jGMA: A Reference Implementation of the Grid Monitoring Architecture

Progression from MPhil to PhD Document

Matthew Grove (matthew.grove@port.ac.uk)

Distributed Systems Group, University of Portsmouth

Registration Period
01 October 2003 to 30 September 2004

Date Submitted
6 October 2004

Table of Contents

1 Statement of Aims

The aims of this project are to study, investigate, and then develop a framework for transporting information around a distributed and heterogeneous system.

Wide-area distributed systems require scalable mechanisms that can be used to gather and distribute information to a variety of endpoints. The emerging Grid infrastructure is rapidly being taken up for technical computing as well as in business and commerce. The Distributed Systems Group for the last few years been developing a resource monitoring system known as GridRM [1], which needs to distribute information over the wide-area, between so called, GridRM gateways.

This project will attempt to create a messaging system based on emerging grid standards to address the wide-area communications needs of GridRM and more widely investigate and overcome the problems and issues associated with developing scalable distributed software.

1.1 Research Objective

The overall objective of this research is to develop a messaging system capable of providing a robust infrastructure capable of scaling over a wide-area. This includes investigating both non-blocking (event driven) and blocking messaging. We are keen to provide a robust system to fulfil GridRMs requirements for wide-area communications and are investigating the GGF's [2] GMA [3] to see if it provides the features we need (see Section 2). A key aspect of the project will be studying the means to create what is effectively a distributed database to provide a look-up service (Virtual Registry) to allow end points to be located and used. We will be investigating the issues associated with providing debugging and monitoring tools for jGMA (our messaging system), which is a necessary issue to address for a distributed system. Finally, when the framework is complete, it will be used in the context of distributed gaming. We intend to create a set of services based on jGMA that provide generic services for building grid-enabled games.

2 Background and Review

In this section we briefly discuss the various GMA implementations currently available in the spring of 2004. Table 1, shows a matrix of the features and functionality of various GMA implementations, which we use to highlight some of the the motivating factors which led us to develop jGMA. We split the GMA implementations into two categories:

- **Standalone:** systems which can be used by other software to provide GMA capabilities.
- **Embedded:** systems with GMA like capabilities, but this functionality is part of a larger piece of software.

The Grid Monitoring Architecture (GMA), is the architecture recommended by Global Grid Forum (GGF) for transporting resource monitoring information around the grid. The GMA specification sets out the requirements and constraints of any implementation; it is based on a consumer/producer paradigm with an integrated system registry. GMA was chosen because it is an emerging grid standard which seems to have the capabilities we require.

2.1 Standalone Implementations

2.1.1 R-GMA (Relational Grid Monitoring Architecture)

R-GMA [4] was developed within the European DataGrid Project [5] as a Grid information and monitoring system. R-GMA is being used both for information about the Grid (primarily to find out about what services are available at any one time) and for application monitoring. A special strength of this implementation comes from the use of the use of a relational model to search and describe the monitoring information. R-GMA is based on Java Servlet technology and uses an SQL-like API. R-GMA can be used in conjunction with C++, C, Python and Perl consumers and/or producers, as well as (obviously) with Java.

2.1.2 pyGMA (Python GMA)

pyGMA [6] from LBNL [7] is an implementation of the GMA using Python. The developers have used the object-orientated nature of Python to provide a simple inheritance-based GMA-like API. While the features of pyGMA are not comprehensive, it is easy to install and use. pyGMA is supplied with a simple registry, which is designed for testing but is not meant to be deployed. Some sample producers and consumers are provided as a starting point for developing more comprehensive services.

	R-GMA	**pyGMA**	**jGMA**	**MDS3**
Languages Supported	Java, C, C++, Python and Perl	Python	Java	C and Java
Implementation Language	Java	Python	Java	C and Java
Installation	Binary – RPMs for RedHat, Source – Python and RedHat like Linux	Uses a Python installe	Binary - one Java .jar file, Source - ANT + Apache Tomcat	GPT package management (included)
Dependencies	Ant, Java 1.4, Bouncy Castle, EDG Java Security, Jakarta Commons, Logging, Jakarta-Axis, Jas, JxUtil, Log4j, MySQL Client/Server, MySQL Java Driver, Netlogger, Prevayler, Python2, Regexp, Swig, Tomcat 4, Xerces C and Java	Python 2, Python SOAP (ZSI), Python-xml, Fpconst	Ant, Apache Tomcat, Java 1.4	Java 1.3 or better, JAAS library, Ant 1.5, Junit, YACC (or Bison), Globus Tookit 3
Transport	HTTP	(HTTP) SOAP	LAN sockets, WAN HTTP	(HTTP) SOAP
I/O Type	Streaming and Blocking	Passive and Active	Blocking and Non-blocking	Query based
Type	Standalone	Standalone	Standalone	Currently Integrated
Registry	RDBMS using MySQL	Simple	Simple/Xindice	Collection of Grid Services
Types of producers	CanonicalProducer, DataBaseProducer, LatestProducer, ResilientStreamProducer	User-based	User-based	User-based
API Size	213 calls (Java API)	46 calls	17 calls	Many calls
Security	EDG-security for authentication, SSL for transport	None	None	GSS (SSL and Certs)
Where used	In-house EDG testbed	DMF	GridRM	GT3 (many)

Table 1: A comparison of GMA implementations

2.2 Embedded GMA Implementations

The Metadata Discovery Service (MDS) [8] that is part of Globus Toolkit version 3 [9] is based on the emerging Open Grid Services Architecture (OGSA) [10]. MDS provides a broad framework within GT3, which can be used to collect, index and expose data about the state of grid resources and services. MDS3 is tailored to work with the OGSA-based Grid Services, it is, itself a distributed Grid Service. While MDS3 is an influential component within GT3, it is not suitable in its current state to use with GridRM as it requires the installation of the full GT3 toolkit.

The Network Weather Service (NWS) [11] allows the collection of resource monitoring data from a variety of sources, which can then be used to forecast future trends. NWS purports to have an architecture based on GMA, and components that exhibit GMA-like functionality. However, even though this may be the case, the GMA parts of NWS are integrated and it would be difficult to break these out of the release.

Autopilot [12] from the University of Illinois Pablo Research Group [13] is a library that can be called from an application to allow monitoring and remote control. Autopilot sensors and actuators (akin to the GMA producers/consumers) report back to a directory service called the Autopilot Manager, which allows clients to discover each other. Autopilot can be used to create standalone GMA enabled components in C++, but it requires and builds on functionality provided by the Globus Toolkit 2.

2.3 Summary

Currently the embedded versions of GMA do not easily lend themselves to being used as standalone GMA implementation; consequently they cannot be used easily in their existing form with GridRM. This leaves three alternatives, pyGMA, Autopilot and R-GMA that could be used.

Calling Python (pyGMA) from Java, which is a requirement of GridRM, is not straightforward. While the Jython project [14] allows the use of Java from within Python, there is no simple mechanism for invoking Python from within Java without creating a customised and potentially complex JNI bridge.

R-GMA does provide a Java API, and initially it was thought that R-GMA would be a suitable implementation for GridRM. However, further investigation found that R-GMA was less than ideal for our purposes; the drawbacks are discussed in [15].

3 The Prototype Architecture

3.1 Introduction

The first steps in our design were to layout a set of general criteria that we considered to be necessary and/or desirable for the system being developed. The set of criteria includes:

- Compliant to the GMA specification,
- Small well defined API,
- Minimal number of other installation dependencies,
- Simple to install and configure,

- Uses Java technologies, and fulfil GridRM's needs,
- Support both non-blocking and blocking events,
- Scale from LAN to WAN proportions.
- Fast, and have a minimal impact on its hosts,
- Scalable across thousands of sites with millions of producers / consumers.
- Choice of registry service, from a simple one, such as text-based files, to an XML-based one, for example Xindice [16], or something else, such as a relational database or Globus MDS,
- Able to work through firewalls,
- Capable of taking advantage of TLS [17] and/or the GSI [18].

These criteria were based on our experiences whilst reviewing and investigating the other GMA implementations, the needs of GridRM, and some overarching principals. Additionally, we decided to write jGMA in pure Java which allows us to take advantage of a range of Java features and related technologies, as well as providing portability via bytecode that should execute on any compliant JVM.

3.2 The jGMA architecture

3.2.1 jGMA components

In order to ensure that jGMA was easy to install, dependencies were limited to a standard JVM and Apache Tomcat [19], which provides a servlet container and a gateway that uses HTTP for inter-gateway communications. This dependency did not compromise our design criteria since GridRM requires Tomcat. Moreover, most application developers are familiar with Tomcat as it is widely used today.

jGMA consists of four components:

- A virtual registry to allow producers and consumers to discover each other,
- A Producer/Consumer servlet (PC servlet) is used for remote communicating events,
- A Consumer,
- A Producer.

Communication between components uses a shared code base, which provides wide (WAN) and local (LAN) components. jGMA has two modes of event passing. The first mode is local, where communications are within one administration domain, i.e. behind a firewall. The second mode is global, when traversing one or more administrative domains, e.g., via one or more firewall(s). For wide-area communications HTTP is used. The gateway PC servlet provides wide-area connectivity for machines which do not have direct access to the Internet.

Figure 1 shows the jGMA components being used together to provide a wide-area message-passing framework. In this figure there are two sites connected over the Internet. The consumer and producer are communicating via the PC servlets, which are handling the WAN communication on their behalf. Although, the registry is sharing a Tomcat container at one of the sites, this could, however, be hosted elsewhere. If the producer and consumer were at the same site they would communicate directly using sockets and the PC servlets would not be involved.

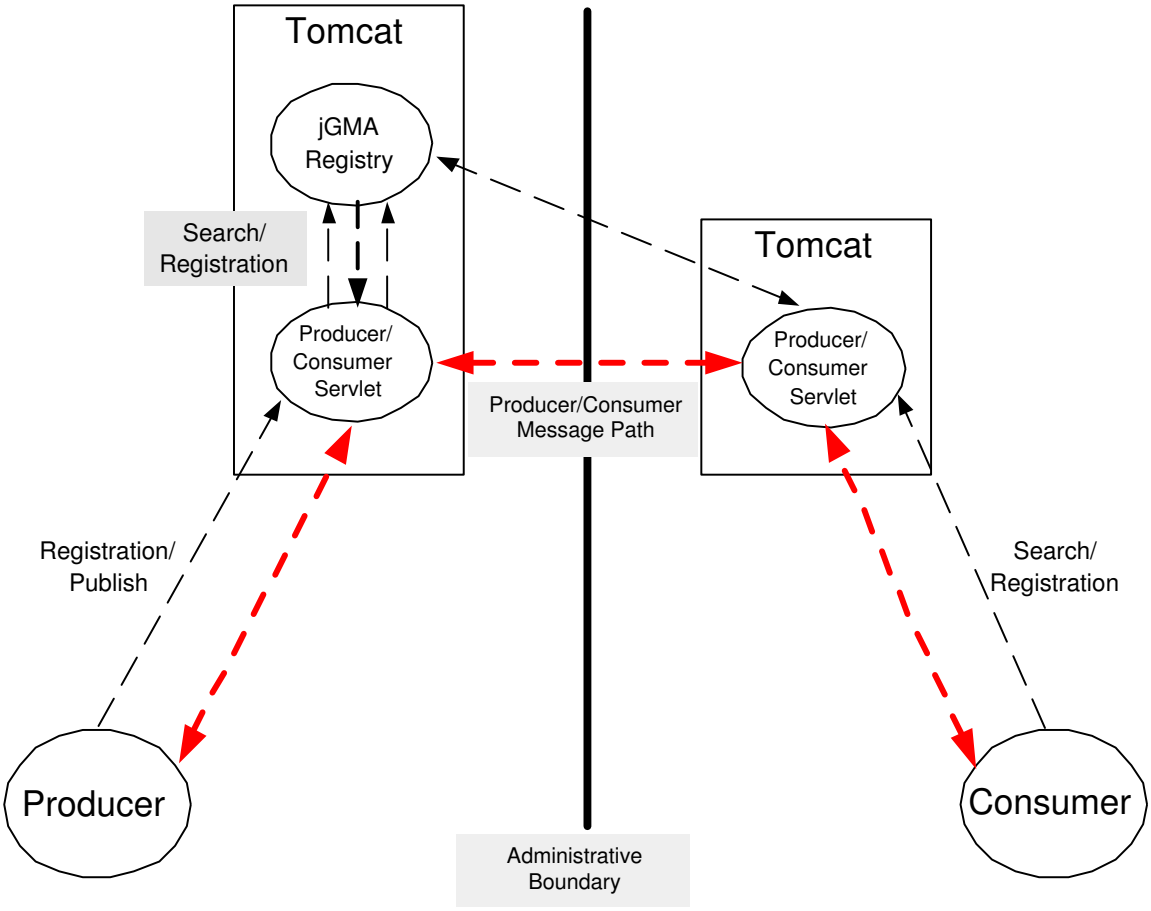

Figure 1: The jGMA Architecture

3.3 The revised jGMA implementation

After completing the first version of jGMA, it became apparent there were some problems with the software. The benchmarking process highlighted an implementation problem as it stressed the system beyond the simple tests used during the implementation stages. In addition, some engineering decisions were made about naming and addressing within jGMA, which proved to have redundant features. The changes made to address these problems are outlined in this subsection.

3.3.1 The Non-Blocking API

jGMA follows the event-driven (non-blocking) programming paradigm since it is not known when a message will be generated. In jGMA the program only executes background house keeping tasks, such as cache flushing, until a producer or consumer generates an event. In initial versions of jGMA an attempt was made to provide a combined non-blocking and traditional blocking API, this created complex software, which was difficult to debug. The API was altered to use an optional blocking wrapper layer around a stand-alone non-blocking API, illustrated in Figure 2.

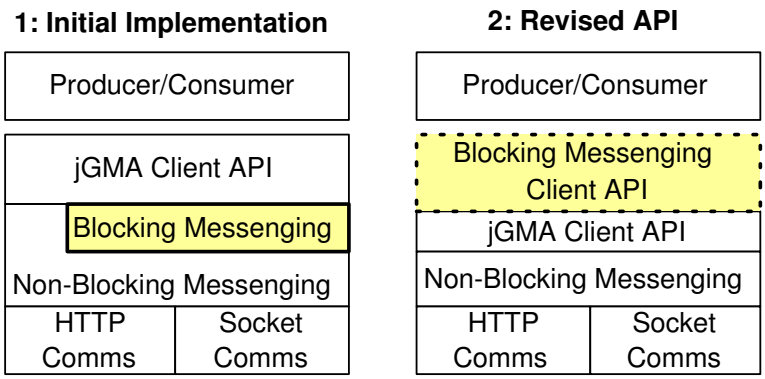

Figure 2: The Revised jGMA API

3.3.2 Revised naming

The initial addressing and naming scheme was based on implementation decisions rather than design, as the API matured parts of the old naming scheme became redundant so a new one was designed and implemented.

A client (consumer or producer) requires some basic information to send a message to another end point. When communicating over a LAN a hostname (either an IP address or a name which will resolve to one) and a socket port number could be used. Because there may be more than one jGMA client running on a single machine each client must have a unique port. An alternative would be to run a proxy server on a well-known port that would differentiate between destinations on behalf of the client.

jGMA does not make the assumption that each client can be directly reached from the Internet. The PC servlet can act as a gateway, which can accept messages from the Internet and then pass them onto a client within the LAN. Similarly jGMA does not assume that all clients have direct access to the Internet so the PC servlet can also accept messages from clients and forward them over the Internet. Clients can contact the PC servlet on a known port and have the hostname hardwired. It would be possible to discover the PC servlet using multicasting, but this manual configuration keeps the code simple. This means that there are two separate sets of addressing issues, which allow LAN and WAN communications.

A new two-tier addressing scheme was adopted [15]: a LAN address is used for socket communications and a WAN address used for inter-servlet communications using HTTP(S). To explain, by way of example, if a consumer queries the registry for a list of producers the PC servlet will translate the WAN addresses of any local producers into LAN addresses. This has the effect of hiding information about a LAN, such as IP addresses, only the URL to the PC servlet is exposed.

3.3.3 Client and Servlet Liveliness

Should a consumer, producer or PC servlet not un-register themselves due to some failure, a stale registration will be left in the registry. It is desirable for these stale records to be cleaned up automatically by jGMA.

A two-tier solution is proposed. Firstly the PC servlet will monitor and test the communications between itself and any consumers and producers registered with it, actively detecting problems with the clients and

then send a signal to the registry if a fault is detected. Secondly the registry will issue each PC servlet with a lease; if the lease expires the registry will clean out any records, which are associated with the PC servlet.

A ping-pong event is periodically sent to each registered client from the PC servlet using the standard jGMA infrastructure. If a reply is not received within a set time the servlet un-registers the client. This tests the liveliness of the jGMA infrastructure between the client and servlet as well as the liveliness of the actual client.

3.4 Summary

The jGMA library has been revised since the first implementation, after tests highlighted problems with the design. By moving the blocking API into a separate layer the software was simplified, which has made the implementation more robust. The jGMA naming now uses a standard URL format and the two-tier approach to addresses minimises the amount of local information, which is published in the registry, and this improves security.

The jGMA API is relatively small; currently there are only 17 methods in the API. Building on the current basic API and utilising other Java features, such as threads, can achieve the higher-level producer/consumer functionality. For example, it is possible to do simultaneous blocking I/O calls by creating two consumers instead of one, or a more complicated client may create both a consumer and a producer.

4 Research Outline

The key areas, which have been researched while producing the core of jGMA are:

- Naming and addressing issues in distributed systems,
- Writing efficient high performance Java.

The core of the jGMA framework is functional, solving the remaining implementation issues will either improve stability or add extra functionality to make the system easier to use. The focus of the research is now the registry component, which needs to be investigated thoroughly.

5 Research Timetable (Future Research Directions)

The topics briefly discussed in this section will be investigated during the remainder of this PhD project.

5.1 Soft-state leasing

As described in the design and implementation Section 3.3.3, there is a need to detect when a PC servlet is no longer working, as any producers or consumers using that servlet will no longer be contactable. The proposed method to overcome this is to use a form of leasing, similar in concept to that used in Jini [20], between the PC servlet and the Registry (over the wide-area). The requirements for the leasing mechanism

will change when the design of the virtual registry is further studied. The development of the leasing system will be left until the implementation issues of the registry are better understood.

5.2 jGMA monitoring

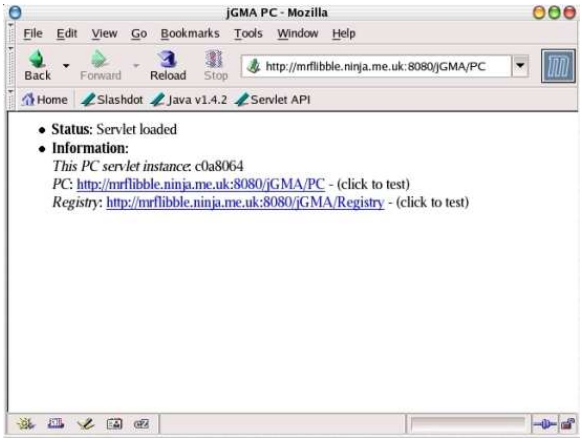

Figure 3: The Happy jGMA Web Page

The PC and registry servlets already provide a limited interface for debugging jGMA. This was inspired (see Figure 3) by the Happy AXIS page. While this helps to check that jGMA is installed correctly, tools are needed to allow jGMA to be monitored while it is running. Currently the developer must watch and interpret one log file for each PC servlet to understand what is happening within jGMA; obviously as the number of PC servlets increases, understanding the log files becomes harder. jGMA Monitoring and Debugging tools will be further investigated during the project.

5.3 The Virtual Registry (VR)

jGMA requires a robust distributed registry. We discuss the two main areas that will be studied to understand the underlying design issues in the following sections. The jGMA system was designed to run with minimal configuration needs and tries to be as flexible as possible in terms of functionality; the VR should follow this design objective, which means its design should add little complexity to the overall installation or configuration of jGMA.

VR requirements:

- Be scalable,
- Store sufficient information to be GMA compliant,
- Be secure, and prevent unauthorised access to the data,
- Require a minimum amount of configuration.
- Have no single point of failure,
- Be robust and tolerant of poor network access,
- Be optimised to return search results as quickly as possible,
- Have persistent storage for the registry back-end.

To make jGMA as easy to install and use as possible the software should function as a standalone system. This is especially useful where there is no permanent Internet connection. One way to achieve this would be to include part of the registry (possibly all of it) with the PC servlet with the standard jGMA software distribution.

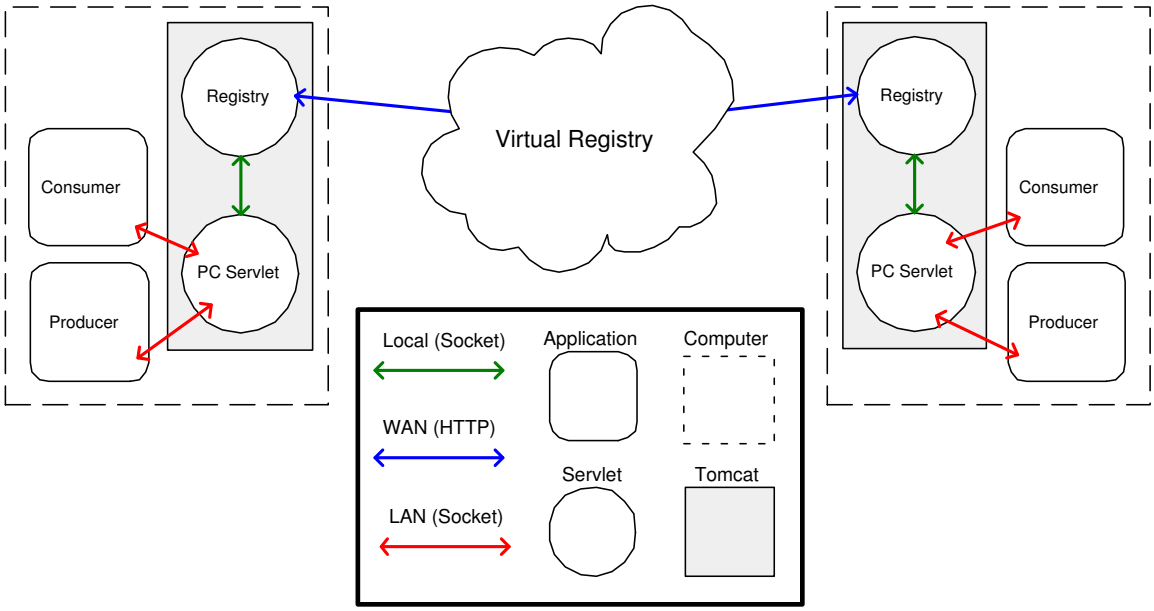

Figure 4: The revised jGMA architecture, here each PC servlet has its own registry component

5.3.1 The challenges for the proposed architecture

There are two main issues to investigate before the VR can be designed and implemented. Since a centralised registry is not scalable, more than one registry will be required, these registry components will need a way to discover each other before they can communicate. The topology of the registry communication will need to be optimised to efficiently use the VR.

Boot Strapping

In the initial implementation of jGMA, a single registry was used to store registration information. That is each PC servlet had the address of the registry hardwired into it. Without this hardwiring, it needs to discover the remote registry, which is a well known boot strapping issue. Other systems have addressed the problem in different ways; common approaches are manually hardwiring remote addresses or using an address caching service. This area will be investigated over the coming months.

The registry communication topology

The second issue when designing the registry infrastructure is to select an optimum topology for jGMA registries to enable efficient query routing. A query must be processed by a VR in an efficient way. In the initial implementation of jGMA, every PC servlet was connected to the one centralised registry. This meant that there was no partitioning of the registration information and a single query of the registry would search every registered producer and consumer. With a distributed VR queries must reach all of the registries necessary to perform the search or some consumers and producers will not be found.

Although caching search information will affect the behaviour of the system, currently we focus on communication between components of the VR. We aim to minimize the number of hops a query must travel to search the entire registry in order to reduce the time it takes to complete a search (minimise latency). Another requirement of the topology is that it must be self-healing, by this we mean that should one or more of the registries fail; it should have a minimal affect on the overall performance of the VR. Possible topologies for jGMA communications are discussed more thoroughly in the First Year Report [15].

5.4 Security

The information jGMA transports may be of a sensitive nature. There is a need to control access to it and stop it from being intercepted and read, especially when it traverses the Internet. Some jGMA design decisions already help the security of the system, for instance the two-tier addressing prevents information about the layout of the LAN from leaking into the WAN layer of jGMA; this is described in Section 3. Currently HTTP is used for the WAN communication of jGMA, moving to the industry standard HTTPS [17] would prevent the messages from being read or altered on route.

Access control is required to restrict which producers and consumers can communicate with each other. There are various grid standards for access control and security one of which is Grid Security Infrastructure (GSI) [18], which was created as part of the Globus project. GSI seems a likely candidate for deployment in jGMA because of its widespread acceptance by grid administrators. This will be further investigated.

5.5 The Blocking API

As described in the First Year Report [15], the initial implementation of jGMA had a combined blocking and non-blocking API, later the blocking functionality was removed to create a simpler overall system. It was proposed that a blocking API be provided as a new layer between the client and the jGMA non-blocking API. The new blocking API will be implemented after more important work, such as the VR, is completed.

5.6 Integration into GridRM

When jGMA used a combined blocking and non-blocking API it was tested with GridRM across several remote sites. After the jGMA API was changed, GridRM could have been altered to use the new API; however, because of the way GridRM was programmed it is easier to integrate it again after the updated blocking layer has been completed.

5.7 A Grid Gaming Framework

Online distributed gaming has become increasingly popular with the widespread uptake of broadband Internet access. Games publishers have each tried to provide an infrastructure to support their online games. There is an opportunity to develop a standard set of services, based on a completed jGMA implementation, to support these games.

A potential set of services that online games may require includes:

- Searching: It is common to have large numbers of servers all running separate games. The user client requires a service to index these games and allow searches based on user-defined parameters such as the current map being played.
- Monitoring servers: Detecting problems with game servers such as large numbers of players can allow the system to load balance or fail over. There needs to be some infrastructure to do this reporting.
- Single sign-on. Many games track individual players to combat piracy. Providing a unique identity for each player makes global high score tables and other statistical tracking possible.
- Security: Authentication and authorisation have become serious problems in recent years. The anti piracy mechanisms such as CD keys have been simple for malicious users to circumnavigate. This creates problems for legitimate users who are barred from the game network once their CD key has been copied.

After completion of the jGMA framework we intend to investigate the issues required to provid an infrastructure for distributed online gaming using jGMA as a communications layer.

6 Formal Training

6.1 Research Training

- Conference papers reviewed and discussed:
  - Cluster 2003, http://www.cs.hku.hk/cluster2003/
  - Grid 2003, http://www.gridcomputing.org/grid2003/
  - KGGI WI Workshop, http://www.comp.hkbu.edu.hk/ william/KGGI03/
  - AIMS 2004, http://w5.cs.uni-sb.de/ baus/aims04/
  - DAPSYS 2004, http://www.lpds.sztaki.hu/dapsys/
  - ASTC 2004, http://www.astc.org/conference/
  - GCC 2004, http://grid.hust.edu.cn/gcc2004/
  - CCGrid 2004, http://www-fp.mcs.anl.gov/ccgrid2004/
  - Grid 2004, http://www.gridbus.org/grid2004/
  - ADIS 2004
  - Cluster 2004, http://grail.sdsc.edu/cluster2004/
  - ISPA 2004, http://www.comp.polyu.edu.hk/ISPA04/
- A general literature survey and review of related research material.
- Took part in the e-Science OGSA Testbed project and its quarterly meetings http://dsg.port.ac.uk/projects/ogsa-testbed/. Helped develop the project web site which received a bronze award at the 2004 UK e-Science All Hands Meeting.
- Deployed multiple versions of the Globus toolkit, and other grid-based software.
- Regular discussions with members of the DSG on research being undertaken.

6.2 Publications / Talks / Networking

Papers

- jGMA: A lightweight implementation of the Grid Monitoring Architecture published in the proceedings of the UK e-Science Programme All Hands Meeting 2004 for the Grid Performability Modelling and Measurement mini-workshop, Sept 01-03 2004, http://dsg.port.ac.uk/ mjeg/jGMA/jgma_ahm2004.pdf
- jGMA: A lightweight implementation of the Grid Monitoring Architecture, technical report September 2004, http://dsg.port.ac.uk/ mjeg/jGMA/jgma_report2004.pdf
- jGMA: A lightweight implementation of the Grid Monitoring Architecture, published in the proceedings of the UKUUG LISA/Winter Conference, February 2004, http://dsg.port.ac.uk/ mjeg/jGMA/jgma_ukuug2004.pdf

Events/Meetings Attended

- Attended UK e-Science Programme All Hands Meeting 2004, http://www.allhands.org.uk/
- Attended UKUUG Winter conference, http://www.ukuug.org/events/winter2004/
- Attended OGSA Testbed meetings

Talks/Presentations

- jGMA at UK e-Science Programme All Hands Meeting 2004, http://www.nesc.ac.uk/events/ahm2004/presentations/71.ppt
- Invited talk on Grid Middleware at UKEA JET, http://dsg.port.ac.uk/ mjeg/jGMA/jgma_jet2004.ppt
- jGMA presented at UKUUG Winter conference, http://dsg.port.ac.uk/ mjeg/jGMA/jgma_ukuug2004.ppt
- Various DSG seminar talks

Mailing lists

- Globus discuss
- R-GMA datagrid project

7 Conclusion

In this report we have described and then discussed jGMA, a reference GMA implementation written in Java. We were motivated to produce jGMA by the lack of a viable alternative to use with our grid monitoring system GridRM, and the ample evidence that such low-level middleware was generally needed for a range of other Grid services and applications.

jGMA, whilst being functional, is at an early stage of development, there are a number of outstanding implementation issues to solve (described in Section 3) and a great deal of further study is required before the Virtual Registry can be implemented. Although jGMA is still evolving it has been made available [21] as a binary release to developers interested in investigating and further enhancing its capabilities.

References

[1] GridRM, http://gridrm.org/

[2] Global Grid Forum, http://www.ggf.org

[3] GMA, http://www-didc.lbl.gov/GGF-PERF/GMA-WG/

[4] R-GMA, http://www.r-gma.org/

[5] Data Grid, http://www.eu-datagrid.org/

[6] pyGMA, http://www-didc.lbl.gov/pyGMA/

[7] LBNL, http://www-didc.lbl.gov/

[8] Globus MDS, http://www.globus.org/mds/

[9] Globus, http://www.globus.org/

[10] Open Grid Services Architecture, http://www.globus.org/ogsa/

[11] Network Weather Service, http://nws.npaci.edu/NWS/

[12] AutoPilot, http://www-pablo.cs.uiuc.edu/Project/Autopilot/AutopilotOverview.htm

[13] University of Illinois Pablo Research Group, http://www-pablo.cs.ui uc.edu/

[14] Jython, http://www.jython.org/

[15] jGMA: First Year Technical Report, http://dsg.port.ac.uk/ mjeg/jGMA/jgma_report_10-2004.pdf

[16] Xindice, http://xml.apache.org/xindice/

[17] HTTP Over TLS, http://www.faqs.org/rfcs/rfc2818.html

[18] Grid Security Infrastructure (GSI) , http://www-unix.globus.org/toolkit/docs/3.2/gsi/index.html

[19] Apache Tomcat, http://jakarta.apache.org/tomcat/

[20] Jini, http://www.jini.org/

[21] Download jGMA, http://dsg.port.ac.uk/projects/jGMA/software/in dex.php