). There is a need to provide the information in context; for example, if information is required in a given time frame and the system delivers it late, it is of no use to the agent. Another aspect to look at is the need for historical data; an agent may require information on a previous state of the Grid. The project focus is on the initial monitoring of the data, however it is expected that the architecture would support a client capable of recording monitoring data for future use. Examples of this information:

- Jobs currently running on a node,
- The capabilities of a resource (for example, CPU speed, memory, disk),
- A queue of tasks held by an agent,
- The number of resources being controlled by an agent.

The information that can be collected is quite diverse. Some can be considered static (for example, the name of a node), while some has the potential to change rapidly (for example, CPU usage). The initial stage of the monitoring process is discovering what data is

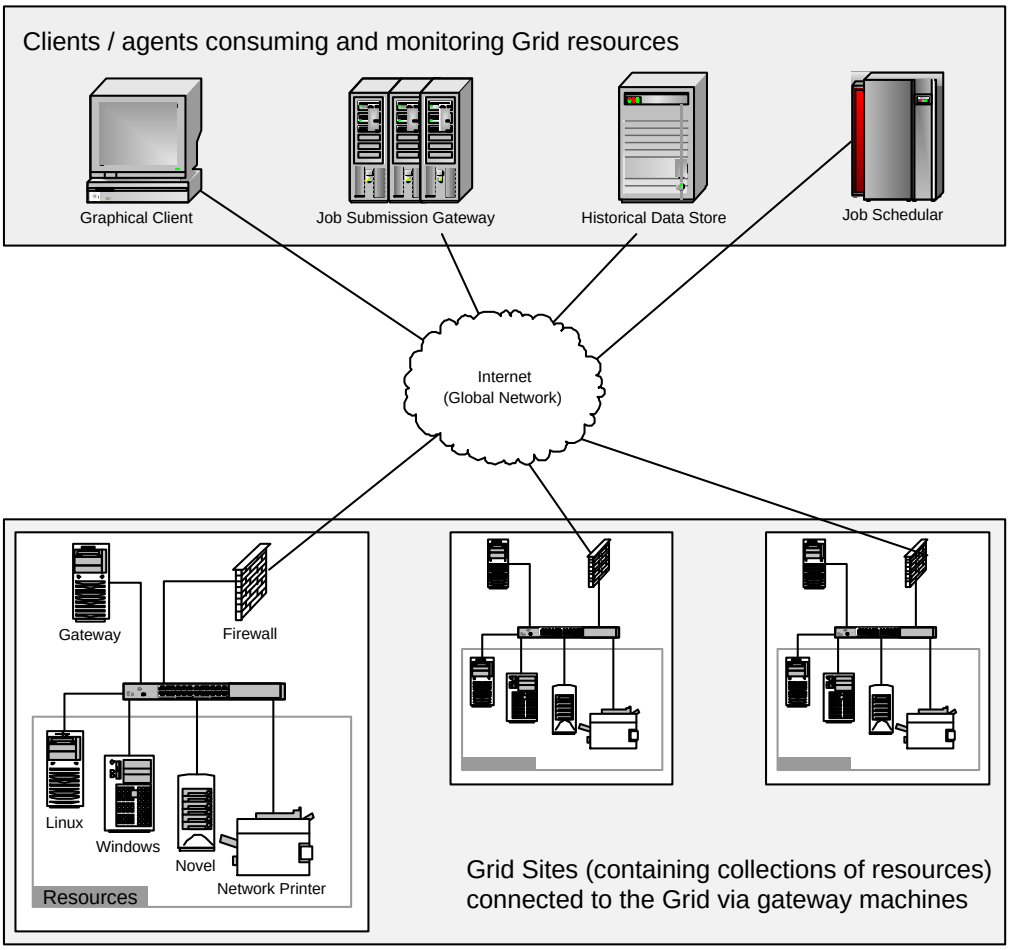

Figure 1.2: Physical view of a Grid

available and then how it can be monitored. This is non-trivial, the problem lies within the heterogeneous nature of the Grid. It is a heterogeneous mix of different architectures and operating systems; there is no standard, which means at the moment each site is choosing a monitoring method to meet its needs. This is a problem for non-local users of a site, as they will probably not have the tools required, or the knowledge needed to find and monitor the information they require.

The Grid concept is to provide a virtual computing platform. The Grid should not be tied to one platform, as there is no one operating system or architecture that meets all needs. We are left with the problem of monitoring systems which all provide different data over different interfaces. Abstracting the monitoring to provide standardisation at another level beyond the initial gathering of the data is required. Solving this problem will be a major part of the project. Ways that others have tried to implement heterogeneous monitoring

along with descriptions of monitoring techniques will be included in the literature review of the project.

1.2 Security

Inter-site communication within the Grid is typically over the Internet - it provides the distributed network, which allows physically separate networks to act as one virtual Grid. The distributed resources within the Grid must be protected. Within the context of monitoring - access to the information being monitored must be controlled. Some of the information is likely to be of use to potential hackers or people wanting to gain access to a site on the Grid. A distributed platform of this size is an interesting intellectual target as well as a powerful resource to abuse.

In many cases, Grid sites are protected by firewalls. This provides a gateway into the site and may hide the nodes from the wider Internet. Network administrators are typically implementing egress (out going) filters on their firewalls, which limit the ports their users can use. This can present a problem to system designers attempting to allow access to servers, which are not using standard protocols and ports. Traversing the firewalls is a problem, which will have to be addressed by this project, as end users and Grid sites will need to have information passed between them. There are common techniques for doing this.

The mechanism by which the monitoring system distributes the information should be secure. Data sent as plain text can be passively read by any part of the Internet it is routed through. In recent years the standardisation of various encryption techniques, for example public private key [1], hashes, and certificates, means that it is possible to use standard protocols such as SSL (Secure Socket Layer) [2] to encrypt data using industry tested algorithms (i.e. X.509 certificates [3]).

Access control is another requirement of a secure system. An administrator will want to restrict access to the resources within the site they control. There needs to be a way to have the identification of the users roam across the Grid. If a user enters the Grid monitoring system through a gateway, they can authenticate who they are to that gateway. If they then want to monitor a resource at another site, the gateway must pass the identification of the user along with the request for information to allow the second site to check whether the user has the permissions to monitor the resource. The home gateway agent does not know about these remote permissions, it can only authenticate its own users. This is a distributed model, which is compatible with the way users have traditionally been granted access to resources.

1.3 Distributed

The Grid by definition is spread across worldwide, networked infrastructures. An ideal distributed system has no single point of failure and parts of the system should be able to fail without breaking the rest. The classic example is the Internet itself, the failure of a node or even large parts of the network do not bring the entire system down. There are some standard problems with designing distributed systems, avoiding single points of failure creates problems when the system needs to discover entities.

Without hard wired entry points (which an ideal distributed system does not have) it can be complicated for parts of the system to join in when they are started. For instance, if a site is created with Grid monitoring software and there is a wish for it join in the Grid monitoring infrastructure it needs to know how to register its presence so that it can be found. There are ways to solve this, the simplest to implement are centralised controllers, like the root DNS servers of the Internet [4]. They of course represent a single point of failure. There are totally distributed discovery methods such as multi-casting and peer-to-peer look up services.

While the aim of the project is to produce, a distributed monitoring infrastructure, the focus is not necessarily to be completely distributed. A compromise should be reached between the distributed nature and the rest of the features required. The aim is not to invent a new distributed model, but to implement a system using a combination of techniques where applicable to strike a balance between functionality and a distributed nature. This is likely to mean no centralised data but some interaction (or hard wiring) for new sites to initially join in the network. This way does not represent a single point of failure and does not require overly complicated services to do initial lookups (a very small part of the systems functionality). Methods to implement this will be looked at in detail.

1.4 Project Management

Six months are scheduled for the writing of the project. The tasks are broken down in Table 1.1.

These tasks define milestones, which will prevent the project from getting behind schedule. It is the intention to invest large amounts of time and effort to keep the project flowing. The previous experience of the author allows for assumptions to be made about the length of each task, how ever un foreseen problems may create extra work during the development

Month	Task
November	Read lots more about R-GMA and the Grid, set up SNMP and Tomcat, write code snippets that can talk to both.
December	Design and implement the abstract layer - probably implementing another very simple monitoring protocol to demonstrate that the monitoring can be extended.
January	Continue work on artefact - make sure all major implementation hurdles have been sorted and prepare for a demo.
February	Finish demo and present it. Start collating work for write up.
March	Have a virtually finished artefact and be well into write up since this is the hard part.
April	Review my implementation as part of the write up...
May	Finish write up.

Table 1.1: Project tasks broken down by month

stages of the project, this is why a large amount of time has been allocated for implementing the prototype.

1.4.1 Additional Project Constraints

In addition to the time management issues there are further constraints placed on the project:

- Man power - There is only one developer working on the project.
- Development platform - There are a finite amount of computing resources avaiable to develop and test the artefact on.
- Money - There is no budget, the entire project must be developed for no monetary cost. The authors use of Free Software operating systems and Open Source development tools negates this restriction somewhat.

1.5 Summary of Aims

The extended definition of what the system should do / have:

- An abstract monitoring layer capable of monitoring via multiple protocols,

- Heterogeneous execution and run on most Grid hardware,
- Site deployment comparable to traditional Grid structure,
- A distributed security model:
  - Encrypted communications,
  - Firewall traversal.
- No single point of failure.

The report must use the prototype to verify the validity of any designed architecture.

Chapter 2

Review

2.1 Introduction

The aim of the project is to produce an abstract framework, which allows distributed and secure monitoring of Grid resources. The proposed system falls into the category of middleware which means it is a software framework tying together other software [5]. This project aims to leverage existing work on Grid infrastructure and software tools in an effort to combine the best ideas and solutions to construct the best possible framework for monitoring.

2.2 Review of Core Technologies

In order design the artefact, existing technologies must be reviewed and assessed. Other implementations, which parallel some of the goals of the system, will be evaluated. The need for monitoring resources has been around longer than the concept of the Grid. Monitoring of local resources is a mature area of study. Generic and standard components, such as the use of the Internet for wide-area communication will not be discussed.

2.3 Security

The system must provide a mechanism to be protected from abuse. Misuse of communications networks and computers is widespread.

Joel M.Chrichlow, in his book, The Essence of Distributed Systems [6] explains that the

requirements of security and privacy within resource management address the procedures that are established to prevent unauthorized access to the resources. He observes that security is normally found on the outskirts of a system acting as a barrier and that the protection that it provides can be greatly reduced or negated by design and implementation faults with the basic system that it is trying to protect.

2.3.1 Encrypted Communications

Data passing through a medium beyond the physical control of an organisation can be passively intercepted (read without changing the data). It can also have the payload changed, which is an example of an active attack. This can be prevented by the use of cryptographic tools. Encrypting communications using standard industry tools, such as DES [7] or PKI [1], does not prevent the interception of data but stops it from being altered on route and stops a third party from understanding the payload.

2.3.2 Authentication and Access Control

Authentication provides a mechanism to establish a trust relationship between two parties. Once the authenticity of a party is established access control rights can restrict their use of the system, this is known as authorisation. The combination of authentication with encryption provides a secure way to transport data. Apart from providing security, authentication provides a way to track the use of the system; one of the uses of this could be if use of the system was charged for on a usage basis [8], the authentication would allow work to be associated with a user for billing. The security infrastructure used by this project is dependant on design and implementation decisions, which will be made later.

2.3.3 Firewall Traversal

A firewall [8], is a filter usually placed at the periphery of an organisations network to control incoming and outgoing traffic. There are two categories of firewall, ones where filters exclude data packets based on rules, and the other that provide application proxies which handle external communications on behalf of the internal network.

Organisations often provide a misguided approach to the use of firewalls using them as a magic bullet for Internet security [9]. Internal security (the security of the implementations of internal systems) is as important as it is possible to bypass a firewall under curtain

circumstances. The widespread use of packet filtering firewalls and application proxies has created a serious problem for distributed systems, which are trying to communicate with each other through these firewall boundaries.

A common method of firewall traversal is encapsulation of data using a standard protocol. An often-used one is HTTP and port 80. HTTP is chosen as most organisations allow access to the Web, the data must be encapsulated as it may be routed through application proxies (in this case web proxies).

2.3.4 Heterogeneous Software

Producing a heterogeneous system for the Gird requires that it run on all UNIX clone operating systems as most of the Grid infrastructure is being developed on these. Apart from different operating systems, there will be different hardware to consider. The deployment platform coupled with the required distributed communications dictate the requirements for heterogeneity:

- Programs must be either interpreted from machine-independent code or recompiled for each architecture,
- Communications must use a non-architecture specific data format [8].

There are several technologies and programming languages, which can be used to create a portable system. Three of the markets leading competitors are reviewed with the needs of producing a distributed Grid application in mind.

Java

Java can be compiled to machine-independent byte code, which is interpreted by a virtual machine at runtime. It provides a garbage collector which reclaims used memory when it is safe to do so [10], this removes a major source of errors in modern programs. The Java Virtual Machine, which interprets the byte code, provides a layered security model to protect the underlying system from either malicious or poorly written code. Runtime type checking means that dynamic loading of objects can be used (relatively) safely [10].

Sun Microsystems provide a subset of Java for producing distributed applications using HTTP for communication, the reference implementation of this technology is The Tomcat Servlet Engine [11].

	Interpreted Byte Code	Cross platform	HTTP server framework
Java	Yes	Yes	Yes
C++	No	Maybe	Add on
.Net (C#)	Yes	Some	Yes

Table 2.1: Programming language requirements

C++

C++ compiles to machine specific binaries. GNU [12] tools such as automake and autconf [13] can make the recompilation of code automatic. Not all C++ can be ported using the GNU tools, there are programming pitfalls which make the code un-portable [13].

There are distributed programming frameworks available for C++ such as CORBA [14]. C++ allows flexibility and speed; these benefits would have to be compromised with writing portable code.

.NET

.NET [15] allows any of the procedural languages provided by Microsoft (C# [16] is the main .NET language and is similar to Java and C++) to be compiled to machine-independent byte code. Microsoft calls this the IL Common Language Runtime [17]]. This would make it as portable as Java, however there is only an interpreter for Microsoft Windows and BSD.

The Microsoft .Net framework includes support for Web Services - a system for creating distributed applications using HTTP for communication.

Summary

Table 2.1 shows that Java is the most suitable language given the development requirements of the project.

2.4 Wide Area Monitoring Systems

This section includes a review of the existing monitoring system, which have similar functionality to the proposed system. The differences will be highlighted and the reasons why no existing system solves the Grid monitoring problem will be explained. Some existing

monitors which were not designed with the Grid in mind are included when they have the desired functionality.

2.4.1 Features desirable in an Ideal System

- Software implementation features:
  - **Open Source License**: Allows the extending of the software without restriction as explained in Section 3 (Derived works) of the Open Source Software Definition [18].
  - **Portable (homogenous system)**: Allow the system to run on the mixture of operating systems which the Grid is made up of.
  - **Robustness**: A stable design and implementation.
- Communications features:
  - **IP based communications**: Remote communications will take place over the Internet.
  - **Multi user system**: Should allow for concurrent use of the system by multiple users.
  - **Distributed architecture**: Monitoring architecture must follow the distributed architecture of the Grid.
  - **Web based monitoring interface**: Web browsers offer a standardised, user centric access to information.
- Security features:
  - **Encrypted Communications**: All data traversing the Internet is susceptible to attack or interception unless it is protected.
  - **Access control restrictions**: Access to the system must be controlled.
- Monitoring features:
  - **Multiple monitor protocols**: More functional system will make use of all available monitor protocols on a resource rather than relying on one.
  - **Discovery of resource information**: Administration time is reduced if system can automatically discover what values can be monitored on a resource.
  - **Real time monitoring**: This project is interested in the timely gathering of monitor data.

2.4.2 Summaries of Other Monitoring Systems

Globus Heart-Beat Monitor

The Globus heart-beat monitor is designed to monitor the state. It was produced by the Globus Project which is creating software to provide infrastructure for monitoring computation grids [19]. One of the interesting parts of the Globus project is an implementation of the GSI (Grid Security Infrastructure) [20] which provides common access control and security for Grid infrastructure. Back-end security for a grid application such as this project may be provided by a frame work like the GSI (in a deployed system).

The Globus HBM Specification provides an overview of the functionality of the system [21]. The HBM is made up of a series of components:

- HBM Client Library (HBMCL),
- HBM Local Monitor (HBMLM), and
- HBM Data Collector (HBMDC).

The monitor is run on every host, with a central repository per site (the data collector) for recording monitored data. The monitor was designed to work independently of the rest of the Globus toolkit (a complex system). The architecture is outlined in Figure 2.1 (See the list above for acronyms).

Netlogger

Netlogger is a monitoring system written for gathering real-time data about distributed systems for analysis [22]. The NetLogger system provides a framework for precision reporting of events (logging). It can be used to report monitor events or behaviour of a distributed application. There have been tools written for the system to monitor host and network performance. It is distributed with a set of programs to help developers work with the system [23]:

- NetLogger message format (ULM) : A simple, common message format for all monitoring events which includes high-precision timestamps.
- NetLogger client API library : C, C++, Java, Perl, Python, and TCL calls that you add to your existing source code to generate monitoring events.

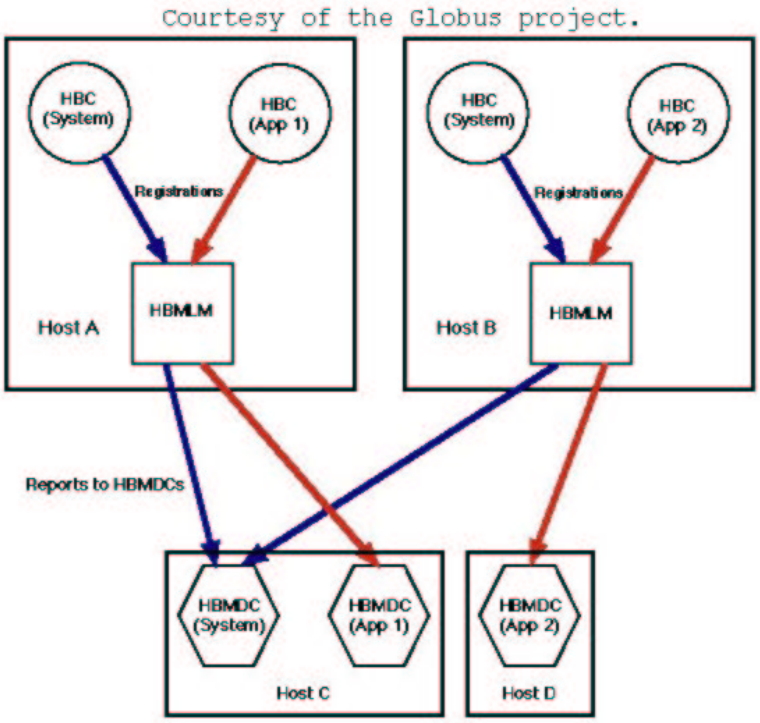

Figure 2.1: The Globus Heart-Beat Monitoring framework

- NetLogger visualization tool (nlv) : a powerful, customizable X-Windows tool or viewing and analysis of event logs based on time correlated and/or object correlated events.
- NetLogger host/network monitoring tools : a collection of instrumented system monitoring tools.
- NetLogger storage and retrieval tools :
  - netlogd: a daemon that collects NetLogger events from several places at a single, central host.
  - netarchd :An event archive system for NetLogger data, based on mySQL

This comprehensive set of utilities makes NetLogger extendable for a programmer. The netlogd program is based on some Python libraries which pass information around the network in plain text, there is work to add security to the system in the form of Globus GSI.

Ganglia

Ganglia is a tool which grew out a project run at Berkeley called the UC Berkeley Millennium Project, which aims to develop and deploy clusters of cluster across the university infrastructure [24] (this could be called a Grid).

Ganglia now uses (it has undergone many revisions) a peer-to-peer system for transferring information using XML and XDR. The system uses multicasting to discover other nodes, which will not work over most of the existing infrastructure [25], the system is shown in Figure 2.2.

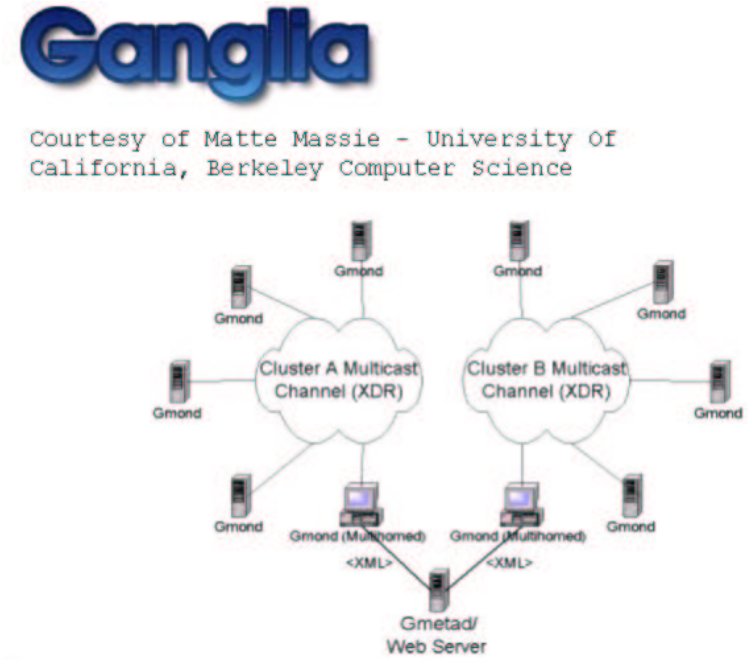

Figure 2.2: The Ganglia cluster monitoring system

Ganglia user an authorisation daemon with RSA certificates to restrict access to the system, how ever the XML data is passed as plain text around the network [25].

Relational Grid Monitoring Architecture (R-GMA)

R-GMA [26] (part of the DataGrid project [27]) aims to provide infrastructure for transporting information about the state of the Grid. It does not perform any monitoring of resources itself. R-GMA provides communication between Grid sites.

The R-GMA concept is sound - produce middleware for other Grid systems to use for inter-site communication. Its use in this project was evaluated and the following problems were found:

- Broken installation method: R-GMA relies on the RPM package format [28] using non standard paths for files which currently only installs on Linux distributions with non standard file system layouts. It is possible to install the software on any Linux machine but the install process must be performed manually. Hopefully the developers will move to a heterogeneous package format using a portable installation process such as the Ant installer [29].
- Monolithic design: The package is overcomplicated by API support frameworks for multiple languages. For instance it is hard to pick out the Java only parts of the system.
- Overcomplicated interface: Writing R-GMA code is simple - the API is well defined but if the system does not work (broken installation) it is very hard to track down the problems.

It is the opinion of the author that R-GMA is useable for a developer of the R-GMA code itself but it is not ready for systems to be developed on top of the framework. Time restraints on this project prevented a more thorough investigation, R-GMA may have worked with a lot of developer effort but it was deemed too great a risk to the goal of producing working software.

DSG Grid Monitoring Prototype

The Distributed Systems Group [30] at Portsmouth University have produced a prototype grid monitor. It uses part of the Globus toolkit to access Grid Meta Data Servers (MDS), which contain resource data about the Grid. The prototype provides a web interface to the MDS machines and can be used to display resource information graphically, the architecture is outlined in Figure 2.3.

The prototype demonstrates an interface to Grid monitoring information. If this prototype were installed at each Grid site it would provide a distributed framework for access the data.

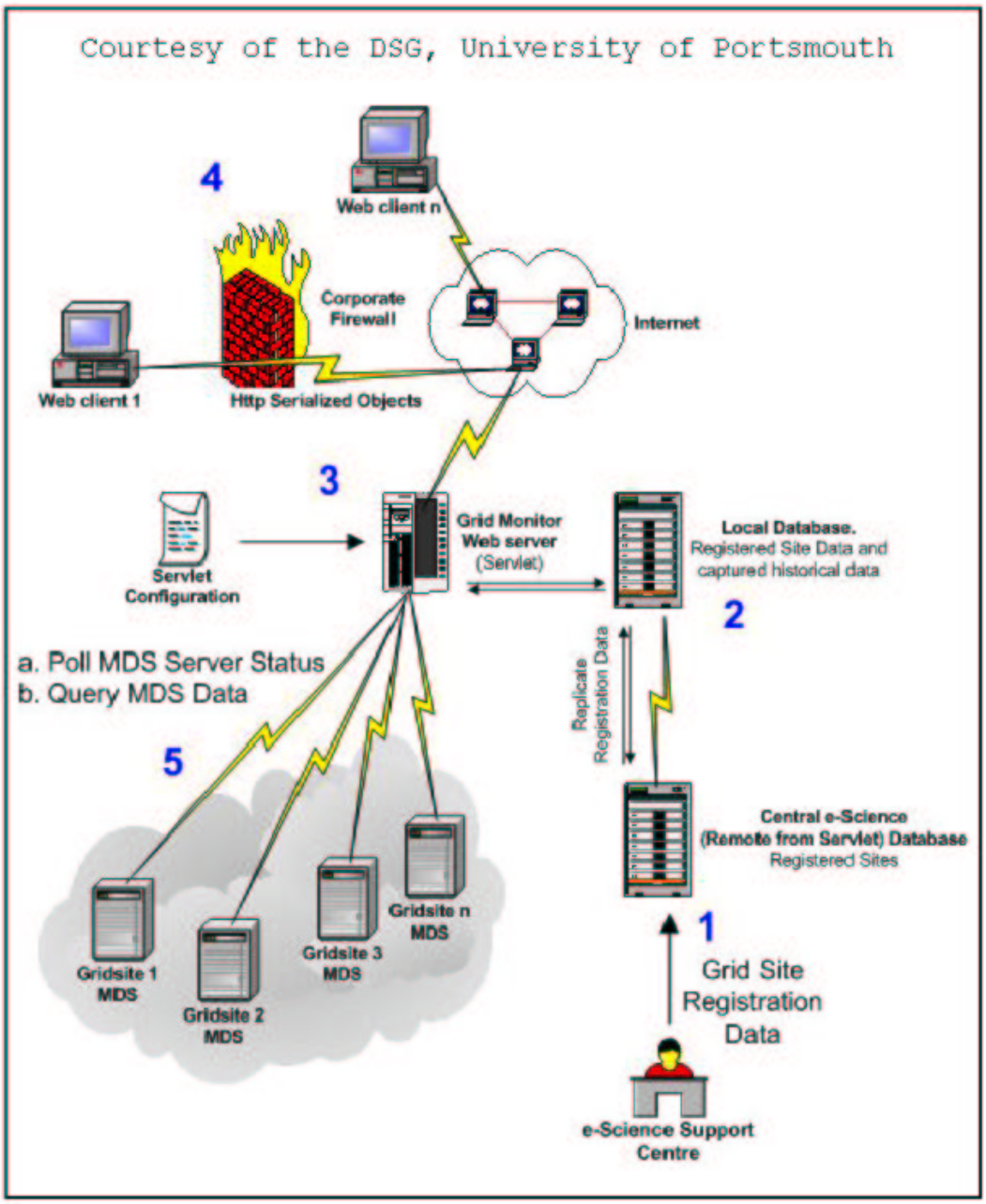

Figure 2.3: The architecture of the DSG Grid Monitor Prototype

2.4.3 Comparison of Features of Existing Systems

Figure 2.4 shows a comparison of the features of the reviewed systems. It highlights the differences and the strengths of each package.

	Globus HBM	Netlogger	Ganglia	R-GMA	DSG Grid Monitor Prototype
Software implementation					
Open Source License	Yes	Yes	Yes	Yes	Yes
Portable (homogenous system)	Semi	Semi	Semi	Yes	Yes
Communications					
IP based	Yes	Yes	Yes	Yes	Yes
Multi user system	Yes	Yes	Yes	Yes	Yes
Distributed architecture	Yes	Yes	Yes	Yes	Yes
Web based monitoring interface	Yes	No	Yes	No	Yes
Security					
Encrypted Communications	No	No	No	Yes	No
Access control restrictions	No	No	Yes	Yes	No
Monitoring					
Multiple monitor protocols	No	Yes	No	NA	No
Discovery of resource information	No	No	Yes	NA	No

Figure 2.4: Comparison of features of reviewed monitoring systems

2.5 Summary

None of the reviewed frameworks provide all of the features required for a secure distributed Grid monitoring system. During the review process the developer read discussions on each specific implementation, which helps give a better understanding of the issues with developing a monitoring framework.

Chapter 3

Design and Implementation

3.1 Introduction

In an effort to simplify the design and implementation process, a rolling prototype methodology will be adopted. Before this is discussed, the requirements of the system need to be analysed.

3.1.1 Initial Requirements

- Be able to work through firewalls,
- Only need one node per site accessible to the Internet (a site gateway),
- Security,
  - Provide authentication to restrict access,
  - Use encryption for communication over the Internet.

3.1.2 Modified Implementation Requirements

There are further restrictions that the implementation of the system will add to the initial requirements, as the system is reasonably complicated to implement, the developer was unfamiliar with the overall area and how best to design the system to work correctly and efficiently. The developer was however familiar with some of the ideas needed to implement the

final artefact, but many aspects of the system were bound to fall down at the implementation stage, but then they could be modified based on experience.

An example of a Hidden System Requirement

A typical client for the system is a Java applet. This is an obvious way to access the system in a heterogeneous way. Java security uses the sand box approach, where the applet can only communicate with the server where it was downloaded. This means that if a user wants to monitor a node, which is part of a different site, once they have downloaded the applet all communication must be from the applet to the serving site, even though the client knows that the data it wants is not at that site.

A workaround for this problem is to have the site gateways forward all requests, which are not for them, on to the remote site transparently. This feature must be introduced as a requirement of the system.

In order to prevent the requirements of the system from changing late in the implementation stages of the project, the developer can attempt to find any potential problems and solve them as early as possible during the development stage, this dictates the way artefact is designed and implemented.

3.1.3 Design and Implementation Strategy

In order to reduce the effect of changing requirements introduced because of problems with the implementation an iterative approach can be used. Self-contained modules can be created to solve any problems envisiged by the developer when implementing aspects of the artefact. These modules can be developed in isolation. The stand-alone components can be integrated into the overall system one at a time after separate testing. This methodology is discussed in more detail later, and shown in Figure 3.1.

3.1.4 Design Components

It is important to have a good architectural view of the system at each stage of the design and implementation process. It is expected that fine-grained design decisions will be made as problems become apparent during development of the system. A list of coarse-grain areas that need to be considered is below.

Communication

As described in the Chapter 2, the most common way to traverse a firewall is to tunnel all traffic over HTTP. Java provides a tested platform to do this by way of Java servlets. A simple client/server paradigm will be sufficient. In this case, a site gateway servlet may be acting as a server if information is being requested of it and as a client when it passes any information on to other sites (routing). The end user will also be using a client to access a site gateway. It should be possible to use common code to do this, so that any client implemented can use the same components to access a servlet.

This HTTP communication will have to be encrypted to meet the system requirements. HTTP supports SSL [2] - so using it gives us the benefit of standardised built-in encryption support.

Monitoring

The task of this artefact is to monitor the resources at a site. The original specification dictates that this must be extendable. What this means is that it must be as easy to add new protocols to the system or add further functionality. A desirable feature is the use of multiple monitor agents to combine information - in other words if a machine is running two different types of monitoring agents it would be good if the system could fuse the data and provide an abstract monitor of the overall system.

An interesting issue is how to make it easy to add new monitoring agents to the system and how to provide a common view of a resource when different kinds of monitors, possibly with different data formats, are being used.

Authentication

As outlined in Chapter 2, there is a need to encrypt communications, which travel across the Internet in order to protect the system from abuse. There is also a need to restrict access to the system - a distributed way to provide authentication needs to be implemented. A user may want to access data on a remote site - the administrator of the remote site will not have the login credentials for this user so the system must be able to do a remote authentication whereby it contacts the users home site. This means that the system must be able to derive the users home site from their logon credentials. It is expected that a production version of the system would use common Grid authentication systems such as GSI for the back-end authentication process.

Clients

Although the goal of the project is the development of a framework, it is implausible to develop and test the system without some kind of user interface. Part of the project will involve an example interface to demonstrate the systems usage. It can be developed in parallel with the main parts of the project.

3.1.5 Design Philosophy

There are three key and overriding design goals for this system:

- Distributed - wherever possible this system should be built using distributed system design goals. The aim here is produce an artefact that is scalable with no points of failure. There are some areas, which cannot be addressed within the time constraints of this project. A key one being resource discovery without hardwiring, this is outside the scope of the project. Even though this aspect is left out, it must still be considered, for instance by leaving code stubs where this could added to the system later.
- Stateless - although this is a distributed system design issue it should be highlighted. A stateless system will always fall back to its starting state even after a catastrophic failure. When using the Internet as the communication medium, matters such as network latency and quality of service are often beyond the systems control. There is trade off that occurs for the increase in work that must be undertaken for every action. In real terms, this will affect the speed of the system, but will promote robustness. It is important that the system can produce timely responses to queries, hopefully mechanisms such as caching should make this possible.
- Abstract middle-ware design - the goal is to produce an abstract system, which demonstrates the ideas behind the project. It is hard for the developer to just create the framework, which is why components are being produced to work with the abstract architecture (a front end and monitor clients). The point is however, to produce a working prototype that demonstrates that the chosen architecture is viable.

3.1.6 Design Plan

After discussing the breakdown of the requirements, it is possible to suggest a plan to follow for developing and implementing the system. This plan needs to be followed to produce both

a prototype of the system and a final design specification for an abstract grid monitoring achitecture. The overall goal of the project is produce this specification; the process of producing the prototype will support this final model.

Design and Implementation Stages

The specifics of each of these stages will be dependent on the output of the previous one.

1. The production of an abstract system design template that describes the architecture of the system. This is a high-level design, it should produce a specification, which will highlight areas, which may cause problems during implementation and may change the overall design.

2. The implementation of small modules to test ideas and produce working components to build the system with.

3. The construction of a first prototype. This does not need to fulfil all of the requirements, but should demonstrate the overall design goals and provide evidence that the design is sound and possible to implement.

4. The evaluation of first prototype, by referring back to the specification outlined in stage 1. The concept is to focus on interesting problems within the implementation, by this stage the system should be more fully understood. It is likely that a much better (or more functional) implementation can be designed at this point. The overall specification may change.

5. Another round of small separate modules will be developed to explore any further issues that need addressing within the implementation of the first prototype.

6. The merger of the new modules with the original prototype. It is possible that this will be a fundamental enough change to the system structure to be essentially a redesign rather than an upgrade - the object-oriented nature of Java should ensure that code can be ported without having to duplicate work. This stage will produce a second and more viable prototype.

Figure 3.1 gives an overview of the design and implementation process.

After completion of the second prototype it should be possible to produce a detailed specification for the final Abstract Grid Monitor. The issues investigated during the implementation of the first prototype and the stand-alone components will support this.

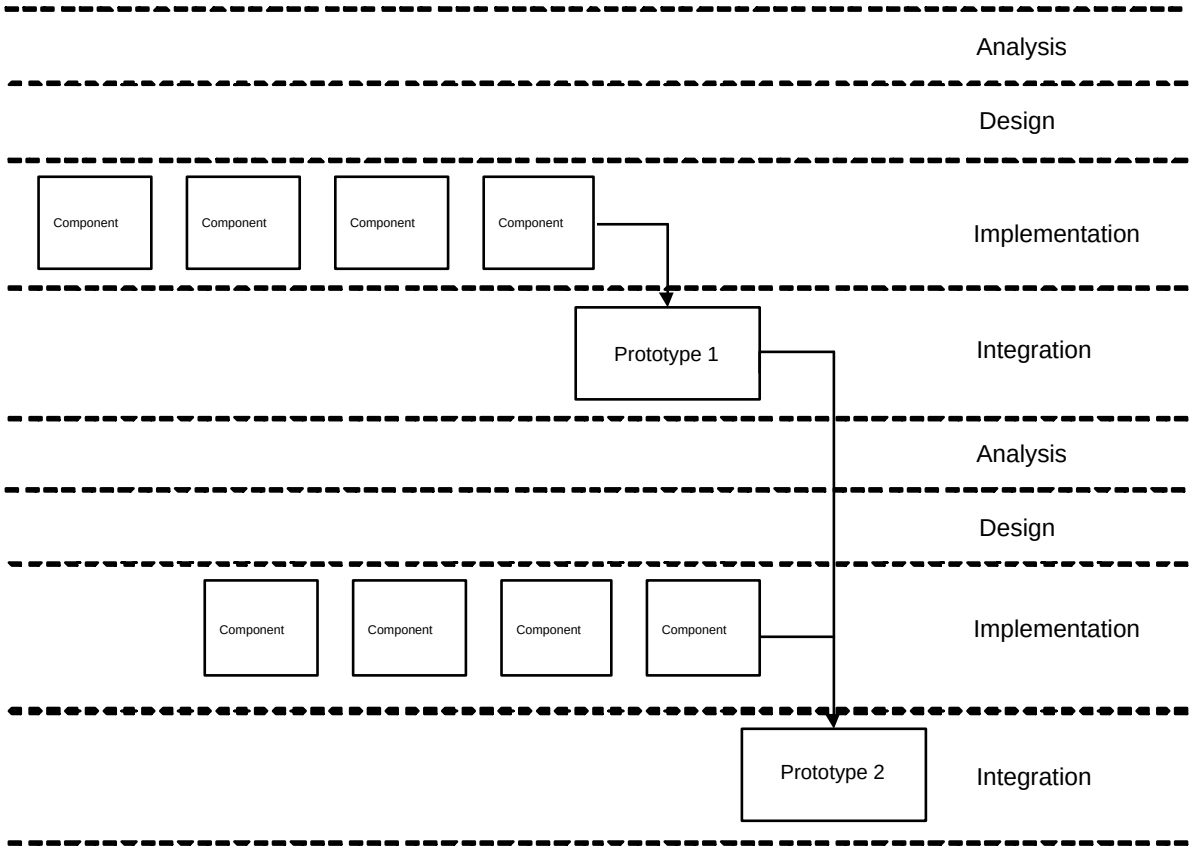

Figure 3.1: The design & implementation process

3.2 Abstract System Design

This is the design of the architecture that the artefact will use.

3.2.1 Server Interface

The artefact uses a client/server paradigm. Where a user (client) queries a site gateway (server) for information. The gateway transparently processes this request and returns the information to the client. This interaction in shown in Figure 3.2.

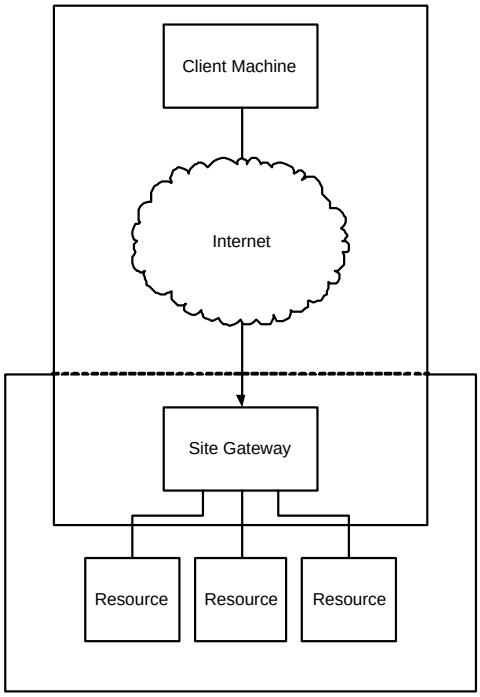

Figure 3.2: Client gateway interaction

Server Interface Requirements

It is possible at this stage to define what information the site gateways will have to be able to serve.

If a query for information cannot be satisfied by the gateway because it involves remote data, the gateway must pass it on to the site that can handle the request and transparently return the results to the client, this is shown in Figure 3.3.

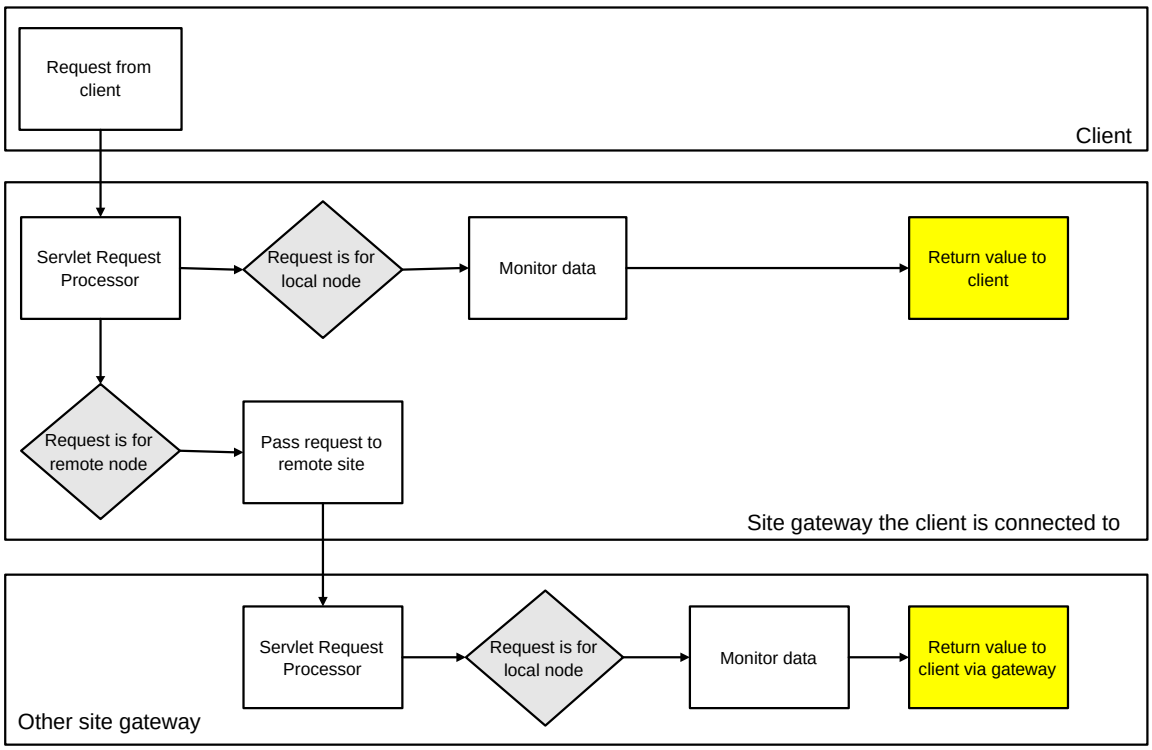

Figure 3.3: Gateway request processing

A list of replies and queries, which gateways may process:

- A list of sites - every site needs a unique name to identify it and a URL to the site gateway so it can be accessed. The issue of naming and discovery is interesting but is not the focus of the project. It is sufficient to 'hardwire' this data into each site by providing a list of site URLs and names for each gateway. This means there is a need for a general data store for each site gateway to hold this information - and probably other things.

- A list of nodes given a site name - a gateway needs to maintain a list of the nodes at it's site which can be monitored. There needs to be some kind of discovery mechanism for creating this list. It may be possible to produce auto discovery code which can find nodes at the site and record that they can be monitored. If this is not possible this information could be hardwired into a data store.

- A list of products given a node name and a site name - this means returning a list of all of the things that can be monitored on a node. The code that handles monitoring of a node, must be able to translate the monitor specific names for objects that can

be monitored, into a standardised list. There is a need to have a standard naming convention for resources (such as CPU usage) because each node may be monitored using a different protocol, which may have a different internal naming convention.

- The value of a monitored object given its name, a node name and a site name - this is the actual monitoring of something on a node such as CPU usage.

Server Interface Definition

From this we can create a list of services which the server part of the site gateways must provide clients. This is a definition of the interface that compatible clients will use and that all compatible servers must implement.

```
GetSiteList();
GetNodeList(SiteName);
GetProductList(SiteName,NodeName);
GetProductValue(SiteName,NodeName,ProductName);
```

This is the layer of abstraction that a client will interact with. Since a gateway passing on a request to a remote site is acting as a client this doubles as the inter-site communication definition. As long as this definition does not change any client and server which implement it will be able to communicate regardless of the internal workings of the program.

This abstract layer will make implementing the system much easier for the developer. During the implementation stages it is likely that there will multiple development sites running different versions of the code. If the communication interface remains the same the fact that they are using different code will not stop sites from participating in the system.

Removing the need to resynchronise every site after each code change, should speed up the development by reducing the amount of time distributing and checking code versions.

3.2.2 Remote Communication

This system is using a simple client server implementation for communication. In order to make use of the standard firewall traversal solution all client server communication will be via HTTP. HTTP is a stateless protocol and this supports the stateless design ethos which is being followed.

The Java servlet engine Tomcat provides a platform to deploy servers which can be interfaced with via HTTP. The mechanics of the HTTP communication is provided by Tomcat. The implementation of a server and a client which can use HTTP for communication will be one of the programs written in the second stage of the design and implementation. It is important to have this code working early on during the project development as it is required for any remote communication.

3.2.3 Server Replies

There must also be a specification for what data the server will reply to the requests with and what format it sends replies in.

Server Reply Definition

Each of the methods outlined in the server interface definition will return data to a client which calls it. This needs to be standardised before clients are written as it will break compatibility between servers and clients if it is changed later.

```
GetSiteList();
  Returns a list of site names.
GetNodeList(SiteName);
  Returns a list of IP addresses of nodes within a site.
GetProductList(SiteName,NodeName);
  Returns a list of products aviable on a node.
GetProductValue(SiteName,NodeName,ProductName);
  Returns the value of a product from a node.
```

In each case the data returned by a method is of the same type, for example every item returned by GetSiteList() is a site name - it does not mix any other types of data. This means it is possible to have very simple replies to these methods. They can be a simple text list of items separated by a new line. If a method returned mixed data it would be required to also return an identifier for each item (this would likely be done using XML).

Relating to HTTP

A client will request a specially formatted URL, which the server can map to each of the four methods it must provide. The server will perform the required work and then return

the data. Since the client knows what it just requested it will know what data type it is receiving. This is very similar to a normal web server which has a page requested using a URL, which it then maps to a file, opens it and sends it back to the web browser.

For a client to invoke one of these methods on the server it must construct the correct URL, which will be mapped by the server to the method. It then sends this to the server using HTTP. The server replies with zero or more text items all of the same data type.

3.2.4 Server Data Store

It is indented that the artefact will use data caching to increase the responsiveness of the system - this cache will need to be stored somewhere. There are also settings such as the site name and the URLs of other site gateways, which the server will need to store locally. There is a need for some kind of local data store.

JDBC Interface to Database Back End

The Java language has an abstract interface to databases called JDBC [31], which allows programs to access many different kinds of databases using SQL. Since there may be a need to store a large amount of data in the form of the cache, a relational database is a good choice as a back end data store. JDBC means that the specific implementation of the database is not relevant as long as JDBC supports it (examples MySQL [32], Microsoft SQL [33], Oracle [34], and PostgreSQL [35]).

Solving the Changing Data Requirements

One of the problems of using this rolling prototype design methodology is that the requirements of this data store are very likely to change over time. With traditional database design the requirements dictate how the database is designed.

An interesting way to solve this problem is to create a system which abstracts the interface to the database. This idea of abstraction follows the design philosophy of the project. This can be extended to create an interface which standardises access to data, which does not require a strict database structure.

This can be broken into two parts:

1. When writing code, which must access the data store never access the database directly

- do it through an abstract interface. This means that if the database changes, only the data store interface code is affected.

2. Change the physical layout of the database to allow any relational information to be stored. This means that it is not necessary to create new tables and new definitions when there is a need to store some new kind of data.

This second part is an interesting solution but also time consuming for the developer. The data store is not the main focus of the project so during early development the database will be designed in a traditional way. The abstract interface outlined will allow the back end to be switched to the new kind of data store when and if there is time to implement it.

3.2.5 Resource Monitor Interfaces

To demonstrate the monitoring framework the well-documented SNMP [36] (Simple Network Management Protocol) monitoring daemon will be used. Since the project is to produce middleware the important thing is to choose an example back-end monitor which will be easy for the developer to work with.

The aim of the system is to monitor the state of resources and allow access to that information. As discussed there are many different monitoring tools, which are mainly incompatible. These tools use different internal naming schemes to describe the data they monitor and provide different ways to access this information.

If the system is to be able to use different types of monitors there is a need to translate the monitor descriptions of the data into some common format. When there is a standard for naming the resources, the type of monitor which is providing the data becomes hidden from the program.

This abstraction would make it easier to change the specific monitor code and to add new monitors without having to rewrite any of the rest of the code.

Naming Convention

Within this report **product** will be used to describe any resource within the context of monitoring.

There are two classes of resource that may be monitored. Information about the resource, which is largely static in nature - for instance the name of the machine, to a user this is

Product Name		OID Name
infoDescription	<->	1.3.6.1.2.1.1.1.0
infoMemTotal	<->	1.3.6.1.2.1.25.2.2.0
dataNumberProcesses	<->	1.3.6.1.2.1.25.1.6.0

Table 3.1: SNMP OID Mappings

interesting, but it is unlikely that they will want to continuously poll its value, as it rarely changes. The other class contains all resources which change value regularly and which a client may be interested in polling:

1. Data - a resource which changes value regularly,
2. Info - a resource whose value is usually static.

By using these class names as prefixes to the product names we add extra functionality - a client would know what class of data it is looking at by examining its name.

While developing the code to access the monitors a list of available products and the mappings to the monitor specific resource names will have to be made. This is another requirement of the data store - it will have to hold this table of mappings.

As an example the SNMP monitor uses resource descriptors called OIDs (Object ID). The data store will have to map OID values to product names.

It makes sense to allow these mappings to work in both directions; a monitor may translate a product name into an OID in order to query the monitor daemon or a monitor may request a list of available OIDs from the daemon and translate them into product names.

Monitor Drivers

In order to access the monitors in a standard way, the code, which will talk to each monitor will need to provide standard functions for the abstract monitor interface to call. The code for each monitor could be called a driver.

When a driver is loaded it will be given the IP address of a node to try and interface with (on the standard port for the monitor - this could be extended to allow non standard

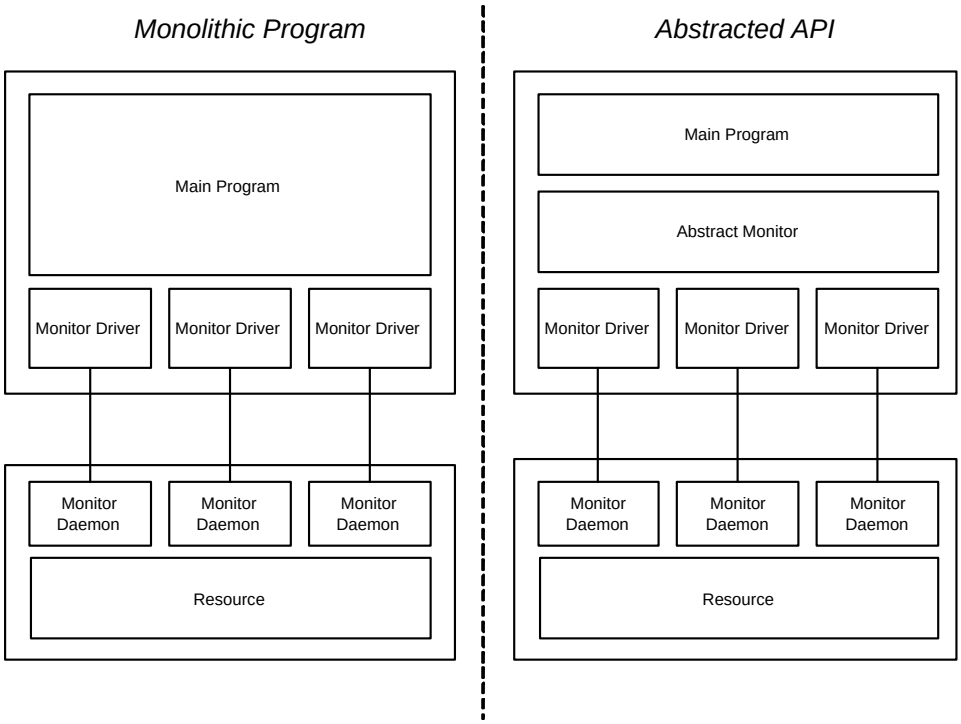

Figure 3.4: An abstract monitor

ports). Loading of the driver should fail if it cannot interface with the node. This provides a mechanism to do automatic discovery of resources, the driver can be loaded for a node and if it fails to monitor the node, the system knows that that driver cannot be used to look at that node (probably because the monitoring daemon that the driver talks to is not present).

Monitor Interface

The abstract monitor interface will be used by the system to monitor a resource. All calls to the drivers will be handled by this interface. The interface will provide a generic view of a resource regardless of what monitors are actually being used. It also abstracts interfacing with the drivers away, hiding their functionality as shown in Figure 3.4.

3.2.6 Summary

The components required to build the artefact have been introduced. Figure 3.5 shows a view of the proposed architecture.

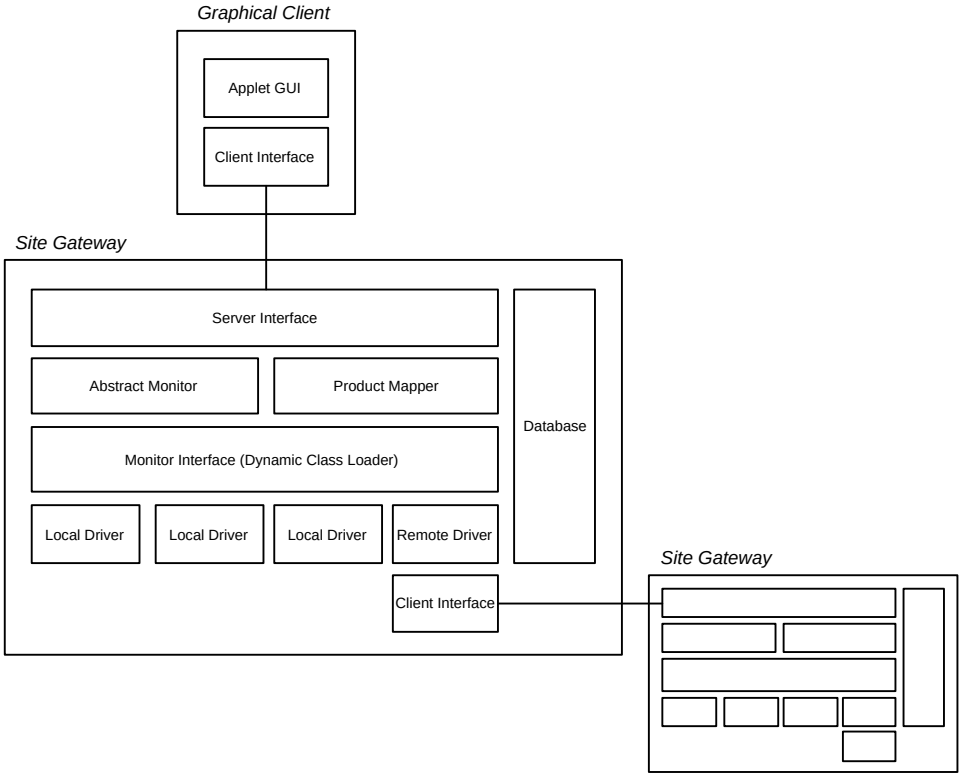

Figure 3.5: Initial architecture

The next stage in building the prototype is to implement small programs to demonstrate areas of functionality, which the developer is unfamiliar with or there is no clear solution:

- A servlet - to demonstrate remote Java over http,
- An SNMP monitor - may be problems getting Java to work with SNMP,
- Another monitor so that there is a pool to demonstrate the multiple monitor framework,
- An applet client to show communication over HTTP solving any Java security problems.

3.3 Implementing Test Programs

This stage of design and implementation requires the creation of some programs to demonstrate that key functionality of the artefact is possible to implement. On completion these programs will contain tested code, which can be used as a foundation to build a prototype. When integrated into the prototype artefact they will form a library, client interface, server interface (servlet) and some monitor drivers.

3.3.1 The SNMP Monitor

Although this is stand alone driver to query an SNMP daemon it is important to bear in mind where the code is going to be used; as one of many drivers called by the monitor interface.

The SNMP Protocol

There are currently two SNMP protocols, version 1 and version 2 [36].

SNMPv2 is a revised protocol, which includes improvements to SNMP in the areas of performance, security, confidentiality, and manager-to-manager communications [37].

SNMPv2 is more complicated protocol than SNMPv1. The SNMP daemon provided by Microsoft with Windows XP and the daemon provided with the Linux distribution Debian both support SNMPv1 but implement different variations of SNMPv2. This highlights one of the problems with SNMPv2, implementations seem to extend the specification laid out in the RFCs. Both daemons implement SNMPv1 in the same way.

Bearing this in mind a more compatible driver would use the SNMPv1 protocol. SNMPv1 supports both the reading and writing of objects. This driver is only required to read values. There are three read functions the SNMPv1 protocol provides which the driver may need to interact with [36]:

- GetRequest - supplies a list of objects and possibly values.
- GetNextRequest - is used to walk through a list of values.
- Trap - this allows the daemon to push events to a client. The daemon can be set to contact the client if certain criteria are met.

There is no need to implement the Trap function in this driver, it is interesting to allow a client to register a Trap for an event with the SNMP daemon but the primary function of this driver is to allow the value of resources to be polled from an SNMP daemon implementing version SNMPv1.

SNMP uses the idea of a MIB (Management Information Base). This is a list of all the resources, which can be monitored. Each resource is identified by an OID (Object Identifier). There are different schemas for the MIBs, what that means is that while still supporting the SNMP protocol different MIBs will contain different OIDs.

The only security implemented in SNMPv1 is the use of a community string [36]. This is a custom string which must be known before a client can query an SNMP daemon. It is possible to have multiple community strings and have them bound to different MIBs. There are standard community names (one reason why SNMP security is poor). "Public" is used to describe a read only MIB and is set up by default. For the purposes of this driver the community name can be hard set to Public, this prevents write access to the MIB but this functionality is not required.

The SNMP protocol is well defined in the RFC, it would be possible to write an implementation of it for this driver. Other people have already done this work many times - there are Java classes available. One implementation was chosen which was written in a way understandable to the developer in case it needed modifying. It fully supported SNMPv1 and was released under the GNU license (Free Software). The package details are listed with all other Java libraries used to develop the artefact in the appendices.

SNMP Monitor Driver

The SNMP driver is a wrapper which will call the SNMP code which does the actual communication. The constructor must test the connection to the SNMP daemon and fail if the connection does not work. The simplest test of the connection is to retrieve a known OID value which all SNMPv1 MIBs have. If this does not work the code knows that SNMPv1 communication is not possible and it can throw a Java exception and fail to load.

The get method must accept an OID name and call the SNMP communication code to retrieve this value. The behaviour if something goes wrong is to return an empty string. The reasoning is this is a non-fatal error which may be transient. The stateless design requires that if an error occurs the system can simply retry the failed method.

Testing SNMP Driver

A stand-alone wrapper program using the SNMP driver can test the implementation in an isolated way.

Pseudocode for SNMP test app:

```
public class SNMPTest{
  public static void main (String hostName){
    Debug.out("Creating Client for " + hostname);
    try{
      SNMPClient testClient = new SNMPClient(hostName);
    }
    catch(AgentException){
      Debug.out("Loading client failed");
      System.exit(0);
    }
    Debug.out("Client loaded, fetching an OID");
    String oidValue = testClient.get("1.3.6.1.2.1.1.1.0");
    Debug.out("OID Value: '" + oidValue + "'");
  }
}
```

This demonstrates how abstraction simplifies the code. The interaction with SNMP is being hidden at what is still a low level function of the system.

While testing the SNMP driver long pauses were observed when trying to run the client for hosts that did not have an SNMP daemon running. The SNMP 3rd party communication code was changed to reduce the time that was waited trying to connect to the SNMP daemon.

```
//dSocket.setSoTimeout(15000);  //15 seconds
dSocket.setSoTimeout(2000);     //2 seconds
```

The initial timeout of 15 seconds produced a lengthy pause when trying to connect to a host without an SNMP daemon. This is likely to happen a lot when the system is dynamically loading drivers to probe nodes for resource monitors. The 15 second timeout is defined in the SNMP RFC. All SNMP communication within this system will take place over a LAN, the SNMP specification had to take into account other slower mediums. The value of 2 seconds

Product Name		OID Name
infoDescription	<->	/proc/version (Proc value)
infoMemTotal	<->	memtotal.sh (Small program)

Table 3.2: Example Procmon product to OID mappings

was proven using simple tests to always discover an SNMP daemon, even when querying a slow host, which was under serious load, the code still worked. There may be room to reduce this timeout further.

3.3.2 Procmon Monitor

Before the developer could write code to handle the loading of multiple drivers it was necessary to write a second driver for a different monitoring protocol. In order highlight the problems with writing drivers the developer chose to implement a custom monitor with both a server and a client, which would force issues to be solved when writing the driver abstraction layer. Other monitoring protocols are either very similar to SNMP or time consuming for the developer to implement.

Summary of Procmon Monitor

The monitor uses the standard client server paradigm. The client can either request the contents of a file on the server machine or the result of an application run on the server machine. The design leverages the virtual file system "proc" which is present on many UNIX clone operating systems. Proc holds dynamic information about the state of the machine which is kept up to date by the kernel. The Procmon monitor allows remote access to these files and this provides a very simple way to gather resource information. The Procmon design is outline din Figure 3.6.

A Procmon client sends a request to the server in the form of a path name. The monitor opens the file and returns the contents. This is very simple but does have problems. Proc represents a snap shot of the resource information at a given time, which means it is not possible to monitor something like how many bytes were sent over a network link in the last second. A simple solution is to allow a client to call a small program on the server which can do some computation on the values of proc and return the result.

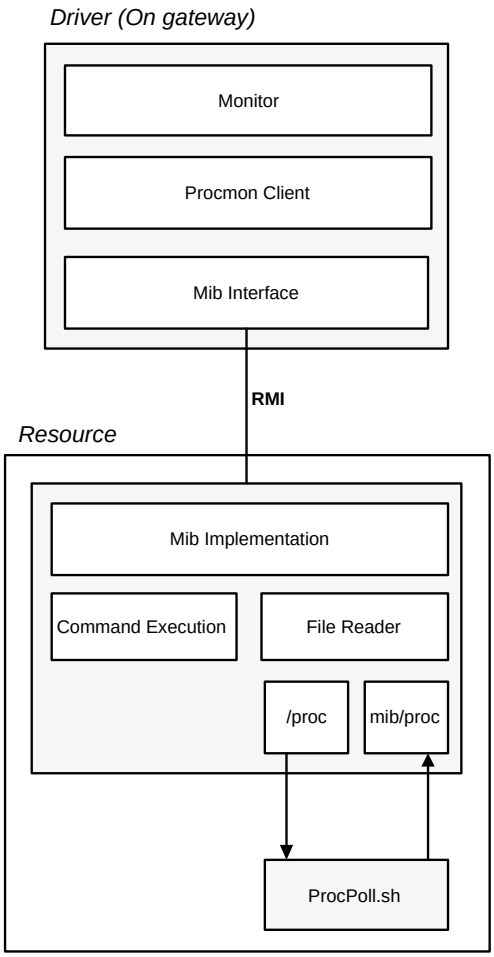

Figure 3.6: The Procmon design specification

This produces a very flexible system, which is able to produce a very wide range of resource data.

3.3.3 An Abstract Monitor

The protoype requires an abstration layer which will use all possible drivers to monitor a node. Before this can be done a standard interface is needed to access the monitor drivers.

As shown above a difference between the two drivers written is that they use different names for the OIDs. A layer is required for each driver to translate a standard product name into a monitor specific name (OID). All drivers could share this translation layer.

OID Name Translation - Product Map

A static library capable of mapping OID names specific to a monitor protocol to a product name which is a generic description used by the system. The mapping should work in both directions.

At this point in the project there is no general data store for the system. The Product Map needs a way to store the mappings. In the later stages of the project there may be a generic data store capable of holding this information. This must be kept in mind as it may be desirable to change the Product Map to use this data store at a later date.

The Product Map used a simple relational database (Appendix A) to store the OID mappings.

Dynamic Code Loading

To provide extensibility, the abstract monitor can dynamically load the monitoring drivers. There are two problems to solve:

- Knowing the names of available monitors,
- Loading the code.

To simplify the initial implementation of this dynamic code, loading the names of any drivers can be hard-coded into the abstract monitor. This could be altered to be fully dynamic at a later date.

The loading of dynamic code in Java can be handled by the mechanism of introspection. It is possible to extract information about a compiled class at run time. To instantiate a class dynamically the program must create a specification for the classes constructor, which is handled by the introspection classes. After instantiation the created object behaves in the same was as one loaded conventionally.

Summary of the Abstract Monitor

The Abstract Monitor can dynamically load monitor drivers for a given node and expose a combined translated product name list. The stages of this process are:

1. Abstract monitor is loaded for a node (identified by IP address),
2. Dynamic monitor drivers are loaded,
3. Successfully loaded drivers are queried for all possible OID values that are are in the Product Map database,
4. The OID names are translated into generic product names by the Product Map and the list is returned.

Once the Abstract Monitor has successfully loaded it internally maintains the mappings of generic product names for the dynamic driver. In this way the Abstract Monitor can pass a query for a generic product name to the correct dynamic monitor driver.

The combining of the product lists is done in a first come first serve manner. If a dynamic monitor driver tries to register an available product with the Abstract Monitor when the product has already been registered by another driver the registration will be ignored. This creates a simple implementation but can be improved; there could be extra functionality that makes an informed decision about which monitor to use when a product was found more than once. Things that could be taken into account:

- The speed of a typical response from the class of monitor,
- The effect of monitoring a resource on a system using the monitor,
- The number of products already registered with the monitor.

3.3.4 Servlet

The Servlet is the server interface of the system.

Servlets handle method invocation by processing HTTP requests. A client invokes methods on the servlet by sending specially formatted HTTP requests. The Java servlet framework provides classes which process them; this code is hidden in the framework.

There are two ways to encode data using HTTP, Post and Get [38]. Post encodes data in the HTTP request; this data is not cached (part of the HTTP/1.1 specification). A Get request encodes the data in the URL part of HTTP; this can be cached and may appear in web proxy logs.

For the developer working with HTTP, Get requests are easier as they can be created by hand in the URL bar of a web browser. For instance, this URL would pass the variables testvar and othervar to the script index.php on the www.dsg.port.ac.uk web server:

```
http://www.dsg.port.ac.uk/index.php?testvar=yes&othervar=3
```

At this stage in the project there is no client interface to the server, it is important when writing and debugging the servlet interface that there be a way to create correct URL requests for the server, until a working client is complete the developer can craft the URLs by hand.

3.3.5 Applet Client

A Java applet provides a way to provide dynamic content to a user. The advantage they have to a developer over static web based clients is that the applet runs locally on a users machine. This allows extended functionality beyond that of a server side program.

A Java applet client is required to explore the potential of a client side interface to the system. An applet is subject to strict security constraints to protect the client machine, this raises design issues.

Exploring a Dynamic Interface

There are two functions that a client for this system needs to provide:

- A way to navigate the sites, node and choose a product to look at.
- A way to display a product value (in an interesting and intuitive way).

A Dynamic Graphical Tree

The site / node / product navigation can be represented in a tree. The focus of the project is not to look into human computer interaction; a standard solution will be followed displaying the navigation. Modern operating systems use graphical trees to navigate hierarchical data; users should be familiar with the interface hopefully making the interface intuitive. The graphical tree must be dynamic - nodes need to be added at run time as the tree is browsed and the server returns data. The completed tree class is shown in Figure 3.7.

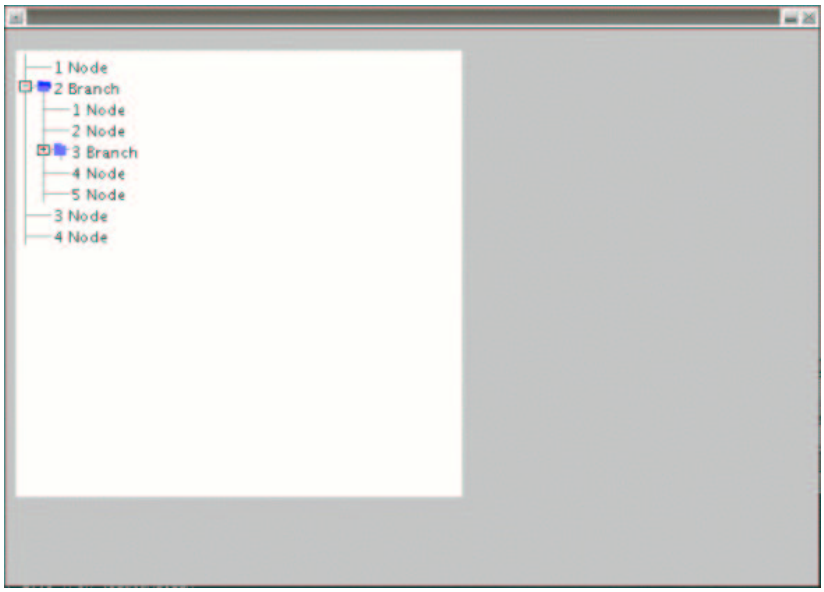

Figure 3.7: The graphical tree running in a Java AWT application

Animated Graphing

One way to plot numerical data is on an animated graph. A custom lightweight animation class can display polled resource information as a line graph. The class implementation is kept as simple as possible to reduce load on the client machine. The implementation of the dynamic graphing is shown in figure 3.8. The implementation was kept simple as it is meant as an example, a fully developed class would label the axis etc.

Java Applet Security Issues

A standard applet downloaded from a web page is un trusted - the user (and client virtual machine) has no idea what the code is programmed to do. The Java VM provides security

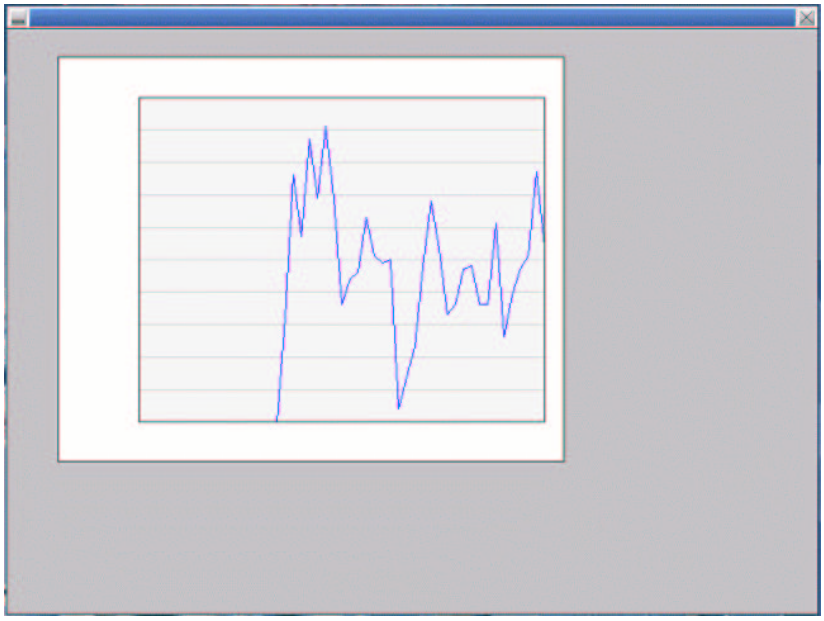

Figure 3.8: Animated graph class running in an AWT window

preventing malicious acts by the applet to allow un trusted applet to be run in a trusted environment [39].

This must be taken into consideration when developing an applet. The only code which this client uses that requires interaction with anything outside of the program, is the network communication with the server.

The Sun Java Security - Frequently Asked Questions [39] states in Section 8:

```
"Applets are not allowed to open network connections
to any computer, except for the host that provided
the .class files. This is either the host where the
html page came from, or the host specified in the
codebase parameter in the applet tag, with codebase
taking precedence."
```

For example, if you try to do this from an applet that did not originate from the machine foo.com, it will fail with a security exception:

```
Socket s = new Socket("foo.com", 25, true);
```

This means that the client is restricted to network communication only with the host it was downloaded from. Care must be taken to open the communication requests to exactly the same URI as the serving host - using an IP address or a host alias will make the applet fail to run.

3.4 Design and Implementation of Prototype 1

The first milestone of the project is to assemble a working prototype of the system. It will not be fully compliant with the specification but will demonstrate that the model for the system is sound.

3.4.1 Reduced Functionality Specification

Prototype 1 must:

- Provide an applet client to access the system.
- Run on at least three different sites to simulate a grid.
- Allow the transparent handling of inter-site communication within the servers.
- Dynamically monitor a node.

The basic design philosophies discussed in the design introduction will be followed such as robustness through statelessness. The main omission to the original specification is the lack of security, which would complicate the debugging of the HTTP communication.

3.4.2 Implementation

The basic functionality for this prototype has already been developed as separate components. The code was extracted and sorted into Java packages to provide structure for the API.

HTTP Client

A component missing from the first stages of development was a HTTP client to call the methods of the servlet. As discussed a formatted Get request is used to invoke the methods on the server. HTTP communication is a common requirement of Java programs but at the time of writing (JDK1.4) there are no Sun Java classes to simplify the creation of a HTTP client. It would have been possible to implement a client from scratch but there is a well used a publicised set of classes to do this provided by Oreilly (see Apendix C for Java

package information). Their classes have been released under a free license compatible with this project.

The Oreilly package is "com.oreilly.servlet" but although as the name suggests it was written to support servlets there is nothing to stop its use with other types of Java programs. The HTTP client should be able to be used by both a standard client such as the applet or for inter-site communication handled by the servlet gateways.

Inter-site HTTP

This introduces the problem of how to handle the inter-site communication. If a servlet receives a request from a client requesting information from another site, the servlet must use the HTTP client to pass the request to the correct servlet.

The most elegant solution found to solve this problem was to create a remote monitor driver. This was a monitor driver implementing the same external interface as the standard monitor drivers, but used the client HTTP code to pass the request to another servlet. This hid the behaviour within the abstract monitor and only required one change to the code; The dynamic driver loader was altered to only use the remote driver if the request was for an external site.

This introduced a new data requirement; the remote driver needed to be able to look up a remote sites URL in order to be able to create the formatted Get request required to contact the remote gateway. This was solved in this prototype by creating a simple table in the same relational database used for the product map. It is expected that this information will be stored in some kind of consolidated data store in the next prototype. Figure 3.9 shows the architecture of the system at this stage with the reuse of the HTTP client code highlighted.

Automatic Node Discovery

Some functionality was required to provide each site with a list of nodes that could be monitored. The existing abstract monitor could be used to test whether a resource could be monitored or not. A scanner method was written which loaded the abstract monitor for every IP address on the same subnet as the site gateway (effectively every address on the local LAN) and recorded the IP of any nodes, which could be monitored. A simple table in the MYSQl/JBDC database was used to record the data; this would be replaced with the more abstract data store later in the project.

The scanner was set to run every time the servlet was started. In order to speed up the

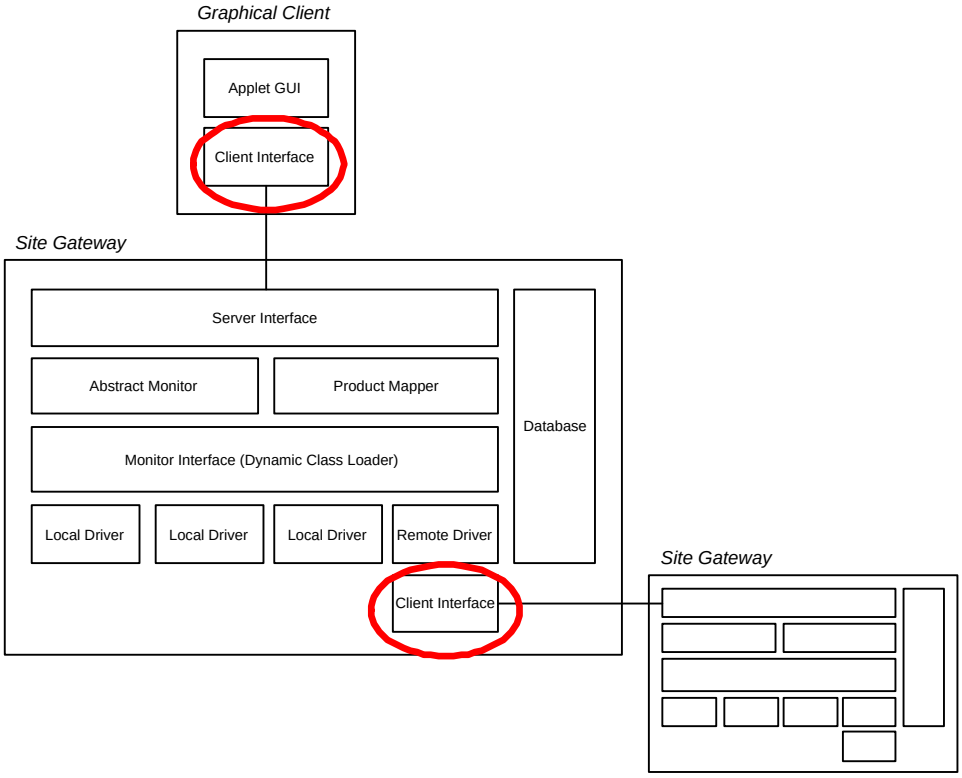

Figure 3.9: Architecture with client code reuse highlighted

scanning process each node was scanned concurrently by using a new thread for each node. This created a dynamic list of local nodes which could be monitored.

There was an extra motivation for setting this up at this stage of the project; the developer did not want to have to take the time to maintain a list of available nodes. The automatic system also kept the list more up to date than the developer would have when the system was running on an un familiar LAN. Figure 3.10 shows the scanner running multiple abstract monitors to look for monitorable nodes.

3.4.3 Deploying The Prototype

In order to develop and test the prototype three sites were used. Each site was connected to the Internet using a broadband comparable connection provided by a different ISP. The goal was to reproduce a real world environment to explore the issues with running such a system.

The networks at each site had one gateway computer which the system was installed on, this

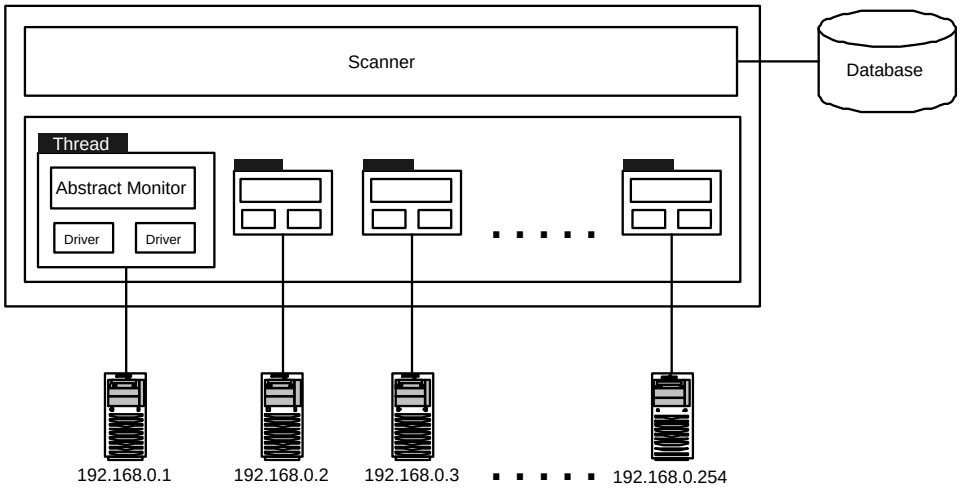

Figure 3.10: Automatic node discovery

is illustrated in Figure 3.11. Each network had different resources connected to it using 100 megabit ethernet.

HTTP Issues

At all three sites there were issues with firewall traversal. In each case the gateway computer was hosted within a LAN where Internet access was provided by NAT (Network Address Translation). This mean that the gateway computer was not visible from the Internet, the servlet needs to have at least one port accessible from the Internet for communication to work.

Forwarding one port from the sites router to the gateway machine solved this problem. Two of the sites had firewalls, which were altered to allow access to the forwarded port. One of these two sites was a trusted private network; to further secure the system the firewall was configured to only allow traffic from the other two site gateways. This had the effect of removing the capability of this site to act as a gateway computer for the applet client - only inter site communication was possible.

In each case known HTTP ports were forwarded, it was possible to use the traditional 8080 port at two of the sites (Figure 3.12). At the other site the router was already using this port so port 8089 was used. This may create an issue with some proxy servers, which are configured to only allow HTTP traffic from the standard ports. With a deployed system it would be desirable to use port 80, which is the default port for the HTTP protocol.

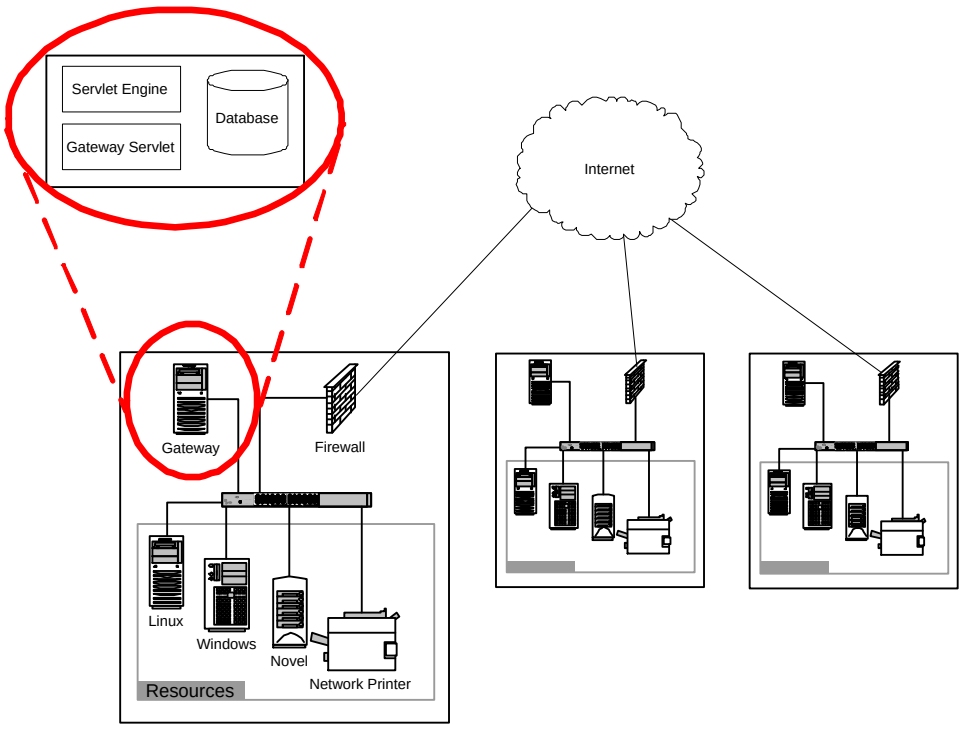

Figure 3.11: The physical sites

The introduction of possible latencies (higlighted in Figure 3.13) when using the Internet to connect the sites was the first test of the robustness of the system. Issues will be discussed later.

Code Synchronisation

When running the system on three different sites while it is still being developed there are possible issues with code synchronisation; When a new feature is added at one site all other sites must be updated to get the new feature, but more importantly compatibility between sites might be a problem if the code is not kept synchronised. While automation can be used to make the process simpler the site synchronisation is laborious and time consuming for the developer.

The early design of the communication interface layers (servlet and client) means that as long as the standard interfaces are not changed the sites can still communicate. All functionality is hidden within the communication layers. This black box design reduces the need to keep sites synchronised leaving it up to the developer to decide when large enough changes warrant testing of new code on all three sites.

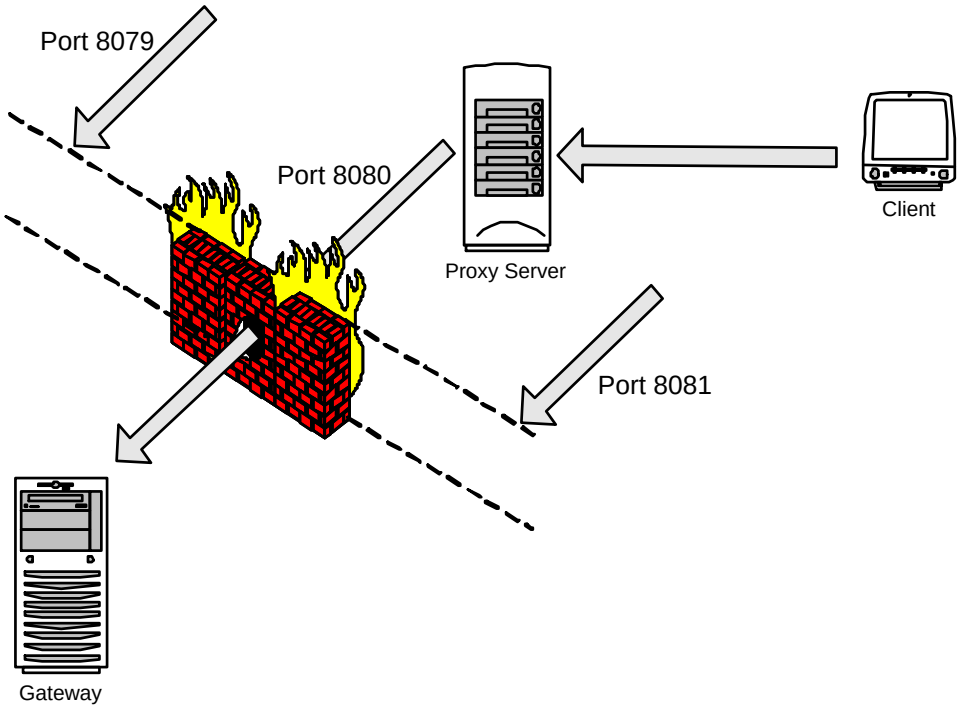

Figure 3.12: HTTP communication with firewall tunnelling

3.4.4 Prototype 1 Summary

The first protoype fully implements the initial architecture outlined in Figure 3.5 in Section 3.2.6. The completed applet client is shown in Figure 3.14, the graph is animating at approximately one frame per second which demonstrates the appropriate timeliness of the artefact at this stage in the implementation process.

The first prototype accounted for a large amount of development time and created a much better understanding of the design. This process proved the overall concept of the system is viable. The second half of the design and development of this system will analyse this prototype with the aim of identifying areas for improvement. The second prototype will address more advanced functionality which build on this framework.

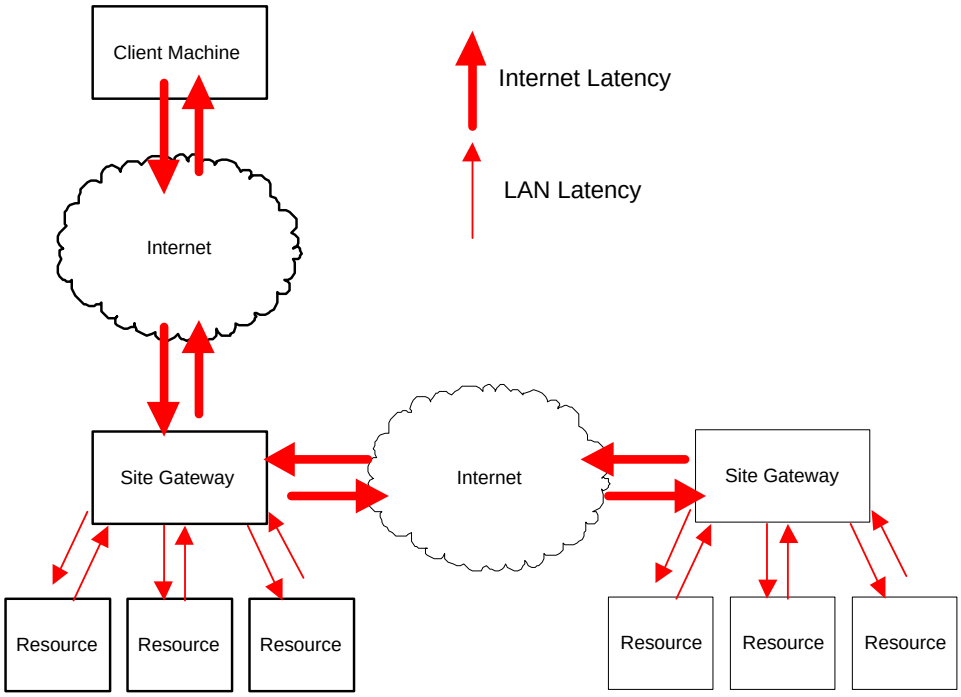

Figure 3.13: Possible latencies within the system

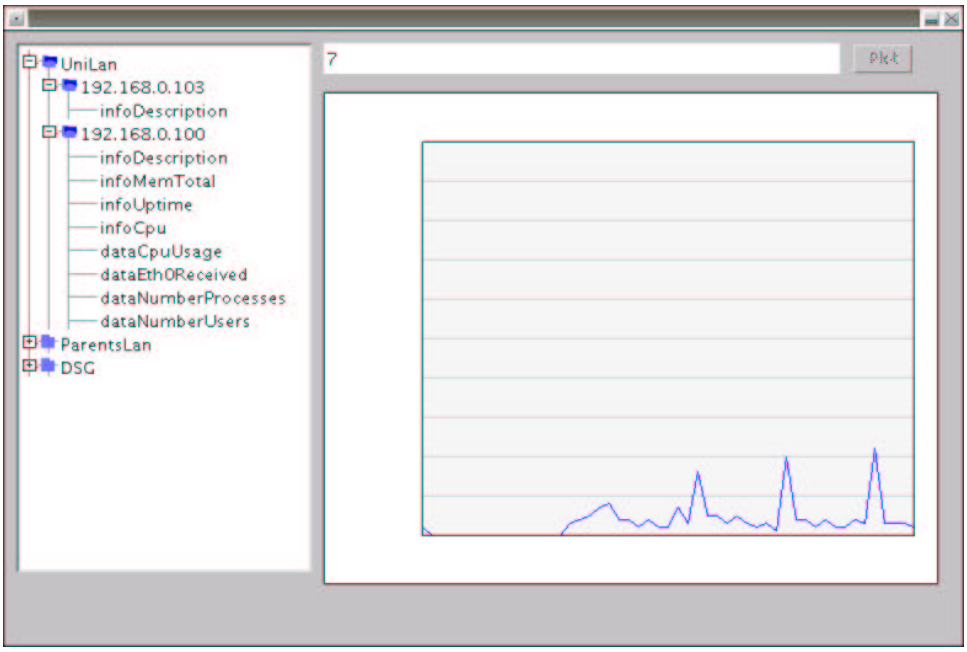

Figure 3.14: Running applet client

3.5 Analysis of Prototype 1 and Specification for Prototype 2

This section will look at issues raised during the development of the first prototype and discuss ways to address them. Going back to the original specification, a new set of requirements combined with the new ones discovered when building the first prototype will be made. They will be analysed to identify possible problems with the design and ways to add extra functionality. This will require another set of stand-alone implementation testing components to prove the validity of ideas - this is the same technique used when developing the first prototype.

3.5.1 Timeliness

An important requirement of the monitoring system is that it can report data in a timely manner. Although the first prototype provided data at around one poll per second (timely enough) it is likely that security overheads will increase latency with later prototypes. The bottleneck in the system is the HTTP communication; it is normal for this to be the slowest component. To address this problem a standard approach of caching data will be used. The aim of the cache is to reduce the amount of communication over HTTP.

Cache Design

While the focus is caching the data retrieved over HTTP, the cache can be used to elegantly create a system to store any kind of data (including local settings such as the site name). The cache must store data locally in a relational database; the cache system will need methods to control access to and updates for the data. Using a TTL (time to live) for cached objects the system will attempt to avoid the problem of cache stagnation where the cached data is not the same as the real data at the source [40]. A problem is that the cache does not know when the data at the source has changed.

There are two mechanisms to address this:

- Time to live (lease time)
- Lease managers [40]

Time to Live (TTL) means that the every cached piece of data (object) has an expiry time. This is an approximation of when the data is likely to have changed at the source. If the TTL has expired the object is removed from the cache. A TTL provides a simple way to minimise cache stagnation. It does not solve the problem completely; The TTL is an estimate of when the object will change at the source, this will sometimes be inaccurate. The two effects on the cache behaviour are:

- It could expire an object that has not changed,
- It can serve a cached object for which the value has changed.

Both of these are undesirable. In general - especially when debugging the system a low TTL is helpful to the developer, this avoids stale objects being passed around the network and minimises the time that the programmer must wait to see the effect of a coding change on the cache.

A lease manager can notify caches when an objects value changes [40]. It does this for caches which have the object stored with a lease that hadn't expired. This means that a cache would know that any objects it has stored with valid lease times (TTL) are correct. This avoids the problem highlighted above, but the process of implementing a lease manager is complicated. It creates an overhead at each gateway where all data served must be tracked to insure that the manager can check whether changing data affects any remote caches.

The Effects of Statelessness on Timeliness

Every transaction the servlet processes involves the creation of a series of objects to satisfy the request. The stateless methodology means that these objects are destroyed after the transaction is completed in order to return the application to the starting state. The creation of new objects for every transaction is slower than if some of the objects were reused (a statefull design).

In order to reduce the total object creation overhead and method execution time, while following the stateless philosophy, the developer has some options:

- Reduce the number of objects needed to satisfy transaction,
- Optimise objects and their methods to load and execute faster,
- Reduce the complexity of objects.

Extra functionality provided by the cache for the second prototype is to cache local data as well as remote data. The remote cache must be built and tested before this idea can be explored further as it requires an understanding of the speed and overhead of using the cache. The potential use of the cache for both local and remote data must be considered while designing it.

subsectionA Universal Data Store

The system has a need for a back-end data store capable of holding various types of information. The data which required storing in the first prototype was:

- The list of monitored resources nodes,
- The product map,
- The URLs of remote site gateways.

Producing the cache will create a further data requirement:

- Remote and local data cache

There is the potential for the system to be caching remote data for which it has no product map. For instance a remote site may have monitor drivers which expose different data than that of a local site. If the local site caches this remote data it will not have a product map to describe the data - the data store should avoid having to keep all product maps in sync. The means the stored information must be self-describing, a general schema can then be used. The current simple local product map database will be retained as it avoids having to synchronise caches between sites. All other data will go into the universal data store, which will have to store a general schema, the relations between the data to provide context and the data itself. Since a lot of the data will be either a local or remote cache it must also store the lease time.

This universal data store could also be described as a relational database of relations. The design of the data store will be discussed further during the next development stage when a standalone component will be built to test this idea.

3.5.2 Security

In the first prototype, the security requirements of the specification were ignored to reduce the complexity of the client server communication code and reduce the number of features

which needed to be implemented.

The two security requirements laid out in the original specification were encrypted communication and restricted access to the system.

Secure Communications

The communication code written for the first prototype used the HTTP protocol. This data was unencrypted which allowed data to be sniffed by any node it traversed on the Internet, possibly even worse it could be changed in transit. Which means it is vulnerable to both passive and active attacks. The second prototype should implement HTTPS (HTTP with SSL) in order to prevent these kinds of attacks.

Restricting Access

Access restrictions can be split into two catagories:

- Restricting any access to the Grid Monitor system
- Access control lists for individual resources

It is desirable for administrators of a site participating in the Grid to be able to grant access to their resources rather than have a central authority delegate. In order to have a decentralised and scalable system the security must be distributed. This means that logon credentials must be spread throughout the system. It is expected that a deployed system may use features of the standard Grid Security Infrastructure (GSI) [20], the authentication scheme should be easily extendable to allow for back end authentication to be changed.

If the assumption is made that users of the system also have a site they are associated with, the security information for that user could be stored at that site (their home gateway). When a security check is required a home gateway can authenticate one of its users locally. If the user connects to a remote gateway that site will contact the users home gateway for security information, this information could be locally cached to reduce the communication overhead.

ACL lists can be created to grant users access to resources. Implementing fine toothed ACLs would be important for a production system but this second prototype will only restrict access to the system, not individual resources. This feature would take a disproportionate amount of time for the developer to implement when the main focus of this project is not security.

Choosing a Unique Username

This introduces the problem of uniquely identifying a user within the system. In order to allow a remote security event for a user, a server must be able to derive the users home site from their credentials.

A simple solution is suggested:

A user name is made up of two parts user@site. Where 'site' is the unique name of a grid site and 'user' is a unique identifier for a user at that site. As an example of this these usernames would map to a unique user within the Grid:

```
mat@DSG
mat@HomeLAN
mab@HomeLAN
```

This makes it possible to only need a username to both uniquely identify each user and derive their home site to do authentication.

Type of Logon Credentials

A string password is used with the username to authenticate a user. There are more complicated authentication mechanisms using certificates. The aim of this project is not to create an authentication infrastructure but it must demonstrate a reasonable degree of security.

With encrypted communication it is acceptable to use plain test usernames and passwords as they will be encrypted during transport.

Parameter Checking

The HTTP Get requests used by the client to invoke methods on the server contain variables, which are used directly by the servlet. There is no sanity checking done by the server before it uses this data. This is dangerous from a security perspective as it allows an attacker to craft specific URLs designed to manipulate the server in a non standard way. Some effects of this can be:

- Denial of service - crashing the servlet,
- Privilege escalation by manipulating remote data,

- Potentially extracting information from the data stores which are not meant to be public,
- Gaining access to the machine running the servlet,
- DOS against other machines within the site via the monitors.

These are all standard issues when writing code which accepts user input or allows a user to manipulate the input. In a production system these issues would have to be addressed. In this prototype they are acknowledged but ignored, as hardening a system is a complicated process involving auditing the code and thorough testing.

3.5.3 Clients

The applet client demonstrates dynamic visualisation of the system. In order to show the abstract nature of the interface and the portability of the communication code, an HTML client will be developed as part of the second prototype. This interface will take the form of servlet running the same client code as the applet and displaying the data as a web page.

This client also allows demonstration of the system on client machines where a Java VM is not available.

3.6 Components for Prototype 2

This second prototype will be more functional than the first. In order to explore new ideas and test implementations same methodology used earlier of creating stand-alone components to test ideas will be used.

3.6.1 Relational Data Store

All data used by the system except the product map will be stored in one database. The traditional way to do this is to use a fixed schema database to store records. This does not allow flexibility to add new kinds of data without changing the structure of the database. The proposed approach is to store the relations between the data in the database with the data. This provides a way to describe the data by walking the relations.

Index Table

The database has two components; a data table and an index table. The index stores names of any entities which will be store in the database. At a low level these could be 'String' or 'Date'. This would be a more abstract view of the database. Because this data store is going to be used specifically for this system the index can be set up to contain a list of higher level entities.

The entities that exist are:

```
siteName
localSiteName
siteServletUrl
nodeIp
nodeName
productName
username
userPassword
```

The Java variable naming convention is used to be consistent with the implementation. All items in the database are classified by one of these entity names.

Data Table

The data table stores the data, the relations it has and a lease time, which is the cache time to live. If data being stored is does not expire, a lease time of 0 can be used. The database is storing a tree data structure in a matrix. If a single record is looked and only the data is shown, it has no context. The tree data structure must be navigated in order to give any data other than a root node context. The table holds two relations, one is the link to the data class in the index table, and the other is a link to the parent node. It is possible for a parent to have multiple children but a child can only have one parent (one to many relationship). Example relations are shown in Figure 3.15.

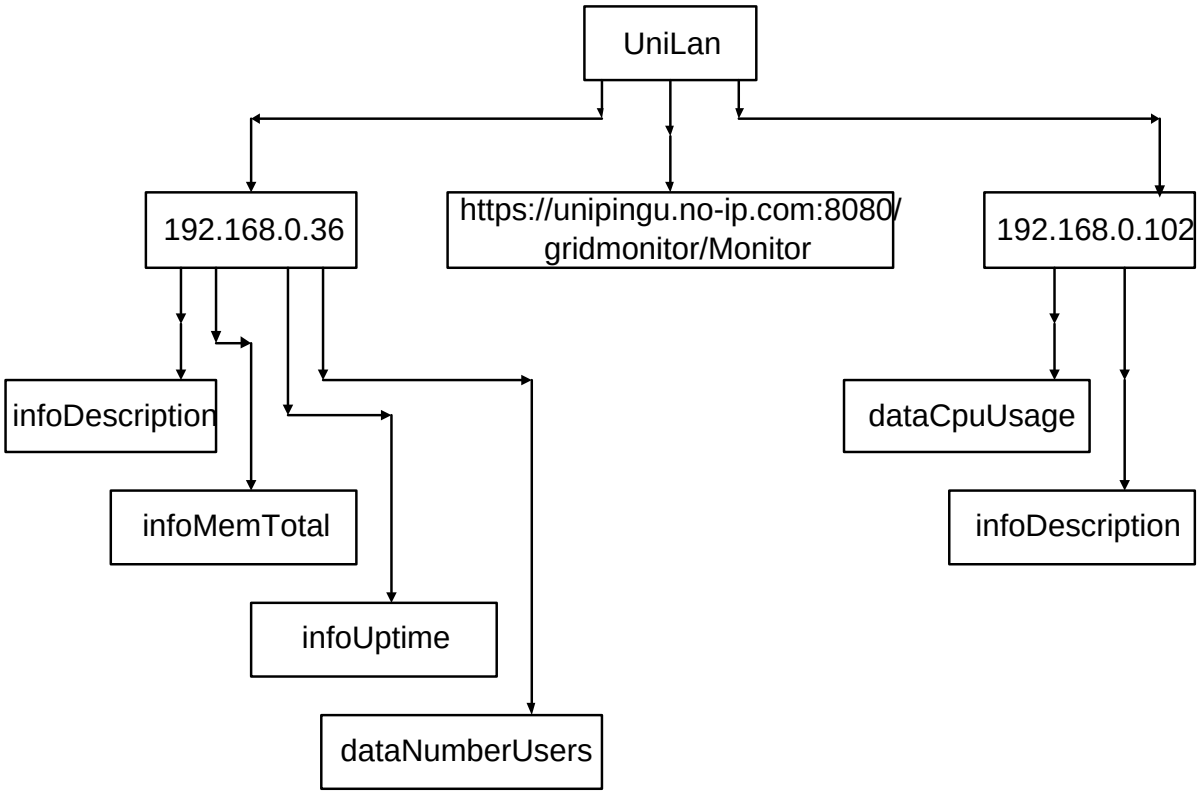

Figure 3.15: Example cache objects and their relations

Traversal of the Data Store

To run a query on the data store all of the parents of a child must be known. The query is run as a set of sequential SQL statements which walk down the parent nodes until the child is found. This was implemented using a simple, powerful recursive algorithm which is

included below:

```
// function to recursively query the datastore for a value
// each item on the stack is a dataId to a child
// and a linkId for the type of entity
// the last item on the stack is the child querying for
// so it has no relation link as it has no children
public static Vector recursiveGet(Stack theStack, String lastDataLinkId){
  String values[] = (String[]) theStack.pop();
  if(!theStack.empty()){
    // run the sql for the query
    Vector keyAndLease = RgmbQuery.getKey(values[0], values[1],_
      lastDataLinkId);
    if(!keyAndLease.isEmpty()){
      // if all went well with sql set the link id for the next child
      lastDataLinkId = (String) keyAndLease.get(0);
      // and call self again
      return recursiveGet(theStack,lastDataLinkId);
    }
    else{
      // wasnt found, return with empty
      return new Vector();
    }
  }
  else{
    // base case has been met
    //return the value of the item linked with DataLinkId
    return RgmbQuery.getValues(values[0],lastDataLinkId);
  }
}
```

3.6.2 Cache

The data store allows a lease time to be saved with every record. In order to use the cache a layer needed to be implemented through which all data traversed. The abstraction was added by inserting a layer between the monitor driver manager and the servlet front end. This meant that any query for data went through this layer, the changes to the API are outlined in Figure 3.16.

Choosing Lease Times

This is potentially a complex issue. A best-case lease time would expire exactly when the data looses integrity and is no longer accurate. A cache can be programmed to adapt its lease times to try and get as close to this best-case scenario as possible. One way it can do this is by recording data about how often a lease time was accurate and by how much and adapting the lease times to try and make them more accurate. Another method is to use historical data to try and predict the most accurate lease time of a piece of data.

A fully developed system would have a lease manager which would handle notifications to other caches and choose the lease times for objects. With no lease manager cache uses hard coded lease times for all objects. The exact value was varied during development but ten minutes was used for all objects except monitored product values which were set to not be cached. This was done to simplify testing the system, the addition of the cache introduced another element to the debugging environment, a more dynamic cache creates potentially hard to track behaviour. Ten minutes is high enough that the developer always new what was in the cache while testing.

3.6.3 Web Client

The web client is a servlet based interface to the system which allows, via a clickable HTML interface, access to the system without the need for a client side Java Virtual Machine. A very simple navigation system was used representing the Grid as a hierarchy which could be navigated through. Data is not polled for a user, if they wish to renew their view of the system they refresh the web page. Apart from the benefits of allow easy demonstration of the system to people with limited control over the software available on their pc the web client demonstrates reuse of the client code developed with the applet client. Very little new code was required to create the web client, most it was HTML generation. A screen shot of the web client is shown in Figure 3.17, it is included for completeness.

3.6.4 Secure Sockets Layer

Moving from HTTP to HTTPS. The developer had known that this would be required at the end of the project, the reason it was left to be one of the last things was that it is much easier for the developer to debug communication code using the plain text transmissions of HTTP. HTTPS is encrypted which means no remote communications could be inspected during programming, which was not desirable.

Server Configuration Changes

The Java servlet framework provides all of the socket communication handling for access to servlets. The first stage of the conversion was to configure the Tomcat servlet engine to use an SSL socket instead of a plain text one.

SSL communication happens in a trusted environment which uses X.509 [3] certificates to prove a servers identity. The certificate is signed by a trusted authority (at a monetary price) and a client will trust the certificate as long as it matches the server and has been properly signed by the trusted authority.

The developer could not afford to get a certificate signed for the project. In a development environment it is common for a developer to self sign the certificate (doing the job that the trusted authority would). This creates a valid certificate but remote clients will not trust it by default. A self-signed certificate was used for the SSL socket of the Tomcat server, a web browser fetching a page over this socket would prompt a user to view the certificate as it was not trusted. In this case the user makes the decision about whether to trust the certificate. In the context of this project it means that any user of the web client would need to 'ok' a dialogue box before they could use the SSL enabled client.

Another issue with the certificate was the server host name. In an X.509 certificate the server name is listed. A client checks that this name matches the DNS name of the server it is talking to. Within the development environment some of the servers running the servlets had two different host names for remote and local access. This meant that the host name in the certificate would not match the server name on both the internal LAN and the Internet.

A web browser adds a warning about this and as with a self-signed certificate the user makes a choice about whether to accept the certificate. This would not be an issue with a deployed system but was present because of the developers home LAN. The problem and the browser behaviour are shown in Figure 3.18

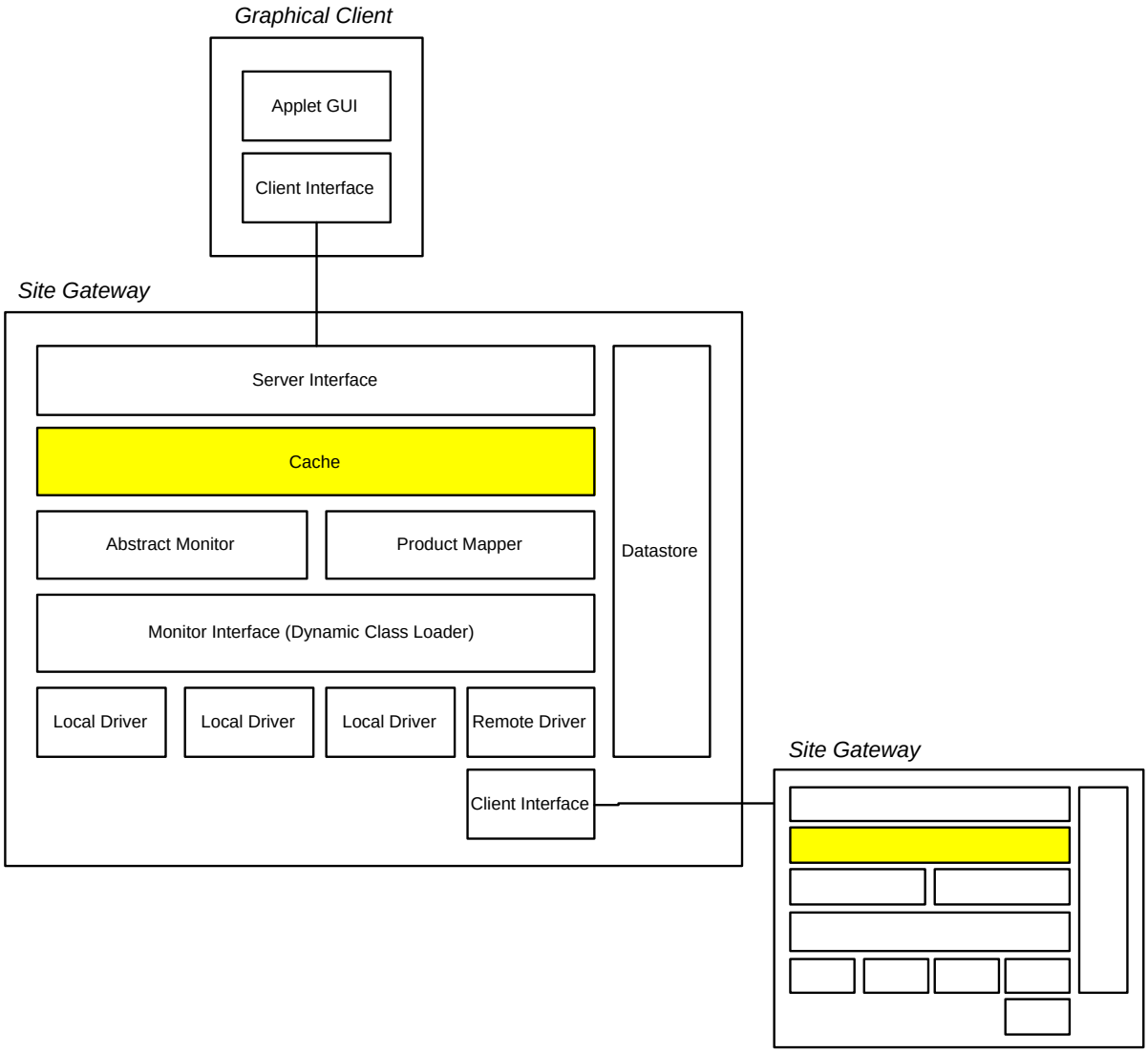

Figure 3.16: Revised architecture with new cache layer

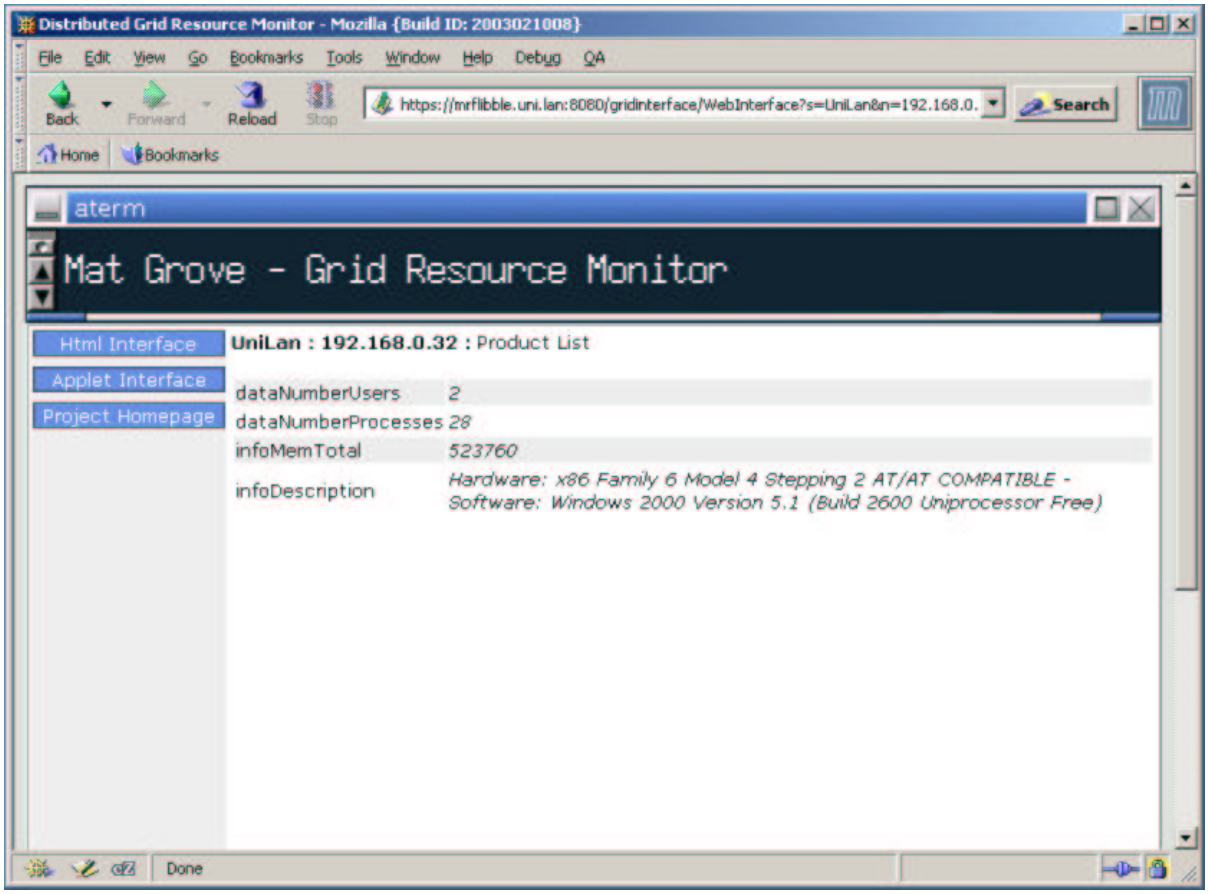

Figure 3.17: Web client monitoring node 192.160.0.32 in site UniLan

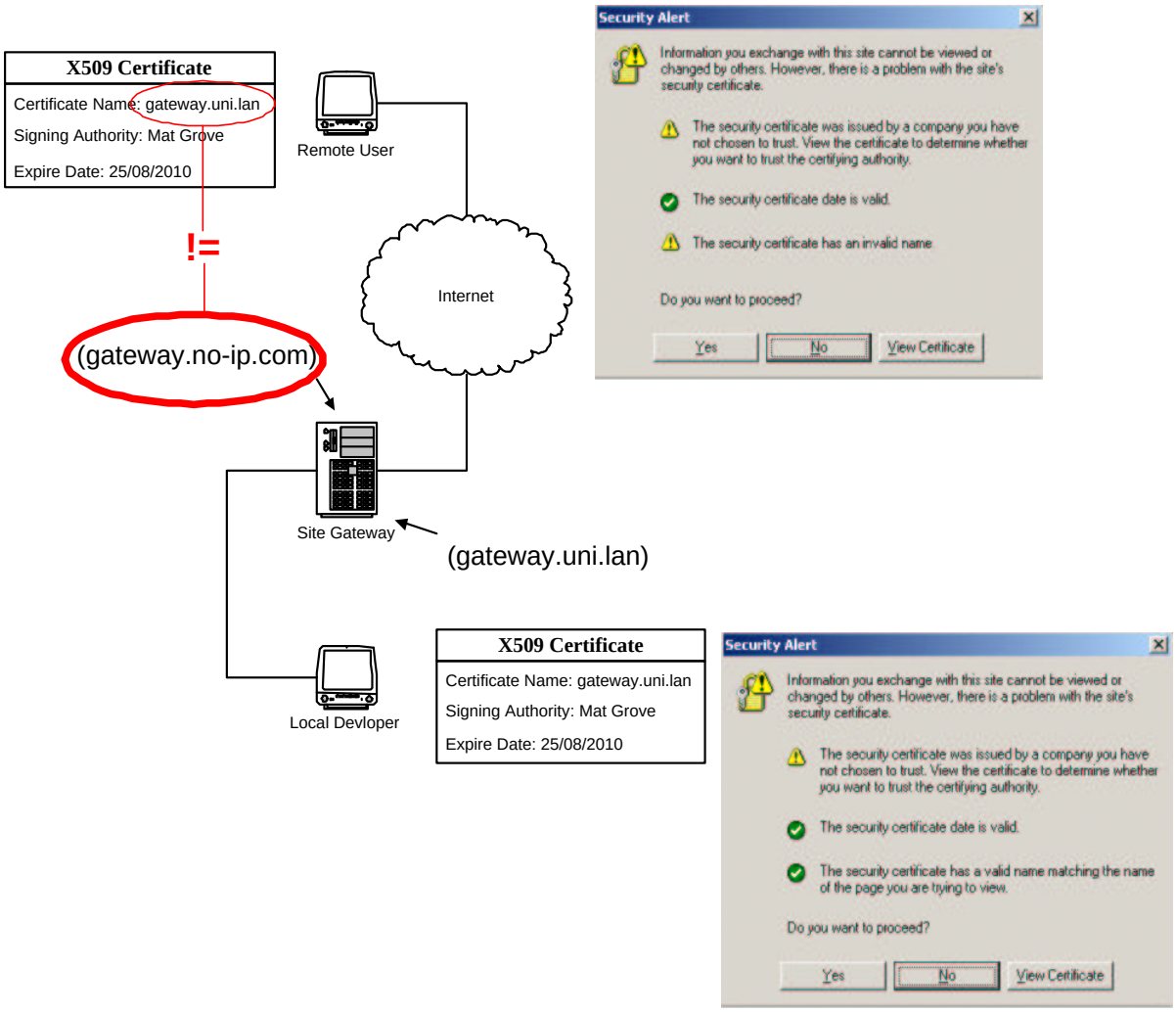

Figure 3.18: Certificate checks and browser dialogue boxes

Issues Changing Communications Code to HTTPS

Changing the client code to use HTTPS was trivial because of the way communications was encapsulated, but there were serious problems involving the certificates.

As described, if a web browser cannot automatically validate a certificate it falls back to asking the user to make the decision. When Java code is being used to do the HTTPS communication it is not possible to ask a user to validate the certificate (especially when the communication is servlet to servlet - there is no user involved).

The developer must implement code that changes the behaviour of Java to override the certificate checks, which would fail even though as far as the development system is concerned the certificate is valid. A Promiscuous X.509 Trust Manager was written to validate the self-signed certificates. The new trust manager validated the certificates by altering some of the checks.

At this point the test component HTTPS client could use the new communication code to connect to system over HTTPS.

3.6.5 Access Control

The stand-alone access control component must be able to parse the username into user and site. When the second prototype is assembled the servlet must have its interface changed so that every client query also has the username and password sent. Because the servlet is stateless every transaction requires the logon credentials. This is another candidate for using the cache to reduce remote communications as authentication checks may be done at a remote site over the Internet - a potential remote authentication for every query from a client would impact the timeliness of the system.

3.7 Implementing the Second Prototype

The design and implementation of code up to this point has allowed in depth analysis of the issues of creating the system. This final prototype should support the overall design philosophy and be as fully featured as possible.

3.7.1 Change of Internal API Structure

The addition of new features added during development create new requirements for the API. At this stage of the project it is possible to build a more abstract system using a cleaner implementation made up of the best ideas from previous work. With any developing system which evolves, changing requirements mean that at some point it is better to stop adapting the old system and create a new framework.

Data Cache

The development of the internal data cache changed the flow of information through the system. This is a fundamental change to the design but creates major benefits to the timeliness of the system.

Authentication

The addition of front-end authentication required simple changes to the servlet interface. The more interesting opportunity it presented was to make the authentication back-end use drivers, in much the same way as the back-end monitoring system. By abstracting authentication remote and local methods could be hidden bellow a manager the same way that the monitor drivers are hidden bellow the abstract monitor interface.

If plugins could be used for authentication then the system would be much more extendable. The initial idea was to have a user authenticate against a local database as part of the data store. Authentication drivers would mean that any authentication system could be used such as a Windows NT Domain Controller or Globus GIS. The assumption is made that some Grid sites already have an authentication system, or use the developing grid standards.

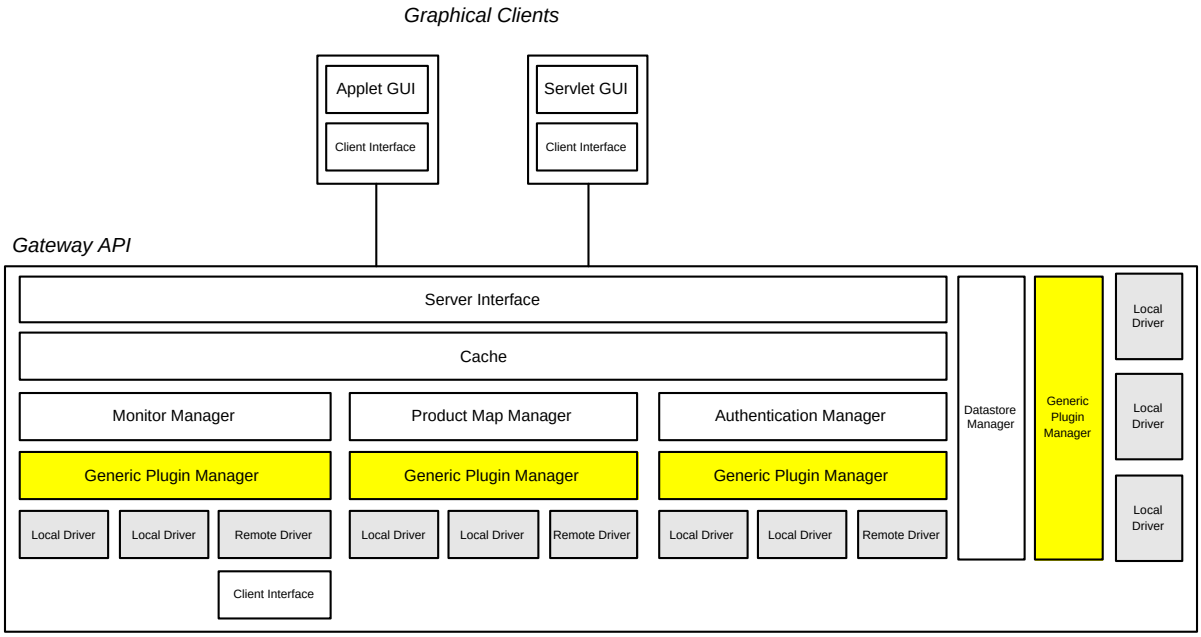

Figure 3.19: Fully pluggable API with plugin mananger

3.7.2 Fully Pluggable API (Generic Plugin Manager)

The features that are plugable in this design are:

- Data store,
- Monitors,
- Authentication,
- Product Map.

A plugable data store has not been discussed. It means that a dynamic driver can provide the back end database, which provides the services for the cache. This project used mySQL but by abstracting it allows for it to be easily changed to using any other kind of database (data store) like R-GMA or an Oracle.

Each plugin category has an abstract manager, this manager uses a generic plugin manager to load dynamic code. A generic plugin manager is a very powerful component. The changes to the API are outlined in Figure 3.19.

When dynamic code loading was previously explored the main issues addressed were:

- Knowing the names of available plugins,
- Loading the code (using introspection).

The naming problem was solved in the first prototype by hard coding the plugin names. This is not ideal and was only done by the developer as a simple fix. It is desirable for the plugin manager to be able automatically discover what plugins are available.

When only loading one type of code (the abstract monitor loading only monitor drivers) classification of plugins is not an issue. With a more generic system there needs to be a system by which the plugin manager can distinguish between different classes of plugins.

The plugin manager must have an interface which allows the program to specify which class of plugin is required. It will also need to be able to look for a specific plugin implementation (such as a remote driver).

Naming and Dynamic Discovery

Within the context of dynamic plugins the four classes are:

- DataStore
- Monitor
- Security
- ProductMap

These are grouped by implementation; each of which can contain one of more plugin classes. The implementation type describes the technology used to provide the plugin. Examples of implementation types are:

Implementtaion	Description
jcifs	Windows NT domain controller access
local	Anything specific to a site
remote	All remote communications
snmp	Plugins using the snmp monitor

Table 3.3: Implementation types

Some of the implementation types can provide plugins of more than one class. The best example is the "remote" type which could potentially provide all of the plugin classes.

By way of example a list of implementation types and plugin classes which were actually implemented is shown:

Implementation type	Plugin class
jcifs	Security
local	ProductMap
remote	Security
remote	Monitor
snmp	Monitor
procmon	Monitor

Table 3.4: Implemented drivers

Physically storing the plugins in a directory structure, which follows this classification system allows the plugin manager to discover available plugins by doing a normal directory listing (see Figure 3.20). A filter is used by the discoverer to return specific plugins allowing fine-grained selection, such as "all monitors except the remote implementation" or "only the remote security plugin".

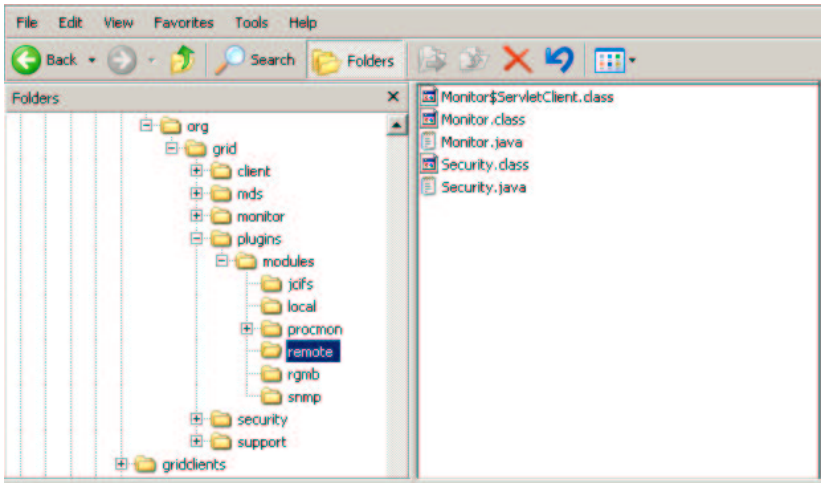

Figure 3.20: Screen shot of plugin directory structure showing the plugin classes for the 'remote' implementation type

Dynamic Loading and Dynamic Access

Inheritance and introspection are used to create the dynamic objects from the plugins which static code can then access. Inheritance is used to provide interfaces to the dynamic code - this means that when the plugin manager loads an object it casts it using a plugin class definition, this describes the methods which all of the plugins of that class must provide.

Summary of Plugin Manager

After converting the system to using the generic plugin manager, adding functionality like extra monitor drivers becomes extremely simple for the developer. Debugging while using dynamic code is more difficult for a programmer, as formulaic code execution is not always obvious - the order in which the plugin manager returns plugins was designed to be consistent to try and help the developer.

3.7.3 Adapting the Servlets to use SSL

The HTTPS client code described in the last section was written and tested as a stand-alone component. The methodology of developing and testing code and then porting it into the prototype had worked for every component built. Replacing the HTTP communication client that the servlets (the web client and the main server) should have been no different.

The developer found that the same code called by a stand-alone program was calling different libraries to when the code was called by the servlet. This was an interesting implementation issue. After tracing code execution it was discovered that a servlet creating a HTTPS URL by default will use the function:

```
com.sun.net.ssl.internal.www.protocol.https.HttpsURL_
  ConnectionOldImpl.getInputStream()
```

While a stand alone component would use:

```
sun.net.www.protocol.https.HttpsURLConnectionImpl.getInputStream()
```

This highlights a problem with implementing and testing components outside of the environment that the prototype would execute in. The developer could not find any documentation about this behaviour. The solution was to force the servlets to use the same SSL libraries that a stand alone component would with a simple Java command:

```
System.setProperty("java.protocol.handler.pkgs","javax.net.ssl")
```

3.7.4 Summary of Prototype 2

All of the components and previous work contributed to the implementation of the second prototype although the major change to a fully pluggable API meant a great deal of small changes were required for existing code.

The completed second prototype demonstrates a working distributed grid monitor. It uses dynamic code loading to provide back-end functionality to be extended easily, this means that it can be deployed on systems with non-standard configurations. Although Java itself is portable it is often the case that systems have prerequisites which make the porting of an application difficult.

An example is the RGMA system which will only install by default on to Redhat Linux distributions with mySQL. RGMA can be adapted to run on any Linux distribution and the code could be altered to use a different relational database but the developers do not make this easy.

It is important for the adoption of a Grid system that it be easy for a site administrator to install, use and control the system.

Chapter 4

Evaluation

4.1 Evaluating the Final Prototype

A continuous process of testing was part of the iterative design and implementation methodology followed. Debugging a distributed application can be difficult. With software running in more than one location the approach the developer used was to have multiple debuggers running for each instance of the software. Clocks were synchronised between machines and time stamped logging was used to create a view of the execution of the distributed system.

This chapter aims to evaluate the final the prototype and the architecture design which it was developed to prove the concept of.

4.1.1 Comparison of Features to Specification

The completeness of the functionality of the second prototype can be measured by comparing it's features the original aims set out in Chapter 1. Some new requirements were added during the design and implementation stages, these will also be analysed.

Abstract Monitoring Layer capable of monitoring via Multiple Protocols

The final API contained an abstract monitoring interface that provided dynamic monitoring of resources using available monitors. This aspect of the implementation spurred the use of a fully dynamic pluggable API, which can load Java classes at runtime. This functionality was extended to allow more of the program to be abstracted in a similar fashion.

Heterogeneous Execution and run on most Grid Hardware

The system was only deployed on Linux x86 based hardware. This was a limitation of the resources available to the project. The use of the Java programming language and Tomcat servlet engine would in theory allow the system to be deployed on any hardware with a Java VM.

During implementation of the dynamic code loading care was taken to insure system independence. When dealing with lower level functions (such as file handling) it is easy for a developer to fall into traps, which make the code un-portable. The classic examples are file descriptors, which use different syntax on different operating systems. For example c: on windows does not exist on UNIX based systems. Where appropriate the developer used Java system calls to programmatically discover properties of the running system, this allows the program to avoid using hard-coded system dependent syntax. With the example of file handles, the Java VM can report the delimiter for separating files and directories on the host machine, under UNIX it would report "/" and under Windows "
". This is an important feature and was not overlooked while programming.

The Procmon example monitor would only work under UNIX operating systems which had the proc file system available. This is not a limitation of the system as Procmon is only intended as a test program for developing the artefact.

Distributed Security Model

The prototypes security API is pluggable in the same way that monitor drivers are. This allows for any authentication mechanism to be used. It is expected that in a deployed system a Grid security system would be plugged in (possibly GSI [20]).

The system supports remote and local authentication of users at site gateways, which means that a user can connect to any gateway. They would not be able to authenticate if their home gateway with their logon credentials was not available. This would be avoided if a distributed security system to be used as a pluggin. This could also be handled through some kind of replication of security information throughout the system.

Encrypted Communications

All remote communication is handled by HTTPS which means it is encrypted using industry standard techniques.

Firewall Traversal

By using HTTPS for transport the system uses a standard firewall traversal technique. The prototype sites did have some problems where they were using HTTPS on non-standard ports. Some application proxies are configured to only allow HTTPS communication on port 443 (the ISO standard HTTPS port). If the system was deployed it would be desirable to use this port to allow maximum firewall penetration.

No Single Point of Failure

Each site gateway is completely self-contained. If a user is connected to a site and requests information from another site which has become unavailable the system will handle the failure gracefully. The user can repeat the request as often as they choose. As the system is stateless repeating a failed method is safe.

A user cannot connect to the system if their home is not available as they will not be able to authenticate. While this is a single point of failure for that particular user the actual system remains functioning regardless of the number of sites, which are available.

The list of other site URLs is hard-coded at each site, thus replicating the data at each site. If the system was altered to automatically discover the sites it would need to use a distributed mechanism to avoid introducing a single point of failure such as a centralised lookup service.

4.1.2 Universal Data Store

This requirement was added to the initial specification during the design and implementation stages. It is a set of code and a MySQL database which allow any relational data to be stored. Adding new relations and data types is possible without changing the structure of the database.

4.1.3 Maximum Acceptable Latencies (Timeliness)

Figure 3.13 in Section 3.4.3 shows the possible sources of latency in the system. The HTTP timeout length between the client and the server governs the maximum length of time that can be taken to process a request as this is the first HTTP request to be issued. This value is well above what could be called timely; a more realistic value for a user to wait for a request

would be 5 seconds. Using this value it is possible to calculate the maximum acceptable latencies within the system, but first the different ways that a servlet can process a request from a client need to be described; Figure 4.1 shows the decision tree that the servlet uses when handling a request:

The bottleneck within this system is the remote communications over the Internet. A maximum latency for remote communication can be calculated. The time taken for each process that may make up a request is measured. The worst-case scenario is created for all local requests.

The local processes of the request always took approximately 50 Milliseconds to complete, this includes the time for up to 5 cache look-ups and one dynamic monitor and one authentication.

LO = Local overhead
MT = The maximum time to complete the whole request
NR = The number of remote communications required to complete the request
RC = Maximum latency of each remote connection

Worst-case scenario (most remote requests):

1. Client requests to server
2. Gateway does remote authentication
3. Gateway forwards request to remote gateway
4. Remote gateway monitors node

MT = 500 Milliseconds
NR = 3
LO = 50 Milliseconds

RC = (MT . LO) / NR
= (500 . 50) / 3
= 450 /3
RC = 150 Milliseconds

The maximum total latency for the remote communications in this case is 450 Milliseconds and the average maximum latency for each part of the remote communications is 150 Milliseconds.

4.1.4 Automatic Node Discovery Issues

The automatic node discoverer shown in Figure 3.10 in Section 3.4.2 suffers from the simplicity of its design. The scanner creates a new thread with an abstract monitor for every IP address on the same subnet as the gateway.

This means that the scanner will only ever find nodes on in the same IP block as the gateway. It is assumed that a deployed system would use a more intelligent mechanism for choosing which IP addresses to probe.

The scanner also suffers from a design flaw which was not found until it was deployed across all of the sites. The primary site gateway which was used for development work was a much more powerful machine than the other two test sites.

Primary site: 800mhz Athlon 128mb Ram
Test site 2: 400mhz PentiumII 64mb RAM
Test site 3: 200mhz Pentium 32mb RAM

The scanner had no rate limiting when spawning each new thread. On a 24-bit IP block there are 254 addresses. The scanner created 254 new threads as fast as it could. The Athlon handled this fine. On the other two test systems an explosive consumption of all system resources was observed. This resulted in the creation of new threads failing on those machines. A simple fix was introduced which added a time delay of one second between thread creations within the scanner. This prevented the consumption of all of the resources as some threads completed before enough new ones were started to consume all available memory.

A more advanced approach in a more sophisticated scanner would be to set a limit on the number of concurrent threads and only allow new ones to be created when one had finished.

4.1.5 Maintainability and Extendibility

The pluggable API with dynamic code loading makes the architecture extremely extendable. New drivers for monitors, database back-ends and authentication can be added without

changing the main servlet code.

The example client interfaces were developed using a standard client communication library; any new or extended functionality clients would use this library, which abstracts the interface to the servlet.

All of the programming uses standard software techniques suck as object orientation, abstraction and encapsulation in order to produce self-contained code. Hiding functionality through object orientated techniques allows a developer to tackle the parts of the implementation they wish to change without necessarily having to analyse all of the program.

4.2 Testing the Final Artefact

While constant iterative testing proved the correctness of the code during development, it is important to evaluate the final artefact. The behaviour of the system in real use is described here.

The client interfaces to the system do not allow any data to be inputted by the user. They derive all of the information from the system by interacting with the site gateway and then allow the user to choose what they wish to do. This simplifies the testing of the system, as the information presented cannot be invalidated by user error.

The data cache provides a buffer for the loss of connectivity between sites. To test the Grid was fully explored through the user interface and then one of the Grid sites was shut down. The view of the Grid remained the same even though one of the sites was not on. The site was restored before the cache TTL expired and the system continued to run normally.

If a user is connected to a gateway that looses connectivity, or the users Internet connection has a problem their client will stop receiving all information and the client displays a network error message. The user can either wait until the site becomes available or connect to another gateway.

If a user tries to start using the system while their home gateway with their security credentials is not available (and the information is not cached) the user receives a user name password error when they should see a message explain that their home gateway is offline - this is an oversight.

4.3 Analysis of Final Architecture

The project produced an architecture for an Abstract Grid Monitor, Figure 4.2, the work to develop the prototype proves the viability of it (the prototype is a working example of this design). Critiquing the architecture, one of the key design aspects is both its main strength and weakness; the stateless philosophy.

The stateless design allows for an extremely robust prototype. It would be very hard to increase the efficiency of the local services such as the cache and monitoring without changing to a state full design, which would allow:

- Easier implementation of a lease manager (which must track all leased data and handle notification of changes to remote caches).
- Persistent monitors removing the performance hit of having to do full discovery on a node every time a value is polled. A driver pool of available monitors would be more efficient. It would also allow the drivers to do lower level caching as they can preserve their state.
- The possibility of streaming data (a persistent connection between client and server). This may be more efficient in some circumstances than repeated polling by a client.

4.4 Evaluation of Project Management

The design and implementation of such a functional prototype was ambitious. The redesign of the architecture between the first and second prototype was expected from the beginning. However the dynamic code and pluggable API were extremely difficult for the developer to implement, which put pressure on the original milestones of the project outlined in Section 1.4.

The failure to implement a fully functional leasing cache (which was not in the original specification) was due to time constraints.

The project prototype and architecture would benefit from one more iteration of design and implementation process outlined in Figure 3.1 on Section 3.1.3. The next logical step would be to use the tested robust architecture to create a more feature rich and efficient system based on a state full model.

The stateless philosophy made producing the prototypes possible within the time frame. It is unlikely that the developer would have been able to complete a state full architecture, which was as robust (or even nearly as robust) as the stateless one is, within the time frame of the project.

4.5 Summary

The iterative design and implementation process leveraged the developer's programming experience to create a working system. The problem was tackled from a practical point of view, with the goal always to create working software. This acted as a motivator for innovation from the developer as design and implementation issues were solved.

The final prototype fully implemented all goals set during the project, it is hoped that the report will aid future work in the Grid monitoring middleware area of study. Some parts of the implementation, notably the dynamic code loading and universal data store, stand out as advanced components, which are likely to be reused in future projects at least by the developer and perhaps by other groups.

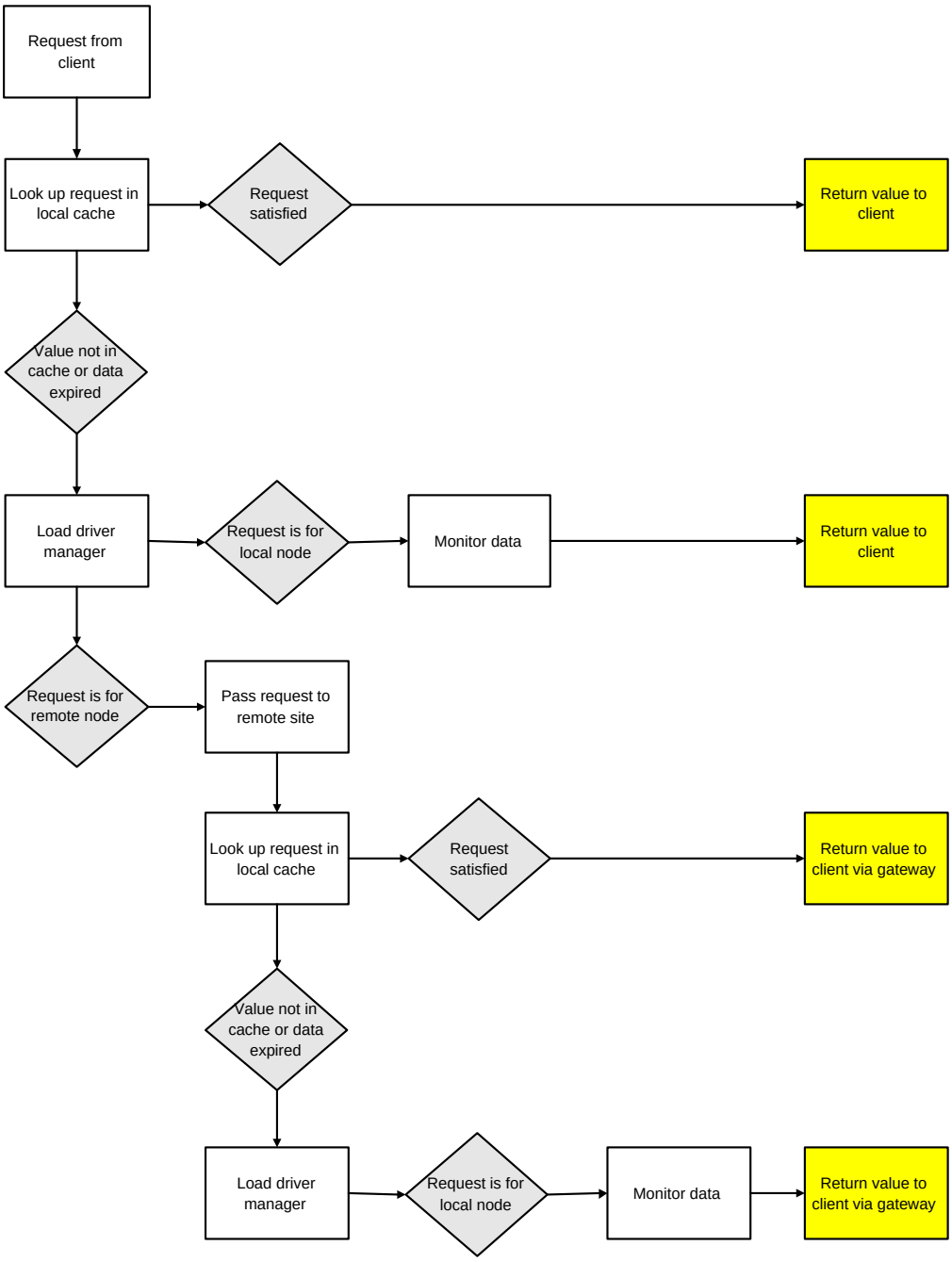

Figure 4.1: Servlet request handling decision tree

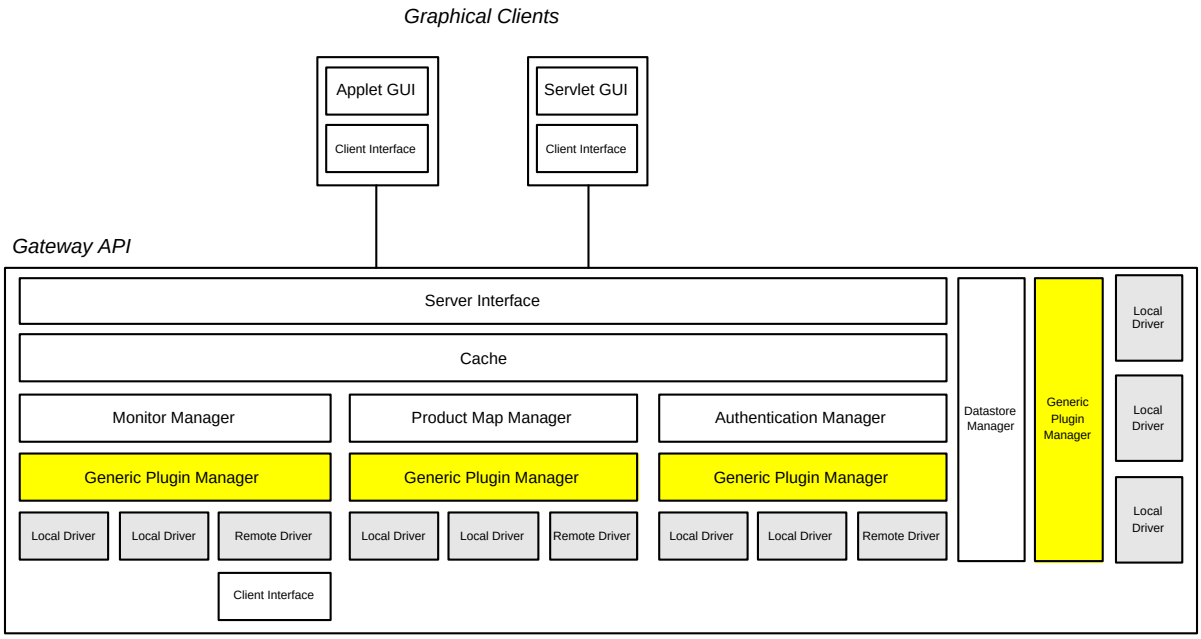

Figure 4.2: Final acrhitecture

Chapter 5

Conclusions

5.1 Summary

The project set out to create middleware using existing monitoring technologies to gather resource data and then Grid concepts to disseminate the information in a controlled and secure manner. To develop this system and prove its validity a fully functional prototype was constructed using a rolling development methodology. The iterative design and implementation process allowed ideas to be explored which helped to increase the quality of the final artefact.

5.1.1 The Artefact

The highlights of the artefact are:

- An elegant data flow through the program, making for simple interactions through the various API layers,
- A universal data store, a useful and powerful component,
- Dynamic code loading, that will be reusable outside of this project,
- Security, via HTTPS support and access control.

The weaknesses of the implementation are:

- Optimisation of the source code,

- No installation script - you just unpack the source code and compile it, it would be nice if an automatic system did this for the user.

5.1.2 The System Architecture

It is hard for the developer to critique the system architecture, it compliments the existing Grid infrastructure, from an abstract view it is not very original; it is based on standard Grid concepts. The implementation view of the API is supported by the artefact, which proves that the design is sound.

5.2 Meeting the Project Objectives

The developers' background is mainly technical rather than theoretical, playing on programming strengths allowed the prototype implementation to follow the design goals relying on innovative code to overcome any problems. There were issues with the software toolkits, such as Tomcat, that were solved by using Internet search engines to find reports from other developers who had the same problem. For example, how to set up a servlet.

Some implementation issues could not be resolved by searching the Internet, as there was no previous solution to the problem. The best example of this is the problems encountered with the servlet implementation of SSL, many references were found to the problem but no one had published a solution. The developer has posted his findings to the relevant Internet discussion boards so that others can critique the work and hopefully it will be of use to programmers who have the same problem in the future.

Writing the report in time is obviously a major objective, the author managed to keep the work on schedule by leaving as much time to do the write up as possible. This applied a lot of time pressure during the development of the prototype which meant long working hours.

5.3 Possible Future Work

- The project only implemented two monitor drivers: SNMP and Procmon. In order to further prove quality of the driver loading and abstract monitor more drivers should be written.

- Username and password authentication could be replaced with more advanced security based on the immerging Grid standards.
- A historical data client could be added. It would likely take the form of another servlet interfacing with the Grid monitors and recording data for future use. It would not form part of the actual Grid monitor it self and would use the standard client interface to interact with the system.
- Change the transportation of data to use R-GMA: future versions of R-GMA may be easier to work with. It might be possible to extract the required functionality from the R-GMA packages simplifying use of the system. Using R-GMA would provide a Grid standard transport layer and remove some of the more complex remote implementation code from the abstract Grid monitor.
- If R-GMA is not used the interface to the server could be altered to be compliant with the R-GMA specification, or another Grid transport layer. This would likely evolve adding XML support to the gateway servlet, as this is becoming the standard for information passing.

5.4 Project Reflections

The implementation of a large, relatively complicated prototype rekindled the developer's love of programming. Distributed system programming is a fascinating area of study and the youthful nature of the Grid project means there is a lot of work still to be done for the first time.

Given a chance to change the way the project was managed, the main alteration would be to start work on the prototype earlier to maximise the number and quality of features included. A more traditional design and implementation process, such as the well-known Waterfall method, would have made the writing of the report simpler. As it is, the author hopes that the slightly non-standard, but honest, approach of writing about the actual process rather than trying to make the project fit a formal model is more interesting to read.

The Procmon monitor was installed to monitor the developers home LAN, although writing the review chapter increased awareness of the strengths of over monitoring systems and it is proposed to move to using Ganglia.

5.5 Final Summary

Milestones within the project are extremely important to keeping work on track. Keeping up with them requires either very good personal organisation skills, or a high level of motivation.

The area of study was extremely interesting to the author who hopes to peruse the topic further in the future. There is a lot of work to be done in the area of distributed monitoring.

Throughout the development the use of standard Java and object orientated programming concepts saved time and effort. It is important to note, however, that OOP can over complicate some things. It is often the case of the right tool for the right job, static procedural code should not be overlooked just because the development language is Java and object programming is the norm.

Bibliography

[1] Understanding PKI: Concepts, Standards, and Deployment Considerations - Carlisle Adams and Steve LLoyd - Sams - 1 November, 2002

[2] The SSL Protocol Version 3.0 specification - Netscape Communications - November 18, 1996 - ftp://ftp.netscape.com/pub/review/ssl-spec.tar.Z

[3] Internet X.509 Public Key Infrastructure (rfc2585) - R. Housley and P. Hoffman (Network Working Group) - May 1999 - http://www.ietf.org/rfc/rfc2585.txt

[4] Domain Name System Structure and Delegation (rfc1591) - J. Postel (Network Working Group) - March 1994 - http://www.ietf.org/rfc/rfc2585.txt

[5] Middleware, in A White paper on Cluster Computing, International Journal of High Performance Computing - M.A. Baker and A. Apon - December 2000

[6] The Essence of Distributed Systems - Joel M.Chrichlow - Parson Education - 2000

[7] Introduction to the Analysis of the Data Encryption Standard - Wayne G. Barker - Aegean Park Pr - 1 October, 1991

[8] Open Distributed Systems - Jon Crowcroft - UCL Press - 1996

[9] Insecure apps threaten firms - Madeline Bennett, December 2002 - IT Week

[10] Java Language Reference - Mark Grand - O'Reilly - 1997

[11] Apache Tomcat Project - The Apache Software Foundation - http://jakarta.apache.org/tomcat/ [visited March 2003]

[12] The GNU Project - The Free Software Foundation - http://www.gnu.org/ [visited April 2003]

[13] GNU Autoconf. Automake, and Libtool - Gary V. Vsughan, Ben Elliston, Tom Tromey and Ian Lance Taylor - New Riders - 2000

[14] Common Object Request Broker Architecture - OMG (Object Management Group) - http://www.corba.org [visited April 2003]

[15] Micrososft .NET - The Microsoft Corporation - http://www.microsoft.com/net/ [visited April 2003]

[16] Product Information for Visual C# .NET 2003 - The Microsoft Corporation - http://msdn.microsoft.com/vcsharp/productinfo/ [visited April 2003]

[17] .NET Product Framework - The Microsoft Corporation - http://msdn.microsoft.com/netframework/productinfo/ [visited April 2003]

[18] The Open Source Definition - Bruce Perens (Open Source Initiative) - http://www.opensource.org/docs/definition.php [Visited April 2003] - 1997

[19] Globus Heart Beat Monitor - The Globus Project - http://www-fp.globus.org/hbm/ [visited April 2003]

[20] Grid Secuirity Infrastructure - GSI Working Group - http://www.gridforum.org/2_SEC/GSI.htm [Visited April 2003]

[21] Globus Heart Beat Monitor Specification - The Globus Project - http://www-fp.globus.org/hbm/heartbeat_spec.html [visited April 2003]

[22] The NetLogger Methodology for High Performance Distributed Systems Performance Analysis - Brian Tierney, William Johnston, Brian Crowley, Gary Hoo, Chris Brooks, Dan Gunter (Lawrence Berkeley National Laboratory) - http://www-didc.lbl.gov/NetLogger/NetLogger.HPDC.paper.ieee.pdf

[23] Netlogger Tool Kit - Lawrence Berkeley National Laboratory - http://www-didc.lbl.gov/NetLogger/summary.html - [visited April 2003]

[24] UC Berkeley Millennium Project - University Of California - http://www.millennium.berkeley.edu/ [visited April 2003]

[25] The Ganglia Project - Matt Massie - http://ganglia.sourceforge.net/ [visited May 2003]

[26] Relational Grid Monitoring Architecture - WP3 (The DataGrid Project) - http://hepunx.rl.ac.uk/edg/wp3/ [visited April 2003]

[27] The DataGrid Project - The DataGrid Working Group - http://eu-datagrid.web.cern.ch/eu-datagrid/ [visited April 2003]

[28] The RPM Package Manager (RPM) - R P Herrold fbo (RPM community) - http://www.rpm.org/ [visited April 2003]

[29] Apache Ant - The Apache Software Foundation - http://ant.apache.org/ [visited April 2003]

[30] Grid Monitoring Prototype - Distributed Systems Group, Portsmouth University - http://www.dsg.port.ac.uk/ [visited November 2002]

[31] JDBC Technology - Sun Microsystems - http://java.sun.com/products/jdbc/ [visited April 2003]

[32] MySQL: The world's most popular open source database - MySQL AB - http://www.mysql.com/ [visited April 2003]

[33] Microsoft SQL Server - The Microsoft Corporation - http://www.microsoft.com/sql/ [visited April 2003]

[34] Oracle 9i Database - Oracle Corporation - http://www.oracle.com/ip/deploy/database/oracle9i/ [visited April 2003]

[35] PostgreSQL Database - PostgreSQL Inc - http://www.postgresql.org/ - [visited April 2003]

[36] A Simple Network Management Protocol (SNMP) (rfc1157) - J. Case, M. Fedor, M. Schoffstall, J. Davin (Network Working Group) - http://www.ietf.org/rfc/rfc1157.txt - May 1990

[37] SNMP FAQ - Various - news://comp.protocols.snmp/ - 11 Jul 1999

[38] Hypertext Transfer Protocol – HTTP/1.1 (rfc2616) - Network Working Group - http://www.ietf.org/rfc/rfc2616.txt - June 1999

[39] Frequently Asked Questions - Java Security - Sun Microsystems - http://java.sun.com/sfaq/ [visited April 2003]

[40] Database Systems - Thomas Connolly, Carolyn Begg and Anne Strachan - Addison Wesley - 1999

Appendix A

Product Map Database

A.1 SQL Definitions for the Product Map Database

A.1.1 Table structure for table ‘products‘

```
CREATE TABLE products (
  id int(11) NOT NULL auto_increment,
  name text NOT NULL,
  description text NOT NULL,
  PRIMARY KEY  (id),
  UNIQUE KEY id (id),
  KEY id_2 (id)
) TYPE=MyISAM;
```

A.1.2 Table Structure for Table ‘product_mappings‘

```
CREATE TABLE product_mappings (
  id int(11) NOT NULL auto_increment,
  agent text NOT NULL,
  productid int(11) NOT NULL default '0',
  oid text NOT NULL,
  PRIMARY KEY  (id),
  KEY id (id)
) TYPE=MyISAM;
```

A.2 Example Views of the Database

A.2.1 Data View from Table 'products'

id	name	description
1	infoDescription	Name of the system
2	infoMemTotal	Total memory in the machine
3	testNull	A product that is never available
4	infoUptime	Human readable uptime of machine
5	infoCpu	Name and speed of installed processor
6	dataNumberProcesses	Number of processes running
7	dataNumberUsers	Number of users logged in
8	dataCpuUsage	The percentage of cpu time being used
9	dataEth0Received	Number of kb received from eth0

A.2.2 Data View Table 'product_mappings'

id	agent	productid	oid
1	procmon	1	/proc/version
2	procmon	2	memtotal.sh
3	snmp	1	1.3.6.1.2.1.1.1.0
4	snmp	2	1.3.6.1.2.1.25.2.2.0
5	snmp	3	4.5.6.7.8.9.9
6	procmon	4	uptime.sh
7	procmon	5	cpuinfo.sh
8	snmp	6	1.3.6.1.2.1.25.1.6.0
9	snmp	7	1.3.6.1.2.1.25.1.5.0
10	procmon	8	data/cpuusage.dat
11	procmon	9	data/netusage_eth0.dat

Appendix B

Universal Data Store Database

B.1 SQL Definitions for the Universal Data Store Database

B.1.1 Table Structure for Table ‘udd_index‘

```
CREATE TABLE rgma_index (
  id bigint(20) NOT NULL auto_increment,
  name text NOT NULL,
  leasetime int(11) NOT NULL default ’0’,
  PRIMARY KEY  (id),
  UNIQUE KEY id (id),
  KEY id_2 (id)
) TYPE=MyISAM;
```

B.1.2 Table structure for table ‘udd_data‘

```
CREATE TABLE rgma_data (
  id bigint(20) NOT NULL auto_increment,
  lexpire bigint(14) default NULL,
  indexlink bigint(20) NOT NULL default ’0’,
  datalink bigint(20) NOT NULL default ’0’,
  data text NOT NULL,
  PRIMARY KEY  (id)
) TYPE=MyISAM;
```

B.2 Example Views of the Database

B.2.1 Data View from Table 'udd_index'

id	name
1	siteName
2	siteServletUrl
3	nodeName
4	nodeIp
5	localSiteName
6	productName
7	userName
8	userPassword

B.2.2 Data View Table 'udd_data'

id	lexpire	indexlink	datalink	data
1	0	1	0	UniLan
2	0	1	0	ParentsLan
3	0	1	0	DSG
4	0	2	1	https://unipingu.no-ip.com:8080/gridmo..
5	0	2	2	https://217.39.8.114:8080/gridmonitor..
6	0	2	3	https://marge.dsg.port.ac.uk:8089/gri..
7	0	5	0	UniLan
775	1048676337137	6	770	dataCpuUsage
772	1048676337493	6	770	infoMemTotal
773	1048676337353	6	770	infoUptime
774	1048676337222	6	770	infoCpu
771	1048676337600	6	770	infoDescription
770	1048676321789	4	2	192.168.0.36
768	1048676321724	4	2	192.168.0.102
769	1048557860396	6	768	infoDescription
766	1051846246424	6	752	infoDescription

```
|765|1051846247006|       6 |    753 |dataNumberUsers                          |
|764|1051846246872|       6 |    753 |dataNumberProcesses                      |
|763|1048676288951|       6 |    753 |dataEth0Received                         |
|762|1048676288832|       6 |    753 |dataCpuUsage                             |
|761|1048676288701|       6 |    753 |infoCpu                                  |
|760|1048676288624|       6 |    753 |infoUptime                               |
|759|1051846246682|       6 |    753 |infoMemTotal                             |
|758|1051846246471|       6 |    753 |infoDescription                          |
|757|1051846247156|       6 |    751 |dataNumberUsers                          |
|756|1051846247003|       6 |    751 |dataNumberProcesses                      |
|755|1051846246792|       6 |    751 |infoMemTotal                             |
|754|1051846246346|       6 |    751 |infoDescription                          |
|753|1051846255084|       4 |      1 |192.168.0.100                            |
|752|1051846254436|       4 |      1 |192.168.0.103                            |
|751|1051846255173|       4 |      1 |192.168.0.32                             |
+---+-------------+---------+--------+-----------------------------------------+
```

Appendix C

External Java Packages

C.1 External Java Packages

Several Java packages were used during the development of the artefact. The following table describes their functionality an in what part of the software they were accessed from.

Name	Author	License	Description
com.oreilly.servlet	Jason Hunter	Free non commercial use	Various HTTP classes
jakarta-tomcat-4.0.6	Apache Foundation	Open Source	Java Servlet Engine
jcifs-0.7.3	Michael B. Allen	Open Source	Samba (windows file and prin
mm.mysql-2.0.6.1	Mark Matthews	Open Source	Mysql JDBC driver
snmp-1.1	Jonathan Sevy	Open Source	Low level SNMP communicat

Table C.1: External Java Packages

Appendix D

Source Code

D.1 gridclients.awt : AwtClient

```
import org.grid.client.awt.MonitorInterface;
public class AwtClient{
public static void main (String[] args){

    MonitorInterface thisClient = new MonitorInterface("https://mrflibble._
      uni.lan:8080");
    thisClient.show();
  }
}
```

D.2 gridclients.servlet : WebInterface

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.grid.client.MonitorConnection;
public class WebInterface extends HttpServlet {
  public void init(){
    System.setProperty("javax.net.ssl.keyStore","/usr/java/tomcat/.keystore");
    System.setProperty("javax.net.ssl.keyStorePassword","redbull");
    System.setProperty("java.protocol.handler.pkgs","javax.net.ssl");
```

```
    System.out.println("Fiddled with cert system properties");
  }
  public void doGet(HttpServletRequest req, HttpServletResponse res) throws _
    ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    HttpSession session = req.getSession(true);
    String userName = (String) session.getValue("userName");
    String userPassword = (String) session.getValue("userPassword");
    printHead(out);
    String siteName = req.getParameter("s");
    String nodeName = req.getParameter("n");
    if(userPassword==null){
      String helpUser = "";
      if(userName!=null){
         out.println("<font color=\"red\">Authentication failure (check user _
           and password)</font><br>");
         helpUser = userName;
      }
      out.println("<br><form action=\"./WebInterface\" method=\"POST\">");
      out.println("<table>");
      out.println("<tr><td>User Name:</td><td><input type=\"text\" _
        name=\"un\" size=\"20\" value=\"" + helpUser + "\"> (Example _
          <i>dave@BigComputingSite</i>)</td></tr>\n");
      out.println("<tr><td>Password:</td><td><input type=\"password\" name=\"_
        up\" size=\"20\"></td></tr>\n");
      out.println("</table>");
      out.println("<input type=\"submit\" name=\"l\" value=\"Log On\">");
      out.println("</form>");
    }
    else{
      MonitorConnection monitorServlet;
      try {
        monitorServlet = new MonitorConnection("https://localhost:8080/_
          gridmonitor/Monitor", userName, userPassword);
      }
      catch(java.net.ConnectException e){
        out.println("Can't connect to Monitor Servlet");
        return;
```

```
}
if(siteName==null){
  // display the site list
  Vector sites = monitorServlet.getSiteList();
  for(int pnt = 0; pnt < sites.size(); pnt++){
    String site = (String) sites.get(pnt);
    out.println("<a href=\"./WebInterface?s=" + site + "\">"); _
      <br><br>");
    out.println(site + "<br>");
    out.println("</a><br>");
  }
}
else{
  if(nodeName==null){
    // display the node list for this site
    out.println("<b>" + siteName + " : : </b>Node List<br><br>");
    Vector nodes = monitorServlet.getNodeList(siteName);
    out.println("<table width=\"100%\">");
    int cols = 0;
    for(int cnt = 0; cnt < nodes.size(); cnt++){
      String node = (String) nodes.get(cnt);
      cols++;
      if(cols==1){
        out.println("<tr>");
      }
      out.println("<td><a href=\"./WebInterface?s=" + siteName + "&n="_
        + node +  "\">");
      out.println("<img src=\"/gridinterface/icons/node.png\" border=0>_
        <br>");
      out.println(node + "<br>");
      out.println("</a></td>");
      if(cols==4){
        out.println("</tr>");
        cols = 0;
      }
    }
    out.println("</table>");
  }
  else{
```

```
        // display the products and their vaules for this node
        out.println("<b>" + siteName + " : " + nodeName + " : </b>Product_
          List<br><br>");
        Vector products = monitorServlet.getProductList(siteName,nodeName);
        out.println("<table width=\"100%\">");
        int row = 0;
        for(int pcnt = 0; pcnt < products.size(); pcnt++){
          row++;
          if(row==2){
            out.println("<tr><td>");
            row = 0;
          }
          else{
            out.println("<tr bgcolor=\"#EDEDED\"><td>");
          }
          String product = (String) products.get(pcnt);
          String productValue = monitorServlet.getProductValue(siteName,_
            nodeName,product);
          out.println(product + "</td><td><i>" + productValue + "</i>");
          out.println("</td></tr>");
        }
        out.println("</table>");
      }
    }
  }
  printFoot(out);
}
private void printNodes(PrintWriter out, String siteName){
}
private void printHead(PrintWriter out){
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Distributed Grid Resource Monitor</title>");
  out.println("<link rel=\"stylesheet\" href=\"/gridinterface/main.css\" _
    type=\"text/css\">");
  out.println("</head>");
  out.println("<body bgcolor=\"dde1dd\" text=\"#000000\">");
  out.println("<img border=\"0\" width=\"791\" height=\"96\" src=\"/_
    gridinterface/header.png\"><br>");
```

```
    out.println("<table width=\"791\" height=\"600\" border=\"1\" _
      cellpadding=\"0\" cellspacing=\"0\" bgcolor=\"white\" bordecolor=_
        \"black\"><tr><td>");
    out.println("<table width=\"100%\" height=\"600\"><tr><td  bgcolor=\_
      "#eeeeee\" width=\"134\" align=\"left\" valign=\"top\">");
    out.println("<table width=\"132\" bgcolor=\"#333333\" cellpadding=\"1\"_
       cellspacing=\"0\" border=\"0\"><tr><td><table width=\"132\" _
         border=\"0\" cellpadding=\"0\" cellspacing=\"0\" bgcolor=_
           \"#678fe3\"><tr><td width=\"132\" align=\"center\" valign=_
             \"top\">");
    out.println("<a href=\"/gridinterface/WebInterface\"><font color=\"_
      white\">Html Interface</font></a>");
    out.println("</td></tr></table></td></tr></table>");
    out.println("<table height=\"2\"><tr><td></td></tr></table>");
    out.println("<table width=\"132\" bgcolor=\"#333333\" cellpadding=\"1\"_
      cellspacing=\"0\" border=\"0\"><tr><td><table width=\"132\" _
        \"#678fe3\"><tr><td width=\"132\" align=\"center\" valign=\"top\">");
    out.println("<a href=\"/gridinterface/applet/index.html\"><font _
      color=\"white\">Applet Interface</font></a>");
    out.println("</td></tr></table></td></tr></table>");
    out.println("<table height=\"2\"><tr><td></td></tr></table>");
    out.println("<table width=\"132\" bgcolor=\"#333333\" cellpadding=\"1\"_
      cellspacing=\"0\" border=\"0\"><tr><td><table width=\"132\" __
        #678fe3\"><tr><td width=\"132\" align=\"center\" valign=\"top\">");
    out.println("<a href=\"http://homer.dsg.port.ac.uk/~matgrove/index.php_
      \"><font color=\"white\">Project Homepage</font></a>");
    out.println("</td></tr></table></td></tr></table>");
    out.println("</td><td align=\"left\" valign=\"top\">");
  }
  private void printFoot(PrintWriter out){
    out.println("</td></tr></table>");
    out.println("</td></tr></table>");
    out.println("<a href=\"mailto:me@matthewgrove.co.uk\">Mat Grove</a>_
      /font>");
    out.println("</body>");
    out.println("</html>");
  }
  public void doPost(HttpServletRequest req, HttpServletResponse res) _
    throws ServletException, IOException {
```

```
    // we only use post for the log on
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String userName = req.getParameter("un");
    String userPassword = req.getParameter("up");
    MonitorConnection monitorServlet;
    try {
      monitorServlet = new MonitorConnection("https://localhost:8080/_
        gridmonitor/Monitor", userName, userPassword);
    }
    catch(java.net.ConnectException e){
      out.println("Can't connect to Monitor Servlet");
      return;
    }
    boolean authOk = monitorServlet.getAuthenticated();
    HttpSession session = req.getSession(true);
    if(!authOk){
      if(session.getValue("userName")!=null){
        session.removeValue("userName");
      }
      session.putValue("userName", userName);
      doGet(req, res);
    }
    else{
      session.putValue("userName", userName);
      session.putValue("userPassword", userPassword);
      doGet(req, res);
    }
  }
}
```

D.3 gridclients.applet : AwtAppletClient

```
import org.grid.client.awt.MonitorInterface;
import java.applet.*;
public class AwtAppletClient extends Applet {
public void init(){
    String serverBase;
```

```
    serverBase = "http://" + getCodeBase().getHost()+ ":" + getCodeBase()._
      getPort();
    MonitorInterface thisClient = new MonitorInterface(serverBase);
    thisClient.show();
  }
}
```

D.4 gridclients.cli : Cli

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
//import javax.servlet.http.*;
//import org.grid.client.MonitorConnection;
import org.grid.mds.MdsGridSecurity;
public class Cli{
  public static void main (String[] args){
    System.out.println("Testing security");
    String u = "mat@UniLan";
    String p = "b";
    boolean b = MdsGridSecurity.authenticate("UniLan",u,p);
    String s = "stuff\n";
  }
}
/**
public static void main (String[] args){
    MonitorConnection monitorServlet;
    try {
      monitorServlet = new MonitorConnection("http://localhost:8080/_
        gridmonitor/Monitor");
    }
    catch(java.net.ConnectException e){
      out("Can't connect to Monitor Servlet",0);
      return;
    }
    Vector sites = monitorServlet.getSiteList();
    for(int pnt = 0; pnt < sites.size(); pnt++){
      String site = (String) sites.get(pnt);
```

```
      out(site,0);
        Vector nodes = monitorServlet.getNodeList(site);
        for(int cnt = 0; cnt < nodes.size(); cnt++){
          String node = (String) nodes.get(cnt);
          out(node,2);
          //if(site.equals("UniLan")){
            Vector products = monitorServlet.getProductList(site,node);
            for(int pcnt = 0; pcnt < products.size(); pcnt++){
              String product = (String) products.get(pcnt);
              String productValue = "";
              //if(req.getParameter("withproducts")!=null){
              //  productValue = monitorServlet.getProductValue(site,node,_
                product);
              //}
              out(product,4);
            }
          //}
        }
    }
  }
  private static void out(String v, int indent){
    String it = "";
    for(int i=1;i<=indent;i++){
      it = it + " ";
    }
    System.out.println(it + " : " + v);
  }
}
**/
```

D.5 gridmonitor : MonitorServlet

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.grid.mds.MdsGridData;
import org.grid.monitor.*;
```

```
import org.grid.security.SecurityManager;
public class MonitorServlet extends HttpServlet {
  private String localSiteName;
  public void init(){
    localSiteName = MdsGridData.getLocalSiteName();
    new MonitorScan("192.168.0",localSiteName);
  }
  public void doGet(HttpServletRequest req, HttpServletResponse res) throws Serv
    // EVERY QUERY needs a username and password, all users must authenticate be
    // start to talk to this site - regardless of what they want to do here
    String userName = req.getParameter("userName");
    String userPassword = req.getParameter("userPassword");
    boolean authOk = SecurityManager.authenticate(localSiteName, userName, _
      userPassword);
    if(!authOk){
      PrintWriter out = res.getWriter();
      out.println("Error: 403");
      return;
    }
    if(req.getParameter("getLocalSiteName")!=null){
      res.getWriter().println(localSiteName);
      return;
    }
    if(req.getParameter("getNodeList")!=null){
      String siteName=req.getParameter("siteName");
      printVector(res.getWriter(), MdsGridData.getNodeList(userName, _
        userPassword, localSiteName,siteName));
      return;
    }
    if(req.getParameter("getSiteList")!=null){
      printVector(res.getWriter(), MdsGridData.getSiteList());
      return;
    }
    if(req.getParameter("getProductList")!=null){
      String siteName=req.getParameter("siteName");
      String nodeName=req.getParameter("nodeName");
      printVector(res.getWriter(), MdsGridData.getProductList(userName, _
        userPassword, localSiteName,siteName,nodeName));
      return;
```

```
    }
    if(req.getParameter("getProduct")!=null){
      String siteName=req.getParameter("siteName");
      String nodeName=req.getParameter("nodeName");
      String productName=req.getParameter("productName");
      res.getWriter().println(MdsGridData.getProduct(userName, userPassword,_
        localSiteName, siteName,nodeName,productName));
      return;
    }
    PrintWriter out = res.getWriter();
    out.println("Fell through gets");
  }
  private void printVector(PrintWriter out, Vector stringData){
    for(int cnt = 0; cnt < stringData.size(); cnt++){
      out.println(stringData.get(cnt));
    }
  }
  public void doPost(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {
    doGet(req, res);
  }
}
```

D.6 org.grid.support : Debug

```
package org.grid.support;
import java.io.*;
public class Debug{
  static int debugOn = 1;
  static boolean hideSQL = true;
public static void out(String msg){
if(debugOn == 1){
     System.out.println("Debug: " + msg);
}
  }
public static void log(boolean local, String type, String msg){
  if(type.equals("SQL") && hideSQL){
    return;
```

```
  }


    String logOut = "Log: ";
    if(local){
      logOut = logOut + "local,";
    }
    else{
      logOut = logOut + "remote,";
    }
    logOut = logOut + type + "," + msg;
    System.out.println(logOut);
  }
}
```

D.7 org.grid.support : StringPair

```
package org.grid.support;
import java.io.*;
public class StringPair{
  public String name;
  public String value;
  public StringPair(String s1, String s2){
    name=s1;
    value=s2;
  }
  public StringPair(String s1){
    name=s1;
    value="";
  }
  public StringPair(){
    name="";
    value="";
  }
}
```

D.8 org.grid.support : IndexPair

```
package org.grid.support;
import java.io.*;
public class IndexPair{
  public int id;
  public String value;
  public IndexPair(int i, String s2){
    id=i;
    value=s2;
  }
  public IndexPair(){
    id=0;
    value="";
  }
}
```

D.9 org.grid.mds.rgmb : GridSecurity

```
package org.grid.mds.rgmb;
import org.grid.mds.GridSecurityModule;
import java.util.*;
import org.grid.support.*;
public class GridSecurity implements GridSecurityModule{
  private static Stack addPairToStack(Stack theStack, String paramOne, String pa
    String params[];
    params = new String[2];
    params[0]=paramOne;
    params[1]=paramTwo;
    theStack.push(params);
    return theStack;
  }
  public boolean authenticate(String siteName, String userName, String _
    userPassword){
    Debug.out("RGMB security module is going to try to authenticate _
      someone");
    Debug.out("Sitename: " + siteName);
    Debug.out("Username: " + userName);
```

```
    Debug.out("Sitename: " + siteName);
    Stack dataStack = new Stack();
    dataStack = addPairToStack(dataStack, "userPassword", "");
    dataStack = addPairToStack(dataStack, "userName", userName);
    dataStack = addPairToStack(dataStack, "siteName", siteName);
    Vector retrievedPasswordList;
    String retrievedPassword = "";
    Vector leaseWithData = RgmbQuery.recursiveGet(dataStack,"");
    if(!leaseWithData.isEmpty()){
      String values[] = (String[]) leaseWithData.get(0);
      retrievedPassword = values[1];
    }
    if(userPassword.equals(retrievedPassword)){
      Debug.out("rgmb security module thinks password is valid");
      return true;
    }
    else{
      Debug.out("rgmb security module thinks password is wrong");
      return false;
    }
  }
}
```

D.10 org.grid.mds.rgmb : RgmbQuery

```
package org.grid.mds.rgmb;
import java.util.*;
import java.net.*;
import java.io.*;
import java.sql.*;
import org.grid.support.*;
public class RgmbQuery{
  // create a lease expire time given the name of the data
  private static String generateLease(String dataName){
    // for now we blindly set lease times for 10 minutes
    long leaseMinutes = 10;
    long lease = System.currentTimeMillis() + (leaseMinutes * 1000 * 60);
    String leaseExpires = String.valueOf(lease);
```

```
    return leaseExpires;
  }
  public static Vector recursiveGet(Stack theStack, String lastDataLinkId){
    String values[] = (String[]) theStack.pop();
    //System.out.println("Val1: '" + values[0] + "' Val2: '" + values[1] + _
      "' link id: '" + lastDataLinkId + "'");
    if(!theStack.empty()){
      Vector keyAndLease = RgmbQuery.getKey(values[0], values[1], _
        lastDataLinkId);
      if(!keyAndLease.isEmpty()){
        lastDataLinkId = (String) keyAndLease.get(0);
        return recursiveGet(theStack,lastDataLinkId);
      }
      else{
        // wasnt found
        Debug.out("recursiveGet Couldn't get a key");
        return new Vector();
      }
    }
    else{
      return RgmbQuery.getValues(values[0],lastDataLinkId);
    }
  }
  public static void recursiveSet(Stack theStack, String lastDataLinkId, _
    String dataValue){
    String values[] = (String[]) theStack.pop();
    //System.out.println("Val1: '" + values[0] + "' Val2: '" + values[1] + _
      "' link id: '" + lastDataLinkId + "'");
    if(!theStack.empty()){
      Vector keyAndLease = RgmbQuery.getKey(values[0], values[1], _
        lastDataLinkId);
      if(!keyAndLease.isEmpty()){
        lastDataLinkId = (String) keyAndLease.get(0);
        recursiveSet(theStack,lastDataLinkId, dataValue);
      }
      else{
        // wasnt found
        Debug.out("recursiveSet Couldn't get a key");
      }
```

```
    }
    else{
      Vector key = RgmbQuery.getKey(values[0], dataValue, lastDataLinkId);
      if(key.isEmpty()){
        // the node has never been cached so we need to create it
        RgmbQuery.createValue(values[0], dataValue, lastDataLinkId);
      }
      else{
        //Debug.out("Got the ID of: " + nodeName + " in site: " + siteName _
          + " and it is: " + (String) key.get(0));
        String parentId = (String) key.get(0);
        // the value already exists so we do an update
        RgmbQuery.setValue(values[0], dataValue, parentId);
      }
      //return RgmbQuery.setValue(values[0],dataValue,lastDataLinkId);
    }
  }
  // update a record when we know its ID
  public static void setValue(String dataName, String data, String dataId){
    String leaseExpire = generateLease(dataName);
    String SQL = "UPDATE rgma_data SET rgma_data.data='" + data + "', _
      lexpire='" + leaseExpire + "'" +
                 " WHERE rgma_data.id='" + dataId + "'";
    try{
      DQ(SQL);
    }
    catch (Exception e) {
    }
  }
  // create a new record
  public static void createValue(String dataName, String data, String _
    dataLink){
    String leaseExpire = generateLease(dataName);
    try{
      String indexLink = DQS("SELECT rgma_index.id FROM rgma_index WHERE _
        rgma_index.name='" + dataName + "'");
      String SQL = " INSERT INTO `rgma_data` (`id`, `lexpire`, `indexlink`, _
        `datalink`, `data`)" +
                   " VALUES ('', '" + leaseExpire + "', '" + indexLink + _
```

```
                    "', '" + dataLink + "', '" + data + "')";
      DQ(SQL);
    }
    catch (Exception e) {
    }
  }
  // returns the Id and the lease time of one object in the dbase
  public static Vector getKey(String dataName, String data, String parentId){
    String SQL = "select rgma_data.id, rgma_data.lexpire from rgma_data,_
      rgma_index where rgma_data.indexlink=rgma_index.id" +
                 " AND rgma_index.name='" + dataName + "' AND rgma_data._
                   data='" + data + "'";
    if(!parentId.equals("")){
      SQL = SQL + " AND rgma_data.datalink='" + parentId + "'";
    }
    try{
      return DQRS(SQL);
    }
    catch (Exception e) {
      return new Vector();
    }
  }
  // returns one value given the parent id and the dataname (type)
  public static String getValue(String dataName, String parentId){
    String SQL = "select rgma_data.data from rgma_data,rgma_index where _
      rgma_data.indexlink=rgma_index.id" +
                 " AND rgma_index.name='" + dataName + "' AND rgma_data._
                   datalink='" + parentId + "'";
    try{
      return DQS(SQL);
    }
    catch (Exception e) {
      return "";
    }
  }
  // returns a list of value given the parent id and the dataname (type)
  public static Vector getValues(String dataName, String parentId){
    String SQL = "select rgma_data.lexpire, rgma_data.data from _
      rgma_data,rgma_index where rgma_data.indexlink=rgma_index.id" +
```

```
                " AND rgma_index.name='" + dataName + "' AND rgma_data._
                  datalink='" + parentId + "'";
    try{
      return DQR(SQL);
    }
    catch (Exception e) {
      return new Vector();
    }
  }
  // load the sql driver
  private static void loadDriver(){
try{
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
    }
catch (Exception e) {
   Debug.out("org.grid.mds.rgmb.Query.setValue()");
       Debug.out("Couldn't load MYSQL driver");
       System.exit(1);
    }
  }
  // execute an sql statement and return the result set
  private static ResultSet executeSQL(String SQL) throws java.sql._
    SQLException{
    try{
      // load the sql driver
      loadDriver();
      // Create a Connection and a Statement
      Connection conn =DriverManager.getConnection("jdbc:mysql://_
        127.0.0.1:3306/project", "snmp", "java");
      //System.out.println("* New SQL is going to run: " + SQL);
      Debug.log(true, "SQL", SQL);
      Statement stmt = conn.createStatement();
      // execute the query
      ResultSet resultSet =  stmt.executeQuery (SQL);
      //System.out.println("* New SQL executed ok");
      return resultSet;
    }
catch (Exception e) {
  System.out.println("New SQL failed");
```

```
  throw new java.sql.SQLException();
    }
  }
  // execute sql and return 1 string value
  private static String DQS(String SQL) throws java.sql.SQLException{
    try{
      ResultSet resultSet = executeSQL(SQL);
      resultSet.next();
      return resultSet.getString(1);
    }
catch (Exception e) {
   throw new java.sql.SQLException();
    }
  }
  // execute sql and return a list of string values
  private static Vector DQ(String SQL) throws java.sql.SQLException{
    try{
      ResultSet resultSet = executeSQL(SQL);
      Vector data = new Vector();
      while(!resultSet.isLast()){
        resultSet.next();
        data.add(resultSet.getString(1));
      }
      return data;
    }
catch (Exception e) {
   throw new java.sql.SQLException();
    }
  }
  // execute sql and return a list of rows with 2 values each
  private static Vector DQR(String SQL) throws java.sql.SQLException{
    try{
      ResultSet resultSet = executeSQL(SQL);
      Vector data = new Vector();
      String pair[];
      while(!resultSet.isLast()){
        resultSet.next();
        pair = new String[2];
        pair[0]=resultSet.getString(1);
```

```
        pair[1]=resultSet.getString(2);
        //Debug.out("Hoiked " + pair[0] + "," + pair[1]);
        data.add(pair);
      }
      return data;
    }
catch (Exception e) {
  throw new java.sql.SQLException();
    }
  }
  // execute sql and return 1 row of values
  private static Vector DQRS(String SQL) throws java.sql.SQLException{
    try{
      ResultSet resultSet = executeSQL(SQL);
      Vector data = new Vector();
      resultSet.next();
      data.add(resultSet.getString(1));
      data.add(resultSet.getString(2));
      return data;
    }
    catch (Exception e) {
      throw new java.sql.SQLException();
    }
  }
}
```

D.11 org.grid.mds : Mds

```
package org.grid.mds;
import java.util.*;
import org.grid.mds.rgmb.RgmbQuery;
import org.grid.client.MonitorConnection;
import org.grid.support.*;
public class Mds{
  public void Mds(){
  }
  public static boolean checkLease(Stack commands){
    Vector results = RgmbQuery.recursiveGet(commands,"");
```

```
    if(!results.isEmpty()){
      String[] firstValue = (String[]) results.get(0);
      String lease = firstValue[0];
      if(!leaseExpired(lease)){
        //Debug.log(true, "lease", "hit");
        return true;
      }
      else{
        //Debug.log(true, "lease", "miss");
        return false;
      }
    }
    else{
      // there were no nodes in the cache which I suppose means the lease
      // is very toasted :)
      Debug.log(true, "lease", "no values");
      return false;
    }
  }
  private static boolean leaseExpired(String lease){
    long lExpire = Long.parseLong(lease);
    // a lease set to -1 can't expire
    if((lExpire > System.currentTimeMillis()) || lExpire < 0 ){
      // lease hasnt expired
      //Debug.out("lease hasnt expired");
      return false;
    }
    else{
      // lease has expired
      //Debug.out("lease has expired");
      return true;
    }
  }
  public static Stack addPairToStack(Stack theStack, String paramOne, _
    String paramTwo){
    String params[];
    params = new String[2];
    params[0]=paramOne;
    params[1]=paramTwo;
```

```
    theStack.push(params);
    return theStack;
  }
  private static Vector stripLeaseData(Vector leaseWithData){
    Vector stripped = new Vector();
    for(int cnt = 0; cnt < leaseWithData.size(); cnt++){
      String values[] = (String[]) leaseWithData.get(cnt);
      stripped.add(values[1]);
    }
    return stripped;
  }
  public static Vector getFromCache(Stack commands){
    //Debug.out("getFromCache");
    return stripLeaseData(RgmbQuery.recursiveGet(commands,""));
  }
  public static void setCache(Stack commands, String dataValue){
    RgmbQuery.recursiveSet(commands, "", dataValue);
  }
}
```

D.12 org.grid.mds : MdsGridSecurity

```
package org.grid.mds;
import org.grid.mds.rgmb.*;
import org.grid.mds.local.*;
import java.util.*;
import org.grid.support.*;
import java.lang.reflect.*;
public class MdsGridSecurity{
/**
  static String module = "rgmb";
  private GridSecurityModule loadAgent(String className, String _
    paramString1, String paramString2 ) throws Exception{
    try{
      // get reference to the class
      Class classRef = Class.forName( className );
      Constructor constr = classRef.getConstructor(null);
      GridSecurityModule securityModule = (GridSecurityModule) _
```

```
        constr.newInstance(null);
      return securityModule;

    }
    catch(java.lang.ClassNotFoundException e){
      Debug.out("Couldn't dynamically load class: " + className);
      throw new Exception("");
    }
    catch(Exception e){
      throw new Exception("");
    }
  }
**/
  public static boolean authenticate(String localSiteName, String _
    userNameSitePair, String userPassword){
    // usernames take the form of user@site
    // but they are stored as user in the home mds with site as the
    // parent node
    StringTokenizer st = new StringTokenizer(userNameSitePair, "@");
    int numTokes = st.countTokens();
    if(numTokes != 2){
      Debug.out("THIS IS A FUCKUP - the user@site pair is munged (look _
        for the @): " + userNameSitePair);
      return false;
    }
    String userName = st.nextToken();
    String siteName = st.nextToken();
    org.grid.mds.jcifs.GridSecurity gs = new org.grid.mds.jcifs._
      GridSecurity();
    GridSecurityModule gsm = (GridSecurityModule) gs;
    Debug.out("Username: " + userName);
    Debug.out("Sitename: " + siteName);
    Stack dataStack = new Stack();
    dataStack = Mds.addPairToStack(dataStack, "userPassword", "");
    dataStack = Mds.addPairToStack(dataStack, "userName", userName);
    dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
    Vector retrievedPasswordList;
    boolean authenticatedOk = false;
    if(Mds.checkLease((Stack)dataStack.clone())){
```

```
      Debug.out("Good lease");
      retrievedPasswordList = Mds.getFromCache(dataStack);
      if(!retrievedPasswordList.isEmpty()){
        String retrievedPassword = (String) retrievedPasswordList.get(0);
        //Debug.out("retrieved password: " + retrievedPassword);
        if(userPassword.equals(retrievedPassword)){
          Debug.out("Using cached password - password is valid");
          authenticatedOk = true;
        }
      }
    }
    else{
      if(siteName.equals(localSiteName)){
        // its a local user so use the GSM for this site
        Debug.out("Using local gsm to get password");
        authenticatedOk = gsm.authenticate(siteName, userName, userPassword);
      }
      else{
        // this user isn't one of ours, we need to do a remote password get
        // from their site, this data will be cached in the standard way
        Debug.out("Going to do a remote fetch for the password");
        //authenticatedOk = remote.authenticate(siteName, userName, _
          userPassword);
      }
      if(authenticatedOk){
        // password was ok so cache it
        cacheUserNameAndPassword(siteName, userName, userPassword);
      }
    }
    return authenticatedOk;
  }
  private static void cacheUserNameAndPassword(String siteName, String _
    userName, String userPassword){
    Stack dataStack = new Stack();
    dataStack = Mds.addPairToStack(dataStack, "userName", "");
    dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
    Mds.setCache((Stack)dataStack, userName);
    dataStack = Mds.addPairToStack(dataStack, "userPassword", "");
    dataStack = Mds.addPairToStack(dataStack, "userName", userName);
```

```
    dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
    Mds.setCache((Stack)dataStack, userPassword);
  }
}
```

D.13 org.grid.mds : MdsGridData

```
package org.grid.mds;
import java.util.*;
import org.grid.mds.rgmb.RgmbQuery;
import org.grid.client.MonitorConnection;
import org.grid.support.*;
import org.grid.monitor.*;
public class MdsGridData{
  public void MdsGridData(){
  }
  public static String getServletUrl(String siteName){
    Vector key = RgmbQuery.getKey("siteName", siteName, "");
    String siteId = (String) key.get(0);
    return RgmbQuery.getValue("siteServletUrl",siteId);
  }
  public static String getLocalSiteName(){
    return RgmbQuery.getValue("localSiteName", "0");
  }
  public static Vector getSiteList(){
    Stack dataStack = new Stack();
    dataStack = Mds.addPairToStack(dataStack, "siteName", "");
    return Mds.getFromCache(dataStack);
  }
  public static Vector getNodeList(String userName, String userPassword, _
    String localSiteName, String siteName){
    Stack dataStack = new Stack();
    dataStack = Mds.addPairToStack(dataStack, "nodeIp", "");
    dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
    Vector cachedNodeList = Mds.getFromCache((Stack)dataStack.clone());
    if(Mds.checkLease((Stack)dataStack.clone())){
      // lease is still good
      Debug.log(true, "cache", "lease hit for getNodeList(" + siteName + ")");
```

```
      return cachedNodeList;
    }
    else{
      Debug.log(true, "cache", "lease miss for getNodeList(" + siteName + _
        ")");
      if(localSiteName.equals(siteName)){
        // this is the local site
        new MonitorScan("192.168.0",localSiteName);
        return cachedNodeList;
      }
      else{
        // we must refetch the info as the lease is up
        // or the data isnt cached
        try {
          String siteUrl=getServletUrl(siteName);
          MonitorConnection monitorServlet = new MonitorConnection(siteUrl,_
            userName, userPassword);
         Vector nodeList = monitorServlet.getNodeList(siteName);
          Debug.log(false, "get", "getNodeList(" + siteName + ") " + siteUrl);
          setNodeList(siteName, nodeList);
          return nodeList;
        }
        catch(java.net.ConnectException e){
          System.out.println("remote mds call failed for getNodeList");
          e.printStackTrace(System.out);
        }
        return new Vector();
      }
    }
  }
  public static Vector getProductList(String userName, String userPassword,_
    String localSiteName, String siteName, String nodeIp){
    Stack dataStack = new Stack();
    dataStack = Mds.addPairToStack(dataStack, "productName", "");
    dataStack = Mds.addPairToStack(dataStack, "nodeIp", nodeIp);
    dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
    Vector productList;
    if(Mds.checkLease((Stack)dataStack.clone())){
      // lease is still good
```

```
    Debug.log(true, "cache", "lease hit for getProductList(" + siteName +_
      "," + nodeIp + ")");
    productList = Mds.getFromCache(dataStack);
  }
  else{
    Debug.log(true, "cache", "lease miss for getProductList(" + siteName _
      + "," + nodeIp + ")");
    MonitorManager currentMonitor = new MonitorManager(userName, _
      userPassword, localSiteName, siteName, nodeIp);
    productList = currentMonitor.getProductList();
  }
  return productList;
}
public static String getProduct(String userName, String userPassword, _
  String localSiteName, String siteName, String nodeIp, String _
    productName){
  MonitorManager currentMonitor = new MonitorManager(userName, _
    userPassword, localSiteName, siteName, nodeIp);
  return currentMonitor.getValue(productName);
}
public static void setNodeList(String siteName, Vector nodeList){
  Stack dataStack = new Stack();
  dataStack = Mds.addPairToStack(dataStack, "nodeIp", "");
  dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
  for(int cnt = 0; cnt < nodeList.size(); cnt++){
    String nodeIp = (String) nodeList.get(cnt);
    Mds.setCache((Stack)dataStack.clone(), nodeIp);
  }
}
public static void setProductList(String siteName, String nodeIp, _
  Vector productList){
  Stack dataStack = new Stack();
  dataStack = Mds.addPairToStack(dataStack, "productName", "");
  dataStack = Mds.addPairToStack(dataStack, "nodeIp", nodeIp);
  dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
  for(int cnt = 0; cnt < productList.size(); cnt++){
    String productName = (String) productList.get(cnt);
    Mds.setCache((Stack)dataStack.clone(), productName);
  }
```

```
  }
}
```

D.14 org.grid.monitor : MonitorScan

```
package org.grid.monitor;
import org.grid.mds.MdsGridData;
import java.util.Vector;
import org.grid.support.*;
public class MonitorScan implements Runnable{
  private String localSiteName;
  private Vector nodeList = new Vector();
  private Vector scanList;
  public MonitorScan(String subnet,String siteName){
    scanList = new Vector();
    for(int cnt=1; cnt<255; cnt++){
      String host=subnet + "." + String.valueOf(cnt);
      scanList.add(host);
    }
    localSiteName = siteName;
    Thread scan = new Thread(this);
    scan.start();
  }
  public MonitorScan(Vector ipList, String siteName){
    scanList = ipList;
    localSiteName = siteName;
    Thread scan = new Thread(this);
    scan.start();
  }
  public void run() {
    if(scanList==null){
      Debug.out("Scanner was asked to scan nothing!");
      return;
    }
    for(int cnt = 0; cnt < scanList.size(); cnt++){
      String host = (String) scanList.get(cnt);
      try{
        Scanner sc = new Scanner(localSiteName,host,this);
```

```
      }
      catch(java.lang.OutOfMemoryError e){
        // maxed out the heap????
      }
    }
    MdsGridData.setNodeList(localSiteName,nodeList);
  }
  public void hostAlive(String hostIp, String hostName) {
    Debug.log(true, "monitor", hostIp + " is monitorable");
    nodeList.add(hostIp);
  }
}
```

D.15 org.grid.monitor.agent : AgentException

```
package org.grid.monitor.agent;
public class AgentException extends Exception{
  public AgentException(String message){}
}
```

D.16 org.grid.monitor : Scanner

```
package org.grid.monitor;
import java.net.*;
import java.lang.Thread;
import java.util.Vector;
public class Scanner implements Runnable{
  String hostIpString;
  MonitorScan parent;
  String localSiteName;
  public Scanner(String localSite, String hostIp,MonitorScan p) {
    hostIpString = hostIp;
    localSiteName = localSite;
    parent = p;
    Thread scanner = new Thread(this);
    scanner.start();
  }
```

```
  public void run() {
    InetAddress host = null;
    try{
      host = InetAddress.getByName(hostIpString);
    }
    catch(java.net.UnknownHostException e){
      return;
    }
    MonitorManager monitor = new MonitorManager("internal", "boogie", _
      localSiteName,localSiteName,hostIpString);
    if(monitor.getProductList().size()>0){
      String hostName = host.getHostName();
      parent.hostAlive(hostIpString,hostName);
    }
  }
}
```

D.17 org.grid.monitor : MonitorManager

```
package org.grid.monitor;
import org.grid.support.*;
import java.util.*;
import java.io.*;
import org.grid.monitor.agent.*;
import org.grid.plugins.PluginManager;
import org.grid.plugins.modules.*;
import org.grid.mds.MdsGridData;
public class MonitorManager{
  private Vector agentList;
  private Vector productList;
  private String localSiteName;
  private String monitoredSiteName = "this hasnt been set to shit yet";
  private String monitoredNodeName;
  private String userName;
  private String userPassword;
public MonitorManager(String user, String password, String localSite, _
  String site, String node){
  userName = user;
```

```
userPassword = password;
localSiteName = localSite;
monitoredSiteName = site;
monitoredNodeName = node;
productList = consolidateProductList(site,node);
}
public String getValue(String productName){
  for(int cnt = 0; cnt < productList.size(); cnt++){
    IndexPair thisProduct = (IndexPair) productList.get(cnt);
    if(thisProduct.value.equals(productName)){
      MonitorDef tmpAgent = (MonitorDef) agentList.get(thisProduct.id);
      //Debug.out("--Calling " + tmpAgent.clientAgent() +" to get " + _
        productName);
      return tmpAgent.get(productName);
    }
  }
  //Debug.out("Fell through looking for " + productName);
  return "";
}
public Vector getProductList(){
  Vector productNames = new Vector();
  for(int cnt = 0; cnt < productList.size(); cnt++){
    IndexPair thisProduct = (IndexPair) productList.get(cnt);
    productNames.add(thisProduct.value);
  }
  MdsGridData.setProductList(monitoredSiteName, monitoredNodeName, _
    productNames);
  return productNames;
}
public int getNumberProducts(){
  return productList.size();
}
private Vector consolidateProductList(String site, String host){
  agentList = new Vector();
  Vector combinedProducts = new Vector();
  Vector addedProducts = new Vector();
  String filter;
  if(site.equals(localSiteName)){
    filter = "!remote";
```

```
    }
    else{
      filter = "=remote";
    }
    String[] params = new String[4];
    params[0] = userName;
    params[1] = userPassword;
    params[2] = site;
    params[3] = host;
    agentList = PluginManager.loadPlugins("Monitor", filter, params);

    for(int cnt = 0; cnt < agentList.size(); cnt++){
      MonitorDef thisAgent = (MonitorDef) agentList.get(cnt);
      //Debug.out("vector has: " + thisAgent.clientAgent());
      Vector pi = productList(cnt, thisAgent, host);
      for(int i = 0; i < pi.size(); i++){
        IndexPair p = (IndexPair) pi.get(i);
        //Debug.out("Report: " + p.name + " has " + p.value);
        if(!addedProducts.contains(p.value)){
          addedProducts.add(p.value);
          combinedProducts.add(p);
        }
      }
    }
    return combinedProducts;
  }
  private Vector productList(int agentIndex, MonitorDef agent, String host){
    //Debug.out("capabilities for: " + agent.clientAgent());
    Vector capabilities = agent.capable();
    Vector productIndex = new Vector();
   for(int cnt = 0; cnt < capabilities.size(); cnt++){
      String product = (String) capabilities.get(cnt);
      IndexPair index = new IndexPair(agentIndex, product);
      productIndex.add(index);
    }

    return productIndex;
  }
}
```

D.18 org.grid.client : MonitorConnection

```
package org.grid.client;
import java.util.Vector;
import java.util.Properties;
import java.io.*;
import org.grid.client.ssl.PromiscuousHttpsMessage;
import java.net.URL;
import org.grid.support.Debug;
public class MonitorConnection {
  private Vector nodes;
  private String servletName;
  private URL servletUrl;
  private String userName;
  private String userPassword;
  public MonitorConnection(String s, String name, String password) _
    throws java.net.ConnectException{
    userName = name;
    userPassword = password;
    servletName = s;
    connect();
  }
  private void connect() throws java.net.ConnectException{
    try {
      servletUrl = new URL(servletName);
    }
    catch(java.net.MalformedURLException e){
      Debug.out("Can't create url for: " + servletName);
      throw new java.net.ConnectException();
    }
  }
  public Vector getSiteList(){
    return getValue("getSiteList",null,null,null);
  }
  public Vector getNodeList(String site){
    return getValue("getNodeList",site,null,null);
  }
  public Vector getProductList(String site, String node){
    return getValue("getProductList",site,node,null);
```

```
}
public String getProductValue(String site, String node, String product){
  return getSingleValue("getProduct",site,node,product);
}
private String getSingleValue(String command, String site, String node, _
  String product) {
  Vector v = getValue(command, site, node, product);
  return (String) v.get(0);
}
public boolean getAuthenticated(){
  Vector test = getValue("",null,null,null);
  if(test.size()>0){
    String value = (String) test.get(0);
    if(!value.equals("Error: 403")){
      return true;
    }
  }
  return false;
}
private Vector getValue(String command, String site, String node, String_
  product) {
  try {
    PromiscuousHttpsMessage msg = new PromiscuousHttpsMessage(servletUrl);
    Properties props = new Properties();
    props.put("userName", userName);
    props.put("userPassword", userPassword);
    props.put(command, "1");
    if(command!=null) props.put(command, "1");
    if(site!=null) props.put("siteName", site);
    if(node!=null) props.put("nodeName", node);
    if(product!=null) props.put("productName", product);
    String inputLine;
    Vector dataV = new Vector();
    BufferedReader in = new BufferedReader(new InputStreamReader(msg._
      sendGetMessage(props)));
    while ((inputLine = in.readLine()) != null){
      dataV.add(inputLine);
    }
    in.close();
```

```
      //System.out.println("Retrieved: " + msg.toString());
      return dataV;
    }
    catch (Exception e) {
      e.printStackTrace();
      System.out.println(e);
      return new Vector();
    }
  }
}
```

D.19 org.grid.client.awt : AnimGraph

```
package org.grid.client.awt;
import java.awt.*;
import org.grid.support.Debug;
public class AnimGraph extends Panel implements Runnable{
  public String pollSiteName="";
  public String pollNodeName="";
  public String pollProductName="";
  private Thread thread;
  private int coordCounter = 0;
  private int coords[];
  private int newCoord = 0;
  private int numberCoords;
  private int xGap;
  private int lastRemovedY = 0;
  private int scaleHeight;
  private final static int INSET = 40;
  private int frameHeight;
  private int frameWidth;
  private int graphHeight;
  private int graphWidth;
  private Image offScreenImage;
  private Graphics offScreenContext;
  public AnimGraph(int maxHeight, int numberOfPlots, String site, String _
    node, String product){
    pollSiteName=site;
```

```
    pollNodeName=node;
    pollProductName=product;
    scaleHeight = maxHeight;
    numberCoords = numberOfPlots;
   setVisible(true);
   setBackground(Color.white);

}
  public void run() {
   Thread me = Thread.currentThread();
   while (thread == me) {

      try {
        thread.sleep(1000);
      }
      catch (InterruptedException e) {
        break;
      }
    }
    thread = null;
  }
  public void insertData(int v){
   coordCounter=coordCounter+1;
   if(coordCounter*xGap>=graphWidth){
     coordCounter=0;
   }

    newCoord=v;
    repaint();
  }
  public void start() {
   frameWidth=getWidth();
   frameHeight=getHeight();

   graphHeight=frameHeight-(INSET*2);
   graphWidth=frameWidth-(INSET*2);

   float xGapf = (float) ((float) graphWidth) / ((float) numberCoords);
   xGap = (int) xGapf;
```

```
 printVal("Gap for x", xGap);
 printVal("Number values", numberCoords);
 printVal("Width of graph", graphWidth);

 graphWidth=numberCoords*xGap;

 coords = new int[numberCoords];

  offScreenImage = this.createImage(frameWidth,frameHeight);
  offScreenContext = offScreenImage.getGraphics();
 thread = new Thread(this);
 thread.setPriority(Thread.MIN_PRIORITY);
 thread.start();
}
public synchronized void stop() {
 thread = null;
}
private int rescale(int value){
    float f = ((float) value / (float) scaleHeight);
    f = f * ((float) graphHeight);
    f = ((float) graphHeight) - f;
    return (int) f;
}
private void printVal(String l, int v){
  System.out.print(l +": ");
  System.out.println(v);
}
public void paint(Graphics g){
  if(offScreenContext==null){
    // we are not ready to be drawn on yet
    return;
  }

 int x1;
 int y1;
 int x2;
 int y2;
```

```
int highestValue = 0;

 offScreenContext.clearRect(0,0,frameWidth,frameHeight);
 offScreenContext.setColor(new Color(245,245,245));
 offScreenContext.fillRect(INSET*2,INSET,graphWidth,graphHeight);
 offScreenContext.setColor(new Color(200,200,200));
 int yGuideLines = (int) ((float) ((float) graphHeight) / ((float) 10));
 for(int cnt = yGuideLines;cnt<=graphHeight;cnt=cnt+yGuideLines){
   offScreenContext.drawLine(INSET*2,cnt+INSET,graphWidth+(INSET*2),_
     cnt+INSET);
 }
 //
 offScreenContext.setColor(Color.blue);
 int pointer = coordCounter;
 coords[pointer]=newCoord;
 for(int cnt=numberCoords;cnt>0;cnt--){
   x1 = (INSET*2)+(cnt*xGap);
   y1 = INSET+rescale(coords[pointer]);
   if(coords[pointer]>highestValue){
     highestValue = coords[pointer];
   }
   //if(coords[pointer]>0){
   if(pointer>0){
     if(cnt==0){
       //Debug.out("This the the bit");
       x2 = x1;
       y2 = INSET+rescale(lastRemovedY);
       //y2=0;
       lastRemovedY=coords[pointer];
     }
     else{
       //Debug.out("Other");
       x2 = (INSET*2)+((cnt-1)*xGap);
       y2 = INSET+rescale(coords[pointer-1]);
     }
     offScreenContext.drawLine(x1,y1,x2,y2);
     if(pointer == coordCounter){
       printVal("x1", x1);
       printVal("y1", y1);
```

```
          printVal("x2", x2);
          printVal("y2", y2);
        }
      }
      if(pointer==0){
        pointer=numberCoords-1;
      }
      else{
        pointer--;
      }
    }
    if(highestValue>scaleHeight){
      scaleHeight = highestValue;
    }

    offScreenContext.setColor(Color.black);
    offScreenContext.drawRect(INSET*2,INSET,graphWidth,graphHeight);
   offScreenContext.drawRect(0,0,frameWidth-1,frameHeight-1);

   g.drawImage(offScreenImage,0,0,this);
   Debug.out("End paint");
  }
  public void update(Graphics g){
    // we dont update - it would redraw the whole panel
    paint(g);
  }
}
```

D.20 org.grid.client.awt : MonitorInterface

```
package org.grid.client.awt;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.Timer;
import org.grid.client.awt.GridTree;
import org.grid.client.MonitorConnection;
public class MonitorInterface extends Frame{
```

```
public Label txtValue = new Label();
private Button btnPollProduct = new Button("Plot");
private MonitorConnection monitorServlet;
private GridTree gTree;
private ScrollPane treeContainer = new ScrollPane();
private AnimGraph myGraph=null;
private Timer pollTimer;
private boolean pollStarted=false;
private String lastRetSiteName="";
private String lastRetNodeName="";
private String lastRetProductName="";
public MonitorInterface(String server){
  setBounds(0,0,780,520);
  setResizable(false);
  setLayout(null);
  this.connectMonitor(server);
  this.drawGui();
  this.addWindowListener(
    new WindowAdapter(){//anonymous class definition
      public void windowClosing(WindowEvent e){
        System.exit(0);//terminate the program
      }
    }
  );
}
public void display(String productValue, String site, String node, _
  String product){
  txtValue.setText(productValue);
  lastRetSiteName=site;
  lastRetNodeName=node;
  lastRetProductName=product;
  if(product.startsWith("data")){
    btnPollProduct.setEnabled(true);
  }
  else{
    btnPollProduct.setEnabled(false);
  }
}
private void drawGui(){
```

```
    gTree = new GridTree(monitorServlet, this);
    setLayout(null);
    setBackground(Color.lightGray);
add(btnPollProduct);
    add(txtValue);
    txtValue.setBackground(Color.white);
    btnPollProduct.setBounds(690,30,50,25);
    txtValue.setBounds(260,30,420,25);
    txtValue.setFont(new Font("Arial",0,14));
    btnPollProduct.setEnabled(false);
    add(treeContainer);
    treeContainer.setBounds(10,30,240,430);
    //treeContainer.setLayout(null);
    treeContainer.add(gTree);
    gTree.setBounds(10,30,230,420);
    btnPollProduct.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent event){
        myGraph = new AnimGraph(100,50,lastRetSiteName,lastRetNodeName,_
          lastRetProductName);
        add(myGraph);
        myGraph.setBounds(260,70,500,400);
        myGraph.start();
        pollTimer = new Timer(3000,pollProduct);
        pollTimer.start();
        pollStarted = true;
        btnPollProduct.setEnabled(false);
      }
    });
}
  ActionListener pollProduct = new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        String valueString = monitorServlet.getProductValue(myGraph._
          pollSiteName,myGraph.pollNodeName,myGraph.pollProductName);
        int valueInt = Integer.parseInt(valueString);
        myGraph.insertData(valueInt);
      }
  };
  private void connectMonitor(String server){
    try {
```

```
      monitorServlet = new MonitorConnection(server + "/gridmonitor/_
        Monitor", "mat@UniLan", "beagleboy1");
    }
    catch(java.net.ConnectException e){
      System.out.println("Can't connect to server: " + server);
      System.out.println("I can't run without a servlet to talk to");
      System.exit(0);
    }
  }
}
```

D.21 org.grid.client.awt : GridTree

```
package org.grid.client.awt;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import org.grid.client.MonitorConnection;
public class GridTree extends Canvas implements MouseListener{
  public Tree dTree = new Tree("root", null);
  public Image shapeNodeJoiner = null;
  public Image shapeBranchJoinerPlus = null;
  public Image shapeBranchJoinerMinus = null;
  public Image shapeBranchIconOpen = null;
  public Image shapeBranchIconClose = null;
  public Image shapeNodeIcon = null;
  public Image shapeContinueBar = null;
  private String workingString="Working...";
  public int shapeW;
  public int shapeH;
  public int shapeYOffset=2;
  public int textH;
  private MonitorInterface parent;
  private MonitorConnection monitorServlet;
public GridTree(MonitorConnection servletConnection, MonitorInterface p){
  parent = p;
  monitorServlet = servletConnection;
    setBackground(Color.white);
```

```
    Vector sites = monitorServlet.getSiteList();
    for(int pnt = 0; pnt < sites.size(); pnt++){
      String site = (String) sites.get(pnt);
      dTree.add(true, site);
      dTree.branch(pnt).add(false, workingString);
    }
    dTree.setOpen(true);
    addMouseListener(this);
  }
  public void mouseClicked(MouseEvent e) {
    reportMouse("Mouse clicked (# of clicks: " + e.getClickCount() + ")", e);
    Node node = dTree.clickTree(e.getX(), e.getY());
    if(node!=null){
      //System.out.println(node.getParents());
      StringTokenizer st = new StringTokenizer(node.getParents(),":");
      Vector variables = new Vector(3);
      variables.add(null);variables.add(null);variables.add(null);
      String siteName, nodeName,productName;
      int sn = 0;
      while (st.hasMoreTokens()) {
        variables.set(sn,st.nextToken());
        sn++;
      }
      siteName=(String) variables.get(0);
      nodeName=(String) variables.get(1);
      productName=(String) variables.get(2);
      System.out.println(siteName + "." + nodeName + "." + productName);
      if(node.hasChild()){
        node.toggleOpen();
        repaint();
      }
      if(productName!=null){
        parent.display(monitorServlet.getProductValue(siteName,nodeName,_
          productName),siteName, nodeName, productName);
      }
      else{
        Tree thisNode = (Tree) node;
        if(nodeName!=null ){ //&& !thisNode.child(0).getName().equals_
          (workingString)
```

```
    Vector productList = monitorServlet.getProductList(siteName,_
      nodeName);
         thisNode.clear();
         for(int cnt=0;cnt<productList.size();cnt++){
           String pn = (String)productList.get(cnt);
           thisNode.add(false,pn);
         }
       }
       else{
         if(siteName!=null){
       Vector nodeList = monitorServlet.getNodeList(siteName);
           thisNode.clear();
           for(int cnt=0;cnt<nodeList.size();cnt++){
             String nn = (String)nodeList.get(cnt);
             thisNode.add(true,nn);
             thisNode.branch(cnt).add(false, workingString);
           }
         }
       }
     }
  }
}
void reportMouse(String eventDescription, MouseEvent e) {
  System.out.println(eventDescription + " x:" + String.valueOf(e.getX()) _
    + " y: " + String.valueOf(e.getY()));
}
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
public void mousePressed(MouseEvent e) {
  reportMouse("Mouse clicked (# of clicks: " + e.getClickCount() + ")", e);
  Node node = dTree.clickTree(e.getX(), e.getY());
  if(node!=null){
    node.setColor(Color.red);
    repaint();
  }
}
public void mouseReleased(MouseEvent e) {
  reportMouse("Mouse clicked (# of clicks: " + e.getClickCount() + ")", e);
  Node node = dTree.clickTree(e.getX(), e.getY());
```

```
    if(node!=null){
      node.setColor(Color.black);
      repaint();
    }
  }
  private Image createShapeNodeJoiner(){
    Image offScreenImage = createImage(shapeW,shapeH);
    Graphics offScreenContext = offScreenImage.getGraphics();
    offScreenContext.setColor(new Color(128,128,128));
    offScreenContext.drawLine(shapeW/2,0,shapeW/2,shapeH);
    offScreenContext.drawLine(shapeW/2,(shapeH/2)+2,shapeW,(shapeH/2)+2);
    //offScreenContext.drawRect(0,0,shapeW,shapeH);
    return offScreenImage;
  }
  private Image createShapeBranchJoinerPlus(){
    Image offScreenImage = createImage(shapeW,shapeH);
    Graphics offScreenContext = offScreenImage.getGraphics();
    offScreenContext.setColor(new Color(128,128,128));
    offScreenContext.drawLine(shapeW/2,0,shapeW/2,shapeH);
    offScreenContext.drawLine(shapeW/2,(shapeH/2)+2,shapeW,(shapeH/2)+2);
    offScreenContext.setColor(Color.white);
    offScreenContext.fillRect(3,5,shapeW-6,shapeH-10);
    offScreenContext.setColor(Color.black);
    offScreenContext.drawRect(3,5,shapeW-6,shapeH-10);
    offScreenContext.drawLine((shapeW/2)-2,shapeH/2,(shapeW/2)+2,shapeH/2);
    offScreenContext.drawLine(shapeW/2,(shapeH/2)-2,shapeW/2,(shapeH/2)+2);
    //offScreenContext.drawRect(0,0,shapeW,shapeH);
    return offScreenImage;
  }
  private Image createShapeBranchJoinerMinus(){
    Image offScreenImage = createImage(shapeW,shapeH);
    Graphics offScreenContext = offScreenImage.getGraphics();
    offScreenContext.setColor(new Color(128,128,128));
    offScreenContext.drawLine(shapeW/2,0,shapeW/2,shapeH);
    offScreenContext.drawLine(shapeW/2,(shapeH/2)+2,shapeW,(shapeH/2)+2);
    offScreenContext.setColor(Color.white);
    offScreenContext.fillRect(3,5,shapeW-6,shapeH-10);
    offScreenContext.setColor(Color.black);
    offScreenContext.drawRect(3,5,shapeW-6,shapeH-10);
```

```
    offScreenContext.drawLine((shapeW/2)-2,shapeH/2,(shapeW/2)+2,shapeH/2);
    //offScreenContext.drawRect(0,0,shapeW,shapeH);
    return offScreenImage;
  }
  private Image createShapeContinueBar(){
    Image offScreenImage = createImage(shapeW,shapeH);
    Graphics offScreenContext = offScreenImage.getGraphics();
    offScreenContext.setColor(new Color(128,128,128));
    offScreenContext.drawLine(shapeW/2,0,shapeW/2,shapeH);
    //offScreenContext.drawRect(0,0,shapeW,shapeH);
    return offScreenImage;
  }
  private Image createShapeNodeIcon(){
    Image offScreenImage = createImage(shapeW,shapeH);
    Graphics offScreenContext = offScreenImage.getGraphics();
    offScreenContext.setColor(new Color(128,128,128));
    offScreenContext.drawLine(0,(shapeH/2)+2,shapeW,(shapeH/2)+2);
    //offScreenContext.drawRect(0,0,shapeW,shapeH);
    return offScreenImage;
  }
  private Image createShapeBranchIconOpen(){
    Image offScreenImage = createImage(shapeW,shapeH);
    Graphics offScreenContext = offScreenImage.getGraphics();
    offScreenContext.setColor(new Color(128,128,128));
    offScreenContext.drawLine(0,shapeH/2,shapeW,shapeH/2);
    offScreenContext.drawLine(shapeW/2,shapeH/2,shapeW/2,shapeH);
    offScreenContext.setColor(new Color(5,10,207));
    //offScreenContext.fillRect(2,4,7,11);
    //offScreenContext.fillRect(7,6,6,9);
    offScreenContext.setColor(new Color(112,115,244));
    offScreenContext.fillRect(3,5,5,9);
    offScreenContext.fillRect(3,7,10,6);
    offScreenContext.setColor(new Color(5,10,207));
    offScreenContext.fillRect(5,6,10,6);
    offScreenContext.setColor(new Color(112,115,244));
    offScreenContext.fillRect(5,10,9,5);
    offScreenContext.fillRect(7,9,5,5);
    //offScreenContext.drawRect(0,0,shapeW,shapeH);
    return offScreenImage;
```

```
  }
  private Image createShapeBranchIconClose(){
    Image offScreenImage = createImage(shapeW,shapeH);
    Graphics offScreenContext = offScreenImage.getGraphics();
    offScreenContext.setColor(new Color(128,128,128));
    offScreenContext.drawLine(0,shapeH/2,shapeW,shapeH/2);
    offScreenContext.drawLine(shapeW/2,shapeH/2,shapeW/2,shapeH);
    offScreenContext.setColor(new Color(112,115,244));
    offScreenContext.fillRect(2,4,7,11);
    offScreenContext.fillRect(7,6,6,9);
    //offScreenContext.drawRect(0,0,shapeW,shapeH);
    return offScreenImage;
  }
  public void paint(Graphics g){
    g.setColor (Color.black);
    g.setFont(new Font("Arial",0,14));
    textH = getFontMetrics (getFont ()).getHeight();
    shapeH = textH + (shapeYOffset *2);
    shapeW = 16;
    if(shapeNodeJoiner==null){
      shapeNodeJoiner = createShapeNodeJoiner();
      shapeBranchJoinerPlus = createShapeBranchJoinerPlus();
      shapeBranchJoinerMinus = createShapeBranchJoinerMinus();
      shapeBranchIconOpen = createShapeBranchIconOpen();
      shapeNodeIcon = createShapeNodeIcon();
      shapeContinueBar = createShapeContinueBar();
      shapeBranchIconClose = createShapeBranchIconClose();
    }
    int end = dTree.printTree(g,this,20,0);
  }
}
/**
    dTree.add(false, "1 Node");
    dTree.add(true, "2 Branch");
    dTree.branch(1).add(false, "1 Node");
    dTree.branch(1).add(false, "2 Node");
    dTree.branch(1).add(true, "3 Branch");
    dTree.branch(1).branch(2).add(false, "1 Node");
    dTree.branch(1).add(false, "4 Node");
```

```
    dTree.branch(1).add(false, "5 Node");
    dTree.add(false, "3 Node");
    dTree.add(false, "4 Node");
    dTree.branch(1).setOpen(true);
    dTree.branch(1).branch(2).setOpen(true);
    dTree.setOpen(true);
**/
```

D.22 org.grid.client.awt : Node

```
package org.grid.client.awt;
import java.awt.*;
public class Node{
  protected String nodeName;
  protected Node parentName;
  protected boolean visible;
  protected int yPosTop = 0;
  protected int yPosBot = 0;
  private int yCoord = 0;
  private int nodeDepth = 0;
  private Color printColor = Color.black;
public Node(String name, Node parent){
    nodeName=name;
    parentName=parent;
  }
  public boolean yIsBetween(int y){
    if((y <= yPosBot) && (y >= yPosTop)){
      return true;
    }
    else{
      return false;
    }
  }
  public String getName(){
    return nodeName;
  }
  public boolean isVisible(){
    return visible;
```

```
}
public void setOpen(boolean canBeSeen){
  visible = canBeSeen;
}
public void toggleOpen(){
  if(visible){
    visible=false;
  }
  else{
    visible=true;
  }
}
public boolean hasChild(){
  return false;
}
protected String getParents(){
  String res = "";
  if(parentName!=null){
    res = parentName.getParents();
    return res + ":" + nodeName;
  }
  else{
    return "";
  }
}
protected void setColor(Color c){
  printColor = c;
}
protected int setNodePosition(int yOffset, int textH, int depth, _
  GridTree canvas){
  nodeDepth=depth;
  yPosTop=yOffset - canvas.shapeH;
  yPosBot=yOffset;
  int totalPad = textH;
  yCoord=yOffset;
  yOffset=yOffset + canvas.shapeH;
  return yOffset;
}
protected void drawNode(Graphics g, GridTree canvas){
```

```
Color oldColor = g.getColor();
g.setColor(printColor);
String label=getName();
int relX = 0;
for(int cnt=1; cnt<nodeDepth;cnt++){
  g.drawImage(canvas.shapeContinueBar,relX ,yPosTop+canvas.shapeYOffset,_
    canvas);
  relX = relX + canvas.shapeW;
}
if(hasChild()){
  if(!visible){
    g.drawImage(canvas.shapeBranchJoinerPlus,relX,yPosTop+canvas._
      shapeYOffset, canvas);
    relX = relX + canvas.shapeW;
    g.drawImage(canvas.shapeBranchIconClose,relX ,yPosTop+canvas._
      shapeYOffset, canvas);
    relX = relX + canvas.shapeW;
  }
  else{
    g.drawImage(canvas.shapeBranchJoinerMinus,relX,yPosTop+canvas._
      shapeYOffset, canvas);
    relX = relX + canvas.shapeW;
    g.drawImage(canvas.shapeBranchIconOpen,relX ,yPosTop+canvas._
      shapeYOffset, canvas);
    relX = relX + canvas.shapeW;
  }
}
else{
  g.drawImage(canvas.shapeNodeJoiner,relX ,yPosTop+canvas.shapeYOffset,_
    canvas);
  relX = relX + canvas.shapeW;
  g.drawImage(canvas.shapeNodeIcon,relX ,yPosTop+canvas.shapeYOffset,_
    canvas);
  relX = relX + canvas.shapeW;
}
relX = relX + 2;
g.drawString(label,relX,yCoord);
//g.drawLine(0,yCoord,200,yCoord);
g.setColor(oldColor);
```

```
  }
}
```

D.23 org.grid.client.awt : Tree

```
package org.grid.client.awt;
import java.awt.*;
import java.util.Vector;
public class Tree extends Node{
  private Vector theTree = new Vector();
public Tree(String name, Node parent){
    super(name, parent);
  }
  public void add(boolean hasChild, String nodeName){
    if(hasChild){
      // this node is a branch
      theTree.add(new Tree(nodeName, this));
    }
    else{
      // this node is a value
      theTree.add(new Node(nodeName, this));
    }
  }
  public void change(int nodeNum,boolean hasChild, String nodeName){
    if(hasChild){
      // this node is a branch
      theTree.set(nodeNum,new Tree(nodeName, this));
    }
    else{
      // this node is a value
      theTree.set(nodeNum,new Node(nodeName, this));
    }
  }
  public Tree branch(int nodeNum){
    return (Tree) theTree.get(nodeNum);
  }
  public Node child(int nodeNum){
    return (Node) theTree.get(nodeNum);
```

```
}
public void clear(){
  theTree = new Vector();
}
public String select(int nodeNum){
  Node theNode = (Node) theTree.get(nodeNum);
  return theNode.getParents();
}
public String select(){
  return getParents();
}
public boolean hasChild(){
  if(theTree.size()>0){
    return true;
  }
  else{
    return false;
  }
}
public Vector getNodeList(){
  Vector nodeList= new Vector();
  for(int cnt=0; cnt<theTree.size(); cnt++){
    Node thisNode = (Node) theTree.get(cnt);
    if(thisNode.hasChild()){
      nodeList.add("+" + thisNode.getName());
    }
    else{
      nodeList.add(thisNode.getName());
    }
  }
  return nodeList;
}
public int printTree(Graphics g, GridTree canvas, int yOffset, int depth){
  depth++;
  if(visible){
    for(int cnt=0; cnt<theTree.size(); cnt++){
      Node thisNode = (Node) theTree.get(cnt);
      if(thisNode.hasChild()){
        yOffset = thisNode.setNodePosition(yOffset, canvas.textH, depth,_
```

```
            canvas);
          thisNode.drawNode(g,canvas);
          Tree thisTree = (Tree) thisNode;
          yOffset = thisTree.printTree(g,canvas,yOffset, depth);
        }
        else{
          yOffset = thisNode.setNodePosition(yOffset, canvas.textH, depth,_
            canvas);
          thisNode.drawNode(g,canvas);
        }
      }
    }
    return yOffset;
  }
  public Node clickTree(int x, int y){
    if(visible){
      for(int cnt=0; cnt<theTree.size(); cnt++){
        Node thisNode = (Node) theTree.get(cnt);
        if(thisNode.yIsBetween(y)){
          return thisNode;
        }
        else{
          if(thisNode.hasChild()){
            Tree thisTree = (Tree) thisNode;
            Node childClick=thisTree.clickTree(x, y);
            if(childClick!=null){
              return childClick;
            }
          }
        }
      }
    }
    return null;
  }
}
```

D.24 org.grid.client.awt : MonitorInterface_notree

```
package org.grid.client.awt;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.Timer;
import org.grid.client.MonitorConnection;
public class MonitorInterface extends Frame{
//  private SiteData siteData;
  private Button btnRefreshSiteData = new Button("Refresh Site Data");
  private Button btnPollProduct = new Button("Poll Product");
  private List lstNodes = new List(10);
  private List lstProducts = new List(10);
  private TextField txtValue = new TextField();
  private AnimGraph myGraph;
  private Timer pollTimer;
  private boolean pollStarted=false;
  private MonitorConnection monitorServlet;
  private String siteName="UniLan";
  private String nodeName;
  private String productName;
  private String pollNodeName;
  private String pollProductName;
  public MonitorInterface(String server){
    setBounds(0,0,730,520);
    setResizable(false);
    setLayout(null);
    this.drawGui();
    this.connectMonitor(server);
    this.addWindowListener(
      new WindowAdapter(){//anonymous class definition
        public void windowClosing(WindowEvent e){
          System.exit(0);//terminate the program
        }
      }
    );
  }
  private void connectMonitor(String server){
```

```
    try {
      monitorServlet = new MonitorConnection(server + "/gridmonitor/Monitor");
    }
    catch(java.net.ConnectException e){
      System.exit(0);
    }
  }
  private void drawGui(){
    setLayout(null);
add(btnRefreshSiteData);
add(btnPollProduct);
    add(lstNodes);
add(lstProducts);
    add(txtValue);
    lstNodes.setBackground(Color.white);
    lstProducts.setBackground(Color.white);
    txtValue.setBackground(Color.white);
btnRefreshSiteData.setBounds(10,40,200,25);
btnPollProduct.setBounds(10,460,200,25);
    lstNodes.setBounds(10,80,200,150);
    lstProducts.setBounds(10,240,200,200);
    txtValue.setBounds(220,450,500,35);
    btnPollProduct.setEnabled(false);
btnRefreshSiteData.addActionListener(new ActionListener(){
     public void actionPerformed(ActionEvent event){
    Vector nodeList = monitorServlet.getNodeList(siteName);
         lstNodes.removeAll();
         lstProducts.removeAll();
         txtValue.setText("");
         btnPollProduct.setEnabled(false);
         for(int cnt=0;cnt<nodeList.size();cnt++){
            String nodeName = (String)nodeList.get(cnt);
            lstNodes.add(nodeName);
         }
      }
    });
    lstNodes.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent event){
        nodeName = lstNodes.getSelectedItem();
```

```
        lstProducts.removeAll();
        txtValue.setText("");
        btnPollProduct.setEnabled(false);
        Vector products = monitorServlet.getProductList(siteName,nodeName);
        for(int cnt = 0; cnt < products.size(); cnt++){
          lstProducts.add((String) products.get(cnt));
        }
      }
    });
    lstProducts.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent event){
        productName = lstProducts.getSelectedItem();
        txtValue.setText(monitorServlet.getProductValue(siteName,nodeName,_
          productName));
        if(productName.startsWith("data") && !pollStarted){
          btnPollProduct.setEnabled(true);
        }
        else{
          btnPollProduct.setEnabled(false);
        }
      }
    });
    btnPollProduct.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent event){
        pollNodeName = nodeName;
        pollProductName = productName;
        myGraph = new AnimGraph(100,50);
        add(myGraph);
        myGraph.setBounds(220,40,500,400);
        myGraph.start();
        pollTimer = new Timer(3000, pollProduct);
        pollTimer.start();
        pollStarted = true;
        btnPollProduct.setEnabled(false);
      }
    });
}
  ActionListener pollProduct = new ActionListener() {
      public void actionPerformed(ActionEvent event) {
```

```
        String valueString = monitorServlet.getProductValue(siteName,_
          ollNodeName,pollProductName);
        txtValue.setText(valueString);
        int valueInt = Integer.parseInt(valueString);
        myGraph.insertData(valueInt);
      }
  };
}
```

D.25 org.grid.client.ssl : PromiscuousHttpsMessage

```
package org.grid.client.ssl;
import com.oreilly.servlet.*;
import java.net.*;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.Security;
import javax.net.ssl.*;
public class PromiscuousHttpsMessage extends HttpMessage {
  public PromiscuousHttpsMessage(URL servlet) {
    super(servlet);
    System.out.println("PromiscuousHttpsMessage is being used");
    //System.setProperty("javax.net.ssl.keyStore","/usr/java/tomcat/._
      keystore");
    //System.setProperty("javax.net.ssl.keyStorePassword","fuckyou");
    //use our own trust manager so we can always trust
    //the URL entered in the configuration.
    //X509TrustManager tm = new PromiscuousX509TrustManager();
    PromiscuousX509TrustManager tm = new PromiscuousX509TrustManager();
    KeyManager[] km = null;
    X509TrustManager[] tma = new X509TrustManager[] { tm };
    try {
      SSLContext sslContext = SSLContext.getInstance("TLS");
      //SSLv3");
      sslContext.init(km, tma, new java.security.SecureRandom());
      SSLSocketFactory sf1 = sslContext.getSocketFactory();
      HttpsURLConnection.setDefaultSSLSocketFactory(sf1);
      HttpsURLConnection.setDefaultHostnameVerifier(new _
```

```
        PromiscuousHostnameVerifier());
    }
    catch (NoSuchAlgorithmException e) {
      System.out.println("_Fucked1");
      e.printStackTrace(System.out);
    }
    catch (KeyManagementException e) {
      System.out.println("_Fucked2");
      e.printStackTrace(System.out);
    }
    //SSLSetup.setDebug();
  }
}
```

D.26 org.grid.client.ssl : PromiscuousHostnameVerifier

```
package org.grid.client.ssl;
import javax.net.ssl.*;
public class PromiscuousHostnameVerifier implements HostnameVerifier {
  public boolean verify(String hostname, SSLSession session) {
    System.out.println("PromiscuousHostnameVerifier lied about 'verify'");
    return true;
  }
}
```

D.27 org.grid.client.ssl : PromiscuousX509TrustManager

```
package org.grid.client.ssl;
import java.security.cert.X509Certificate;
import javax.net.ssl.*;
/*
import com.sun.net.ssl.X509TrustManager;
*/
/**
* A trust manager which trusts a client and server certificates.
```

```
* Used by SSLSetup class.
*/
public class PromiscuousX509TrustManager implements X509TrustManager {
  public X509Certificate[] getAcceptedIssuers() {
    System.out.println("PromiscuousX509TrustManager lied about _
      'getAcceptedIssuers'");
   return null;
  }
  public boolean isClientTrusted(X509Certificate[] chain) {
    System.out.println("PromiscuousX509TrustManager lied about _
      'isClientTrusted'");
   return true;
  }
  public boolean isServerTrusted(X509Certificate[] chain) {
    System.out.println("PromiscuousX509TrustManager lied about _
      'isServerTrusted'");
   return true;
  }
  public void checkServerTrusted(X509Certificate[] chain, String authType) {}
  public void checkClientTrusted(X509Certificate[] chain, String authType) {}
}
```

D.28 org.grid.plugins.modules.procmon.serialised : Mib

```
package org.grid.plugins.modules.procmon.serialised;
import java.rmi.*;
import java.util.*;
/** The RMI client will use this interface directly.
 *  The RMI server will make a real remote object that
 *  implements this, then register an instance of it
 *  with some URL.
 */
public interface Mib extends Remote {
  public Vector getOidList() throws RemoteException;
  public Vector getBulk(Vector oids) throws RemoteException;
  public String get(String oidPath) throws RemoteException;
}
```

D.29 org.grid.plugins.modules.procmon.serialised : MibImpl

```
package org.grid.plugins.modules.procmon.serialised;
import java.io.*;
import java.util.*;
import java.rmi.*;
import org.grid.support.Debug;
import java.rmi.server.UnicastRemoteObject;
public class MibImpl extends UnicastRemoteObject implements Mib {
  private String binPath = System.getProperty("user.dir");
  private Vector oidPaths;
  public MibImpl() throws RemoteException {
    super();
    oidPaths = readMib();
    Debug.out("Mib loaded");
  }
  private String escapeSpace(String s){
    return new String(s.replaceAll(" ", "\\ "));
  }
  public Vector getOidList(){
    return oidPaths;
  }
public Vector getBulk(Vector oids){
  String thisOid;
  Vector results = new Vector();

    for(Iterator i = oids.iterator(); i.hasNext(); ){
      thisOid = (String) i.next();
      results.add(get(thisOid));
    }

  return results;
}

private Vector readMib(){

    Vector oidVect = new Vector();
    FileReader fileReader;
    BufferedReader buffReader;
```

```
    String readLine;
    String mibFile=escapeSpace(binPath + "/mib/mib.conf");
    Debug.out("Loading mib from: " + mibFile);
    try{
      fileReader = new FileReader(mibFile);
      buffReader = new BufferedReader(fileReader);
      while(buffReader.ready()){
        readLine=buffReader.readLine();
        //Debug.out("I read: " + readLine);
        oidVect.add(readLine);
      }
    }
    catch(IOException e){
       Debug.out("Error reading mib file: " + mibFile);
    }
    return oidVect;
  }
  public String get(String oidPath){
    String thisOid;
    boolean knownOid = false;
    for(Iterator i = oidPaths.iterator(); i.hasNext(); ){
      thisOid = (String) i.next();
      //Debug.out(oidPath + "? " + thisOid);
      if(thisOid.equals(oidPath)){
        knownOid = true;
      }
    }
    if(knownOid){
      if(oidPath.endsWith(".sh")){
        Debug.out("Execute: " + oidPath);
        return execute(oidPath);
      }
      else{
        Debug.out("Read: " + oidPath);
        if(oidPath.startsWith("/")){
          return lineReader(oidPath);
        }
        else{
          return lineReader(binPath + "/mib/" + oidPath);
```

```
        }
      }
    }
    else{
      Debug.out("Error unknown oid: " + oidPath);
      return "";
    }
  }
/**
  public ProductList walk(){
  }
**/
  private String lineReader(String file){
    FileReader fileReader;
    BufferedReader buffReader;
    String readLine;
    file=escapeSpace(file);
    try{
      fileReader = new FileReader(file);
      buffReader = new BufferedReader(fileReader);
      readLine=buffReader.readLine();
      return readLine;
    }
    catch(IOException e){
      Debug.out("Error reading file: " + file);
    }
    return "";
  }
  private String execute(String cmd){
   String readLine;
  String allText = "";
  String cmdPath = escapeSpace(binPath + "/mib/mibscripts/" + cmd);
    try {
    Process ls_proc = Runtime.getRuntime().exec(cmdPath);
    DataInputStream    buffIn = new DataInputStream(ls_proc._
      getInputStream());
  while ((readLine = buffIn.readLine()) != null) {
    allText = allText + "\n" + readLine;
  }
```

```
  }
    catch (Exception e) {
      Debug.out("Error executing: " + cmdPath);
    return "";
  }

  return allText.trim();
}

}
```

D.30 org.grid.plugins.modules.procmon : Monitor

```
package org.grid.plugins.modules.procmon;
import org.grid.plugins.modules.*;
import java.util.*;
import org.grid.monitor.agent.*;
import org.grid.support.*;
import java.net.*;
import org.grid.client.MonitorConnection;
import org.grid.mds.MdsGridData;
import java.rmi.*; // For Naming, RemoteException, etc.
import java.net.*; // For MalformedURLException
import java.io.*;  // For Serializable interface
import org.grid.plugins.modules.procmon.serialised.*;
public class Monitor extends MonitorDef{
  private ProcmonClient client = null;
  public Monitor(String user, String pass, String site, String host) _
    throws AgentException{
    super(user, pass, site, host);
    clientAgent="procmon";
    //Debug.out(clientAgent + " dynamicaly loaded ok");
  try{
  client = new ProcmonClient(hostName);
}
catch(AgentException e){
  //Debug.out(clientAgent + " error during constructing");
  throw new AgentException("");
```

```
  }
}
protected String getOidValue(String oidName){
  return client.get(oidName);
}
private class ProcmonClient{
  String hostAddress;
 public ProcmonClient(String host) throws AgentException{
 try{
   //Debug.out("ProcmonClient: Checking hostname");
   InetAddress.getByName(host); // test hostname is real, class _
     will fail if it doesn't resolve
      hostAddress = host;
      //Debug.out("ProcmonClient: Testing first oid");
      String testValue = get("/proc/version");
      if(testValue.equals("")){
        //Debug.out("ProcmonClient: Got test oid but value was null");
        throw new AgentException("Got test oid but value was null");
      }
    }
    catch(Exception e) {
      //Debug.out("Rmi can't connect to host: " + hostAddress);
      throw new AgentException("Rmi can't connect to host: " + hostAddress);
    }
    //Debug.out("ProcmonClient: Constructed ok");
  }
  public String get(String oid){
    try{
      // Get remote object and store it in remObject:
      //Debug.out("ProcmonClient: Starting mib connection");
      Mib remoteMib = (Mib)Naming.lookup("rmi://" + hostAddress + "/Mib");
      //Debug.out("ProcmonClient: Calling remote method");
      return remoteMib.get(oid);
    }
    catch(Exception e){
      return "";
    }
  }
  public Vector getProductList(){
```

```
      return null;
    }
  /**
        Debug.out();
        System.out.println("Getting oid list");
        String thisOid;
        Vector oidList = remoteMib.getOidList();
        for(Iterator i = oidList.iterator(); i.hasNext(); ){
          thisOid = (String) i.next();
          System.out.println("  " + thisOid);
        }
    }
  **/
  }
}
```

D.31 org.grid.plugins.modules.snmp : Monitor

```
package org.grid.plugins.modules.snmp;
import org.grid.plugins.modules.*;
import java.util.*;
import org.grid.monitor.agent.*;
import org.grid.support.*;
import com.snmp.*;
import java.net.*;
public class Monitor extends MonitorDef{
  private SNMPClient client;
  public Monitor(String user, String pass, String site, String host) _
    throws AgentException{
    super(user, pass, site, host);
    clientAgent="snmp";
    //Debug.out(clientAgent + " dynamicaly loaded ok");
  try{
  client = new SNMPClient(hostName, "public");
}
catch(AgentException e){
  //System.out.println(e);
  throw new AgentException("");
```

```
  }
}
protected String getOidValue(String oidName){
  return client.get(oidName);
}
protected Vector getBulkOidValues(Vector products){
  return null;
}
private class SNMPClient {
  InetAddress hostAddress = null; // Host name of agent we are going for
  String snmpCommunity; // We set snmp community name once when we start
  int snmpVersion = 0; // SNMPv1
  String hostIp;
 public SNMPClient(String host, String community) throws AgentException{
 try{
   snmpCommunity = community;
   hostAddress = InetAddress.getByName(host); // test hostname is _
     real, class will fail if it doesn't resolve
     hostIp = hostAddress.getHostAddress();
     /**
     String testValue = get("1.3.6.1.2.1.1.1.0");
     if(testValue.equals("")){
       throw new AgentException("Can't get test oid");
     }
     **/
   }
   catch (java.net.UnknownHostException e) {
     throw new AgentException("Unknown host");
   }
 }
 public Vector getProductList(){
   return null;
 }
 public String get(String oidName){
  //Debug.out("SNMPget: " + oidName + " (" + snmpCommunity +")");

   SNMPv1CommunicationInterface comInterface;
   String oidValue = new String();
 try{
```

```
      comInterface = new SNMPv1CommunicationInterface(snmpVersion, _
        hostAddress, snmpCommunity);

      try{
SNMPVarBindList newVars = comInterface.getMIBEntry(oidName);

  SNMPSequence pair = (SNMPSequence)(newVars.getSNMPObjectAt(0));
SNMPObjectIdentifier snmpOID = (SNMPObjectIdentifier)pair._
  getSNMPObjectAt(0);
SNMPObject snmpValue = pair.getSNMPObjectAt(1);
String typeString = snmpValue.getClass().getName();

if (typeString.equals("snmp.SNMPOctetString")){
String snmpString = snmpValue.toString();

// truncate at first null character
int nullLocation = snmpString.indexOf('\0');
if (nullLocation >= 0){
snmpString = snmpString.substring(0,nullLocation);
}
         oidValue = snmpString;

}
else{
         oidValue = snmpValue.toString();
}
       return oidValue;
      }
      catch (SNMPGetException e){
        //Debug.out("SNMP oid '" + oidName + "' doesn't exist on host " _
          + hostIp);
      }
      catch (SNMPBadValueException e){
        //Debug.out("SNMP bad value returned");
      }
      catch (java.io.IOException e) {
        //Debug.out("SNMP time out connecting to host: " + hostIp);
  }
  }
```

```
      catch (java.net.SocketException e) {
        //Debug.out("SNMP error connecting to host: " + hostIp);
   }

   return "";


    }
  }
}
```

D.32 org.grid.plugins.modules : MonitorDef

```
package org.grid.plugins.modules;
import org.grid.plugins.*;
import java.util.*;
import org.grid.support.*;
public class MonitorDef implements PluginDef{
  protected String hostName;
  protected String siteName;
  protected String userName;
  protected String userPassword;
  protected Vector productCapabilities = null;
  protected String clientAgent = "";
  public MonitorDef(String user, String pass, String site, String host){
    hostName = host;
    siteName = site;
    userName = user;
    userPassword = pass;
  }
  public MonitorDef(){
  }
  public Class[] getParamTypes(){
    try{
      Class string = Class.forName("java.lang.String");
      Class[] paramTypes = { string, string, string, string };
      return paramTypes;
    }
```

```
    catch(java.lang.ClassNotFoundException e){
      e.printStackTrace();
      System.exit(1);
      return null;
    }
  }
  public Vector capable(){
    ProductMapDef pMap = (ProductMapDef) PluginManager.loadPlugin("ProductMap",
    Vector productMapList = pMap.getMap(clientAgent);
    Vector productNames = new Vector();
productCapabilities = new Vector();

// now we check each oid is available and trim the map
// so the map will only contain products we can fetch

    int mibSize=productMapList.size();
    for(int cnt = 0; cnt < mibSize; cnt++){
      StringPair map = (StringPair) productMapList.get(cnt);
      // map.name = the oid (1.21.123.3.12.3)
      // map.value = the product (infoCpuName)
      if(getOidValue(map.name).equals("")){
        //Debug.out("Agent: " + clientAgent + ": " + map.name + " removed _
          from capablity");
      }
      else{
        //Debug.out("Agent: " + clientAgent + ": " + map.name + " added to _
          capablity");
        productCapabilities.add(map);
        productNames.add(map.value);
      }
    }

  return productNames;

  }
  public String clientAgent(){
    return clientAgent;
  }
  public String get(String productName){
```

```
    for(int cnt = 0; cnt < productCapabilities.size(); cnt++){
      StringPair map = (StringPair) productCapabilities.get(cnt);
      if(map.value.equals(productName)){
        return getOidValue(map.name);
      }
    }
    return "";
  }
  protected Vector getBulkOidValues(Vector products){
    return null;
  }
  protected String getOidValue(String oidName){
    System.out.println("Not overridden function");
    return null;
  }
}
```

D.33 org.grid.plugins.modules : DataStoreDef

```
package org.grid.plugins.modules;
public class DataStoreDef {
  public void printClass(){
    System.out.println("I am A Data Store");
  }
}
```

D.34 org.grid.plugins.modules : SecurityDef

```
package org.grid.plugins.modules;
import org.grid.plugins.PluginDef;
public class SecurityDef implements PluginDef{
  public SecurityDef(){
  }
  public boolean authenticate(String siteName, String userName, String _
    userPassword){
    return true;
  }
```

```
  public Class[] getParamTypes(){
    return null;
  }
}
```

D.35 org.grid.plugins.modules.local : ProductMap

```
package org.grid.plugins.modules.local;
import org.grid.plugins.modules.*;
import java.util.*;
import java.net.*;
import java.io.*;
import java.sql.*;
import org.grid.support.*;
public class ProductMap extends ProductMapDef{
  public Vector getMap(String agentClass){
    Vector pMaps = new Vector();
try{
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
    }
catch (Exception e) {
       Debug.out("Couldn't load MYSQL driver");
    }
try{
      // Create a Connection and a Statement
      Connection conn =DriverManager.getConnection("jdbc:mysql://_
        127.0.0.1:3306/project", "snmp", "java");
      Statement stmt = conn.createStatement();
ResultSet resultSet =  stmt.executeQuery ("SELECT _
  product_mappings.oid, products.name FROM product_mappings,_
    products WHERE products.id=product_mappings.productid AND _
      product_mappings.agent='"+ agentClass + "';");
//Debug.out("SELECT oid, value FROM test WHERE host='" + _
  hosts[cnt][0] + "' order by stamp desc limit 0,1;");
         while(!resultSet.isLast()){
           resultSet.next();
           StringPair map = new StringPair(resultSet.getString(1), _
             resultSet.getString(2));
```

```
            pMaps.add(map);
          }

    }
catch (java.sql.SQLException e) {
      Debug.out("Couldn't execute sql command -" + agentClass);
      System.out.println(e);
    }
    return pMaps;
  }
}
```

D.36 org.grid.plugins.modules.remote : Monitor

```
package org.grid.plugins.modules.remote;
import org.grid.plugins.modules.*;
import java.util.*;
import org.grid.monitor.agent.*;
import org.grid.support.*;
import java.net.*;
import org.grid.client.MonitorConnection;
import org.grid.mds.MdsGridData;
public class Monitor extends MonitorDef{
  private ServletClient client = null;
  public Monitor(String user, String pass, String site, String host) throws_
     AgentException{
    super(user, pass, site, host);
    clientAgent="servlet";
    //Debug.out(clientAgent + " dynamicaly loaded ok");
  try{
  client = new ServletClient(user, pass, site, host);
}
catch(AgentException e){
  throw new AgentException("");
    }
  }
  public Vector capable(){
    return client.getProductList();
```

```
  }
  protected String getOidValue(String oidName){
    return client.get(oidName);
  }
  private class ServletClient{
    String nodeName;
    String siteName;
    String userName;
    String userPassword;
    MonitorConnection monitorServlet;
   public ServletClient(String user, String pass, String site, String node)_
     throws AgentException{
      //Debug.out("ServletClient has loaded for: " + site + "." + node);
      nodeName=node;
      siteName=site;
      userName = user;
      userPassword = pass;
      try {
        String siteUrl=MdsGridData.getServletUrl(siteName);
        monitorServlet = new MonitorConnection(siteUrl, userName, _
          userPassword);
      }
      catch(java.net.ConnectException e){
        // couldnt connect
      }
    }
    public Vector getProductList(){
      return monitorServlet.getProductList(siteName, nodeName);
    }
    public String get(String productName){
      return monitorServlet.getProductValue(siteName, nodeName, productName);
    }
  }
}
```

D.37 org.grid.plugins.modules.remote : Security

```
package org.grid.plugins.modules.remote;
```

```
import org.grid.plugins.modules.*;
import java.util.*;
import org.grid.monitor.agent.*;
import org.grid.support.*;
import java.net.*;
import org.grid.client.MonitorConnection;
import org.grid.mds.MdsGridData;
public class Security extends SecurityDef{
  public boolean authenticate(String siteName, String userName, String _
    userPassword){
    if(testPassword(siteName, userName, userPassword)){
      Debug.out("remote security module thinks password is valid");
      return true;
    }
    else{
      Debug.out("remote security module thinks password is wrong");
      return false;
    }
  }
  private static boolean testPassword(String siteName, String userName, _
    String userPassword){
    MonitorConnection monitorServlet;
    try {
      String siteUrl=MdsGridData.getServletUrl(siteName);
      monitorServlet = new MonitorConnection(siteUrl, userName, _
        userPassword);
      return true;
    }
    catch(java.net.ConnectException e){
      // couldnt connect
      // if we cant reach the site we cant authenticate
      // so this will fail
      return false;
    }
  }
}
```

D.38 org.grid.plugins.modules : ProductMapDef

```
package org.grid.plugins.modules;
import org.grid.plugins.PluginDef;
import java.util.Vector;
public class ProductMapDef implements PluginDef{
  public Class[] getParamTypes(){
    return null;
  }
  public Vector getMap(String agentClass){
    return null;
  }
}
```

D.39 org.grid.plugins.modules.jcifs : Security

```
package org.grid.plugins.modules.jcifs;
import org.grid.plugins.modules.*;
import jcifs.UniAddress;
import jcifs.smb.SmbSession;
import jcifs.smb.SmbAuthException;
import jcifs.smb.SmbException;
import jcifs.smb.NtlmPasswordAuthentication;
import java.util.*;
import org.grid.support.*;
public class Security extends SecurityDef{
  public boolean authenticate(String siteName, String userName, String _
    userPassword){
    if(testPassword(userName, userPassword)){
      Debug.out("jcifs security module thinks password is valid");
      return true;
    }
    else{
      Debug.out("jcifs security module thinks password is wrong");
      return false;
    }
  }
  private static boolean testPassword(String userName, String userPassword){
```

```
    try{
      String sambaDomainController = "mrflibble";
      String sambaDomain = "does not seem to fucking matter";
      UniAddress dc = UniAddress.getByName( sambaDomainController );
      NtlmPasswordAuthentication auth = new NtlmPasswordAuthentication( _
        sambaDomain, userName, userPassword );
      SmbSession.logon( dc, auth );
      return true;
    }
    catch( SmbAuthException sae ) {
      // auth failure - bad user / pass combo
      return false;
    }
    catch( SmbException se ) {
      // samba failure - couldnt authenticate with teh server for some reason
      System.out.println("Samba auth - problem contacting server");
      return false;
    }
    catch( java.net.UnknownHostException e ) {
      // samba failure - couldnt aresolv the server hostname
      System.out.println("Samba auth - couldn't resolve auth serer name");
      return false;
    }
  }
}
```

D.40 org.grid.plugins : PluginDef

```
package org.grid.plugins;
public interface PluginDef {
  public Class[] getParamTypes();
}
```

D.41 org.grid.plugins : PluginManager

```
package org.grid.plugins;
import org.grid.plugins.modules.*;
```

```
import org.grid.support.Debug;
import java.util.*;
import java.io.*;
import java.lang.reflect.*;
import org.grid.support.*;
public class PluginManager {
  private static Class[] getParamTypes(String className){
    //System.out.println("Fetching param types for " + className);
    className = "org.grid.plugins.modules." + className + "Def";
    PluginDef loadedObject = (PluginDef) createDynamicObject(className, null, nu
    if(loadedObject == null){
      System.out.println("Falied to load: " + className);
      System.exit(1);
      return null;
    }
    else{
      return loadedObject.getParamTypes();
    }
  }
  public static Object loadPlugin(String className, String filter, Object[] _
    params){
    Vector loadedObjects = loadPlugins(className, filter, params);
    if(loadedObjects.isEmpty()){
      // when ever we ask for a plugin if we return nothing teh program will_
        bust
      // there should always be a plugin to saticfy the query
      // so we stop
      System.out.println("Can't load at least one plugin using query:");
      System.out.println(className + " " + filter);
      System.exit(1);
      return null;
    }
    else{
      return loadedObjects.get(0);
    }
  }
  public static Vector loadPlugins(String className, String filter, _
    Object[] params){
    //System.out.println("Discovering plugins");
```

```
    Vector loadedObjects = new Vector();
    Class[] paramTypes = getParamTypes(className);
    Vector modules = discoverPlugins(className, filter);
    for(int i=0; i<modules.size();i++){
      String plugin = (String) modules.get(i);
      Object loadedObject = createDynamicObject(plugin, paramTypes, params);
      if(loadedObject != null){
        loadedObjects.add(loadedObject);
      }
      else{
        Debug.out("Failed to load plugin: " + plugin);
      }
    }
    if(loadedObjects.isEmpty()){
      // when ever we ask for a plugin if we return nothing teh program
      // will bust
      // there should always be a plugin to saticfy the query
      // so we stop
      Debug.out("Can't load at least one plugin using query: " + className_
        + " " + filter);
      return null;
    }
    else{
      return loadedObjects;
    }
  }
  private static Object createDynamicObject(String className, Class[] _
    paramTypes, Object[] params){
    try{
      //System.out.println("Trying to load: " + className);
      // get reference to the class
      Class classRef = Class.forName( className );
      if(paramTypes == null){
        params = null;
      }
      Constructor constr = classRef.getConstructor(paramTypes);
      Object newObject = constr.newInstance(params);
      if(newObject != null){
        Debug.out("Dynamicaly loaded: " + className);
```

```
    }
    return newObject;

  }
  catch(java.lang.ClassNotFoundException e){
    System.out.println("Couldn't find class: " + className);
    return null;
  }
  catch(java.lang.Exception e){
    System.out.println("Couldn't dynamically load class: " + className);
    return null;
  }
}
private static boolean filterCmd(String filter, String item){
  boolean result = false;
  char command = filter.charAt(0);
  filter = filter.substring(1);
  if(command == '!'){
    if(filter.equals(item)){
      result = true;
    }
    else{
      result = false;
    }
  }
  if(command == '='){
    if(filter.equals(item)){
      result = false;
    }
    else{
      result = true;
    }
  }
  return result;
}
private static Vector discoverPlugins(String classOfPlugin, String filter){
  String rootPluginPath = "/usr/java/lib/classes/org/grid/plugins/modules/";
  String classPathStub = "org.grid.plugins.modules.";
  Vector foundModules = new Vector();
```

```
    File dir = new File(rootPluginPath);
    // This filter only returns directories
    FileFilter fileFilter = new FileFilter() {
        public boolean accept(File file) {
            return file.isDirectory();
        }
    };
    File[] dirs = dir.listFiles(fileFilter);
    for (int i=0; i<dirs.length; i++) {
      // Get filename of file or directory
      String dirName = dirs[i].getName();
      //System.out.println("Found dir: " + dirName);
      if(!filterCmd(filter, dirName)){
        String pathToClass = rootPluginPath + dirName + File.separator + _
          classOfPlugin;
        String pathToClassFileName = pathToClass + ".class";
        //System.out.println("Module to look for is:" + pathToClassFileName);
        File moduleFile = new File(pathToClassFileName);
        if(moduleFile.exists()){
          //System.out.println("Could load: " + pathToClassFileName);
          pathToClass = pathToClass.replaceAll(rootPluginPath, "");
          foundModules.add(classPathStub + pathToClass.replaceAll(File._
            separator, "."));
        }
      }
    }
    return foundModules;
  }
}
```

D.42 org.grid.security : SecurityManager

```
package org.grid.security;
import org.grid.mds.*;
import org.grid.mds.rgmb.*;
import org.grid.plugins.PluginManager;
import org.grid.plugins.modules.SecurityDef;
import java.util.*;
```

```
import org.grid.support.*;
public class SecurityManager{
  public static boolean authenticate(String localSiteName, String _
    userNameSitePair, String userPassword){
    // usernames take the form of user@site
    // but they are stored as user in the home mds with site as the parent_
      node
    StringTokenizer st;
    try{
      st = new StringTokenizer(userNameSitePair, "@");
    }
    catch(java.lang.NullPointerException e){
      Debug.out("THIS IS A FUCKUP - the user@site pair is munged (look for _
        the @): " + userNameSitePair);
      return false;
    }
    int numTokes = st.countTokens();
    if(numTokes != 2){
      Debug.out("THIS IS A FUCKUP - the user@site pair is munged (look _
        for the @): " + userNameSitePair);
      return false;
    }
    String userName = st.nextToken();
    String siteName = st.nextToken();
    Debug.log(true, "security", "Authenticating " + userName + "@" + _
      siteName);
    Stack dataStack = new Stack();
    dataStack = Mds.addPairToStack(dataStack, "userPassword", "");
    dataStack = Mds.addPairToStack(dataStack, "userName", userName);
    dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
    Vector retrievedPasswordList;
    boolean authenticatedOk = false;
    if(Mds.checkLease((Stack)dataStack.clone())){
      //Debug.out("Good lease");
      retrievedPasswordList = Mds.getFromCache(dataStack);
      if(!retrievedPasswordList.isEmpty()){
        String retrievedPassword = (String) retrievedPasswordList.get(0);
        //Debug.out("retrieved password: " + retrievedPassword);
          Debug.log(true, "security", "Cache hit: " + userName + "@" + _
```

```
          siteName);
      if(userPassword.equals(retrievedPassword)){
        authenticatedOk = true;
      }
    }
    else{
      Debug.log(true, "security", "Cache miss: " + userName + "@" + _
        siteName);
    }
  }
  else{
    if(siteName.equals(localSiteName)){
      // its a local user so use the GSM for this site
      //Debug.out("Using local gsm to get password");
      Vector gsms = PluginManager.loadPlugins("Security", "!remote", null);
      if(gsms == null){
        // we couldnt load any security authentication modules which is bad
        // nothing can be locally authenticated but remote authentication
        // should work we cant test these passwords so we must bail out
        // and return false
        Debug.out("Couldn't load any local authentication modules");
      }
      else{
        // we got a vector of security modules we can use
        // we loop through until the password is authenticated
        // this means the behaviour is all local security modules are used
        // one after the other - we are combining the authentication
        // methods
        for(int i=0; i<gsms.size(); i++){
          SecurityDef gsm = (SecurityDef) gsms.get(i);
          authenticatedOk = gsm.authenticate(siteName, userName, _
            userPassword);
          if(authenticatedOk){
            // stop the loop now
            i=gsms.size();
          }
        }
      }
    }
```

```
    else{
      // this user isn't one of ours, we need to do a remote password get
      // from their site, this data will be cached in the standard way
      Debug.out("Going to do a remote fetch for the password");
      SecurityDef gsm = (SecurityDef) PluginManager.loadPlugin_
        ("Security", "=remote", null);
      if(gsm==null){
        Debug.out("Couldn't load the remote authentication module");
      }
      else{
        authenticatedOk = gsm.authenticate(siteName, userName, _
          userPassword);
      }
    }
    if(authenticatedOk){
      // password was ok so cache it
      cacheUserNameAndPassword(siteName, userName, userPassword);
    }
  }
  if(authenticatedOk){
    Debug.log(true, "security", "Access granted for user: " + userName _
      + "@" + siteName);
  }
  else{
    Debug.log(true, "security", "Access denied for user: " + userName _
      + "@" + siteName);
  }
  return authenticatedOk;
}
private static void cacheUserNameAndPassword(String siteName, _
  String userName, String userPassword){
  Stack dataStack = new Stack();
  dataStack = Mds.addPairToStack(dataStack, "userName", "");
  dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
  Mds.setCache((Stack)dataStack, userName);
  dataStack = Mds.addPairToStack(dataStack, "userPassword", "");
  dataStack = Mds.addPairToStack(dataStack, "userName", userName);
  dataStack = Mds.addPairToStack(dataStack, "siteName", siteName);
  Mds.setCache((Stack)dataStack, userPassword);
```

```
  }
}
```

D.43 procmon : ProcmonServer

```
import org.grid.support.Debug;
import org.grid.plugins.modules.procmon.serialised.*;
import java.rmi.*;
import java.rmi.registry.*;
public class ProcmonServer{
  public static void main(String[] args) {
    try {
      Registry reg = LocateRegistry.createRegistry(1099);
      MibImpl localObject = new MibImpl();
      reg.rebind("Mib", localObject);
    }
    catch(RemoteException re) {
      System.out.println("RemoteException: " + re);
    }
  }
}
```