jGMA: A Reference Implementation of the Grid Monitoring Architecture

Distributed Systems Group, University of Portsmouth

End of First Year Report

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

Mark Baker (mark.baker@computer.org)

Registration Period

01 October 2003 to 30 September 2004

Table of Contents

Chapter 1

Introduction

In this chapter we briefly introduce the concept of the Grid and go on to explain the need for a standards-based approach for resource monitoring.

1.1 General Introduction

In 2001, Foster, Kesselman and Tuecke refined their definition of a Grid to ”coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations” [1]. This latest definition is the one most commonly used today to abstractly define a Grid.

It could be said that 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. 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). Closely related to resource sharing is resource monitoring - an essential component of the Grid for tracking and maintaining the system state. By monitoring we mean:

- A way of gathering data about the state of the Grid resources,
- Processing this data for example filtering and fusing it,
- Transferring it from agents to the clients that use this information.

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

Figure 1.1 shows a physical representation of Grid infrastructure. The Grid is basically made up of the following components:

- A series of networked resources which are capable of acting on behalf of a of a client,
- 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.

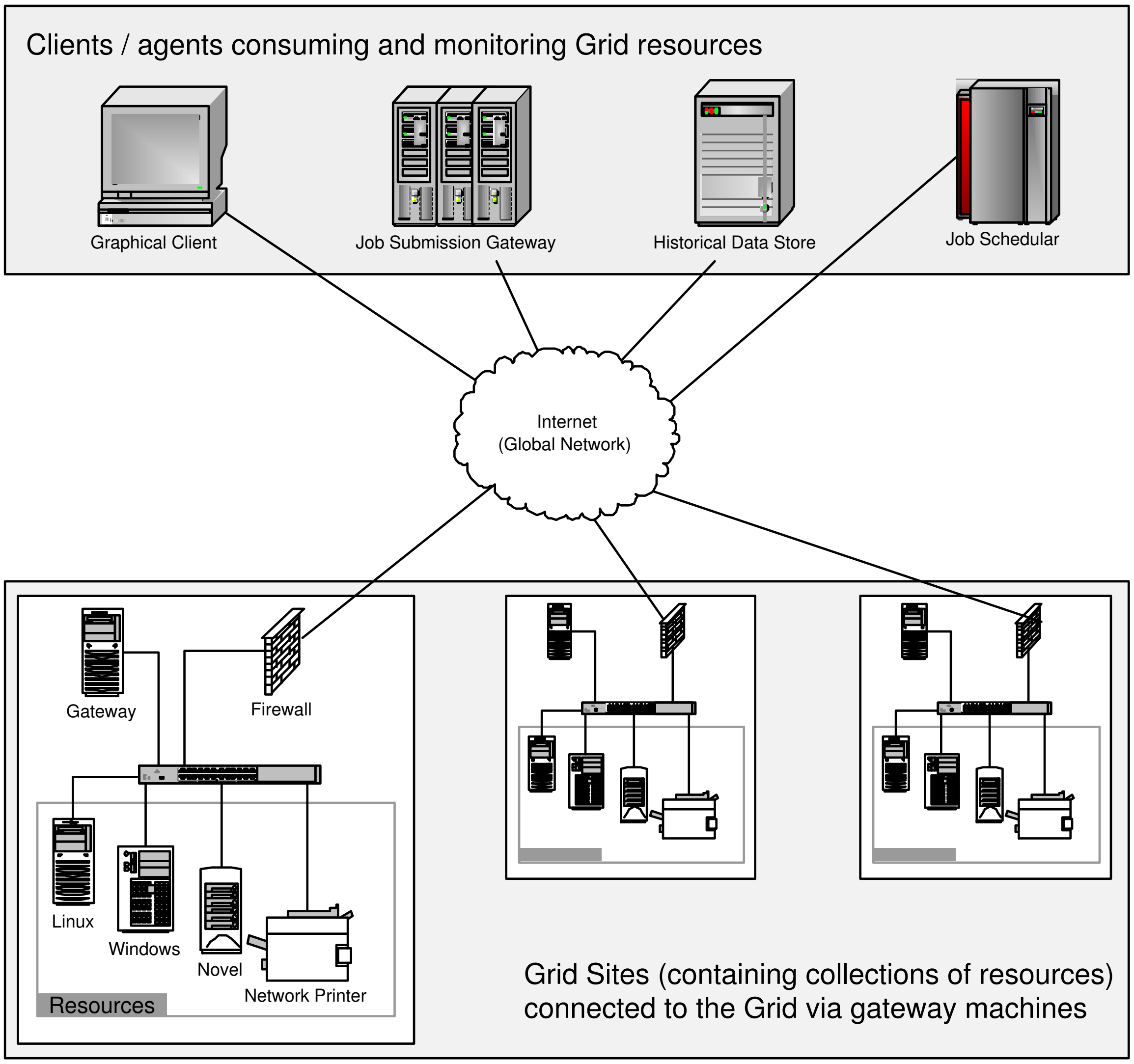


Figure 1.1: A physical view of a Grid

Potentially each of these components, which can be viewed as infrastructure, users, and resources, can generate and acquire data: the components 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 monitoring and controlling the Grid. End users may be interested in monitoring a resource executing 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 may be of no use to the agent. Another service that is needed is one for historical data; an agent may need information on a previous state of the Grid. Examples of information, which may be gathered, are:

- Jobs currently running on a host,
- The capacity or status of a resource, for example, CPU speed, memory, or 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 information 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 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 mixture of different architectures, operating systems, components, and protocols. More importantly, there is no one standard defining how to monitor resources in a Grid environment.

1.2 Motivation

The Distributed Systems Group (DSG) at the University of Portsmouth has for the last few years, been developing a resource monitoring system for the Grid that can gather data from a range of end-points, then filter and fuse this data for subsequent use by a variety of clients. The monitoring system, known as GridRM [2], needs to distribute information over the wide area between, so called, GridRM gateways.

1.3 GMA

The prototype of GridRM uses XML to mark-up events, which are then passed between GridRM gateways over the wide area using HTTP, there is a need for a standards-based approach. We decided to use the Grid Monitoring Architecture (GMA) [3], which is the architecture recommended by Global Grid Forum (GGF) [4] because it identifies and addresses the key issues involved in implementing a monitoring framework for the Grid. The GMA specification sets out the requirements and constraints, which must be addressed for a system to be considered GMA compliant. The GMA is based on a consumer/producer architecture with an integrated system registry, see Figure 1.2.

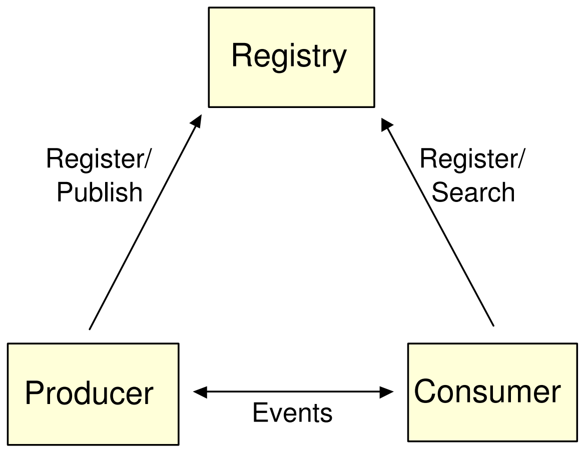


Figure 1.2: The Architectural View of GMA

The GMA is an abstraction of the characteristics required for a scalable monitoring infrastructure for the Grid. The GMA supports a publish/subscribe and query/response model. In this model, producers or consumers that accept connections publish their existence in a directory service (registry). Producers and consumers can then both use the registry to locate parties, which will act as a source or destination for the data they are interested in. It should be noted that monitoring data is sent from a producer to a consumer; however either the producer or consumer may initiate a subscription or query.

The GGF argue [5] that the requirements of GMA cannot be met by existing event-based services, as the data requirements for monitoring information are different compared to other forms of program generated data. The GGF list several desirable features for GMA:

- Low latency,
- Capable of a high data rate,
- Minimal system impact,
- Secure,
- Scalable.

1.4 Summary

In this chapter we have briefly outlined the monitoring requirements of the Grid and briefly defined the needs of GridRM. Having introduced the GGFs GMA, in the next chapter we review the existing GMA implementations, comparing and contrasting their abilities compared to the needs GridRM, and justify why there is a need for another implementation.

In Chapter 3 we will describe the design for a reference implementation of GMA, and then present benchmarks showing the performance and scalability of this initial implementation. Chapter 4 presents a demonstration application used to show off the capabilities and functionality of our implementation. Chapter 5 describes the work that needs undertaken to the current implementation to enhance its abilities to meet our future needs. Finally, in chapter 6 we present our conclusions and describe the research goals and timeline for future work.

Chapter 2

Similar Work

In this chapter 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 the reasons that motivated us to develop jGMA. We split the GMA implementations into two categories:

- **Standalone:** A system providing GMA functionality.
- **Embedded:** A system containing software that exhibits GMA like capabilities.

2.1 Standalone Implementations

2.1.1 R-GMA (Relational Grid Monitoring Architecture)

R-GMA [6] was developed within the European DataGrid Project [7] 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.

R-GMA is the most ambitious and significant variant of the current GMA implementations that was initiated in September 2000. Since then the software has continuously evolved. Currently R-GMA is being used for an ”in-house” testbed [8].

2.1.2 pyGMA (Python GMA)

pyGMA [9] from LBNL [10] 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 2.1: A comparison of GMA implementations

2.2 Embedded GMA Implementations

There are several other systems, which either exhibit GMA like behaviour or have a GMA implementation embedded within them.

The Metadata Discovery Service (MDS) [11] that is part of Globus Toolkit version 3 [12] is based on the emerging Open Grid Services Architecture (OGSA) [13]. 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 complex GT3 software.

The Network Weather Service (NWS) [14] 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 [15] from the University of Illinois Pablo Research Group [16] 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 (version 2).

Table 2.1 lists features and functionally of various implementations of the GMA. As it can be seen R-GMA has great potential, but there are a number of drawbacks, not least of these are the large number of system dependencies required for installation and use. In addition, there are some architecture features, which may limit its scalability and flexibility. Alternatively pyGMA appeared promising, but there are some issues with using it with a Java application, and there are some considerations with regards it having a very simple registry. Finally, MDS3 had the potential to fulfil our requirements for GridRM. Unfortunately the current implementation is embedded in the Globus release, which meant that it would potentially require some reengineering to meet our needs.

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 in their existing form with GridRM. This leaves three alternative GMA implementations, pyGMA, Autopilot and R-GMA.

Calling Python (pyGMA) from Java, which is a requirement of GridRM, is not straightforward. While the Jython project [17] 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 native Java API, and initially it was thought that R-GMA would be a suitable implementation for GridRM. However, there are a number of drawbacks with using R-GMA. It can be seen in Table 1 that there are a significant number of dependencies to build R-GMA from source. Also, R-GMA is aimed at one specific version and distribution of Linux (Redhat 7.3). The developers have used a build process, which relies on files and libraries being in non-standard places and the use of a non-portable mechanism for compilation (shell scripts). There is a binary release of R-GMA, however, this is via RPMs, which again limits the platforms on which the system can be automatically installed. Another problem that was encountered is the rapid development and changing nature of R-GMA. This can create problems for a developer trying to work with such a large code base, because it is constantly evolving to keep up with the latest trends and needs of the large number of developers and potential users.

A requirement of GridRM is that it is easy to install and configure across multiple platforms. A complicated set of prerequisites would make its deployment a lengthy and potentially complex task. GridRM requires a GMA implementation that has a lightweight Java API, which is functional, easy to use, and extensible.

Chapter 3

jGMA Design Criteria

3.1 Introduction

The global layer of GridRM requires a wide-area event-based system for passing control and monitoring information between the local GridRM gateways. Ideally, from our point of view, we would have preferred to integrate a third-party GMA implementation into GridRM; this is for obvious reasons, such as reduced development time and minimal support requirements. However, as stated in Chapter 2, none of the existing GMA implementations met our requirements, and consequently we have developed our own version.

The first steps in our design were to layout a set of general criteria that we considered to be necessary and/or desirable. 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,
- Designed to work locally over a LAN or over a wide-area such as the Internet,
- Fast, and have a minimal impact on its hosts,
- Scalable,
- Choice of registry service, from a lightweight one, such as text-based files, to an XML-based one, for example Xindice [18], or something else, such as a relational database or Globus MDS,

- Able to work through firewalls,
- Capable of taking advantage of TLS and/or the GSI.

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, related technologies, as well as providing portability via bytecode that should execute on any compliant Java Virtual Machine.

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 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,
- Consumer,
- Producer.

Communication between components uses a shared code base, which provides:

- Wide-area (WAN) communication between remote components,
- Local (LAN) communication between local 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).

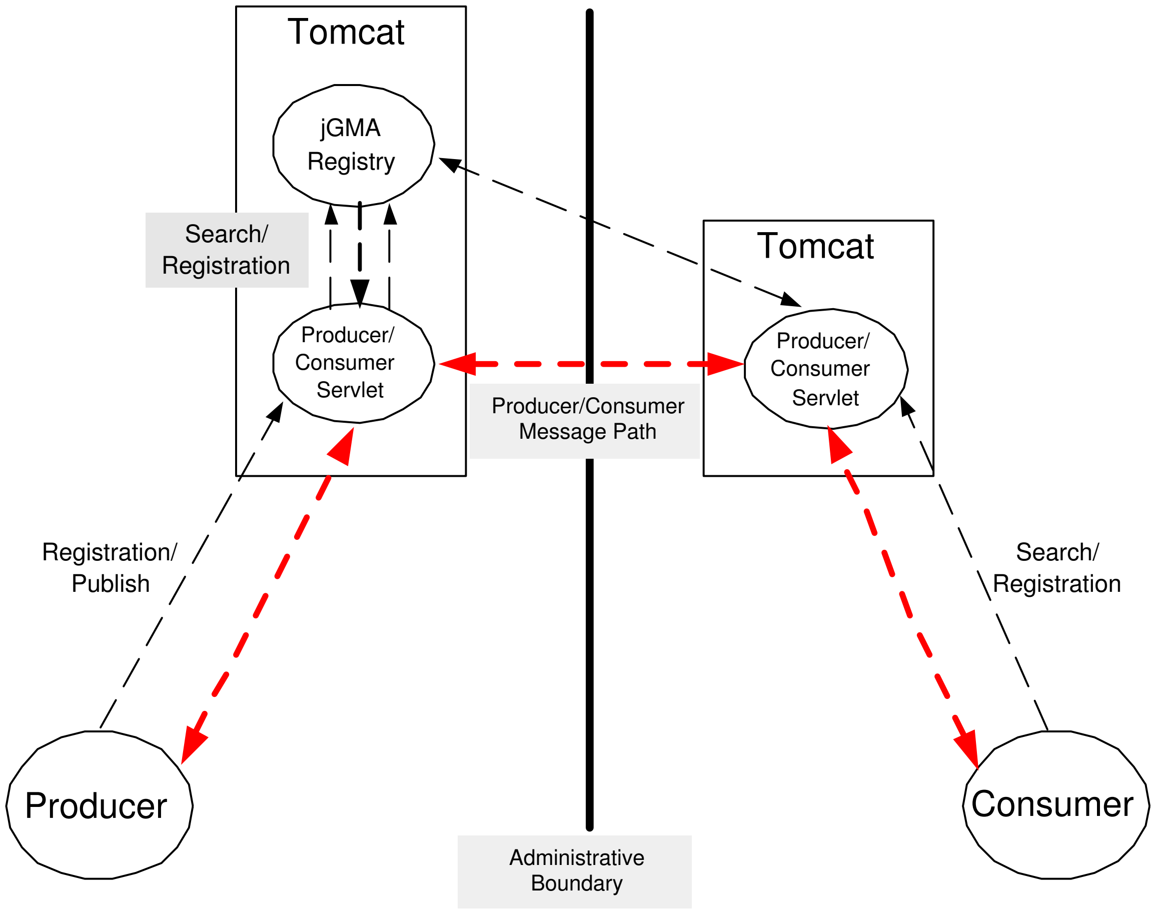


Figure 3.1: The jGMA Architecture

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, which is common to the nodes in standard Beowulf cluster [**?**].

Figure 3.1 shows the jGMA components being used together to provide a wide-area message-passing framework. In this example 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.

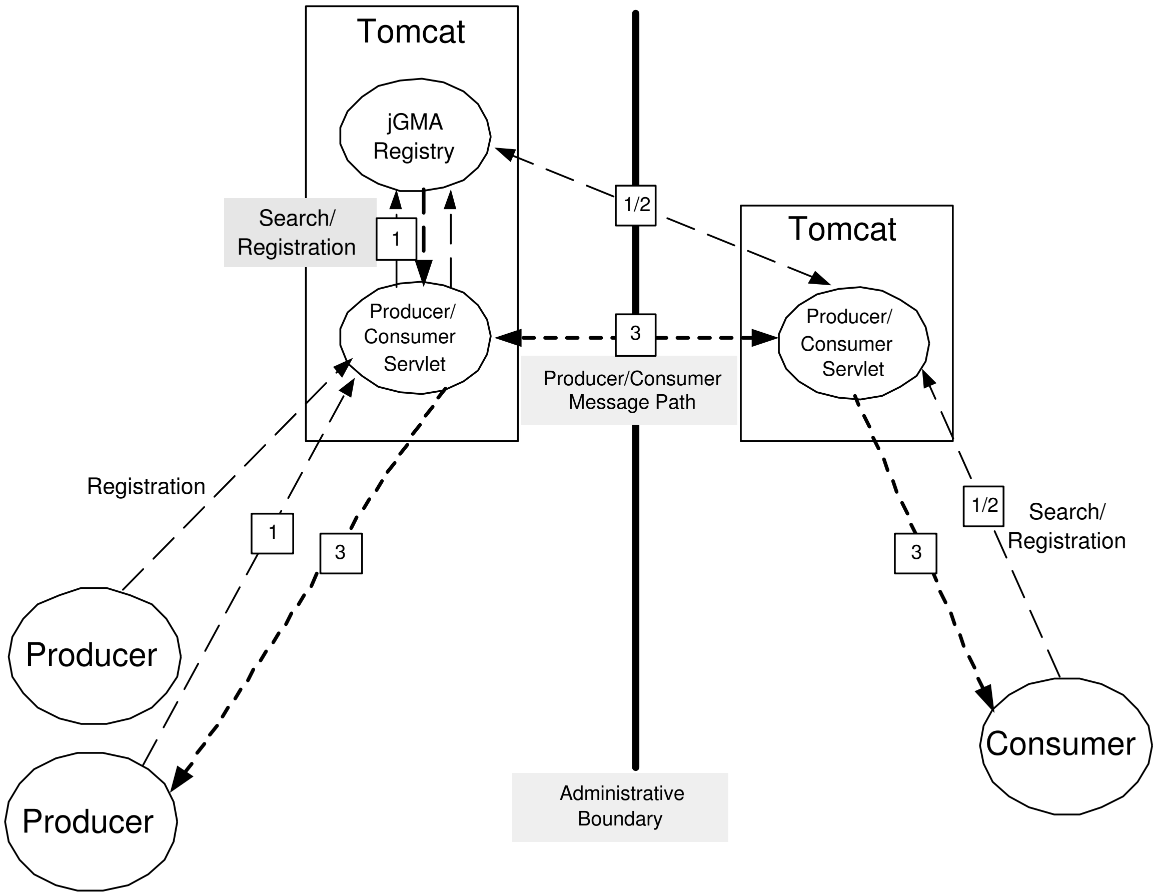


Figure 3.2: Wide-area jGMA Communications

3.3 Example WAN Communications

Figure 3.2 illustrates the steps that occur when a jGMA consumer attempts to interact with one or more producers.

- [1] The producer and consumer register with the registry:
  - The producer/consumer create a connection to the local PC servlet,
  - The producer/consumer sends the PC servlet a human readable name and a description of the its capabilities or interests (marked up in XML),
  - The PC servlet creates a WAN address, from which the producer/consumer can be accessed remotely,
  - The PC servlet sends the capabilities/interests of the producer/consumer and the WAN address to the registry,
  - The PC servlet sends a ’RegistrationComplete’ event to the producer/consumer.

- [2] The consumer queries the registry:
  - The consumer sends an SQL-like query to the PC servlet,
  - The PC servlet forwards this query to the registry,
  - The Registry replies with a list of matching producers (marked up in XML),
  - The PC servlet sends a 'RegistryMessage' to the consumer.
  - The PC servlet sends a 'registration complete' event to the consumer.
- [3] Consumer/producer communications:
  - The consumer parses the registry response and selects a producer to communicate with,
  - The consumer sends a message via a socket connection to the PC servlet,
  - The PC servlet sends the message to the remote network via HTTP,
  - The remote PC servlet, on the producer network, sends the message to the producer via a socket connection,
  - The producer sends a reply to its local PC servlet,
  - The producers PC servlet sends the reply via HTTP back to the consumers PC servlet,
  - The consumer PC servlet sends the reply back via a socket to the consumer.

3.4 The initial jGMA Implementation

While conceptually the producer, consumer and Producer/Consumer servlet are different; the jGMA implementation reuses the same code-base for each. This is possible because although they have different logic for processing jGMA messages a large proportion of their functionality is focused on exchanging messages (events).

Figure 3.3 shows the internal structure of the producer, consumer, and Producer/Consumer servlet. This figure attempts to highlight the importance of the socket `send()` and `receive()` methods. If these methods were poorly implemented, i.e. exhibited poor performance, it would affect the whole system. Our initial analysis of the program flow showed that the majority of the execution time was spent manipulating, copying and sending the jGMA messages through the system.

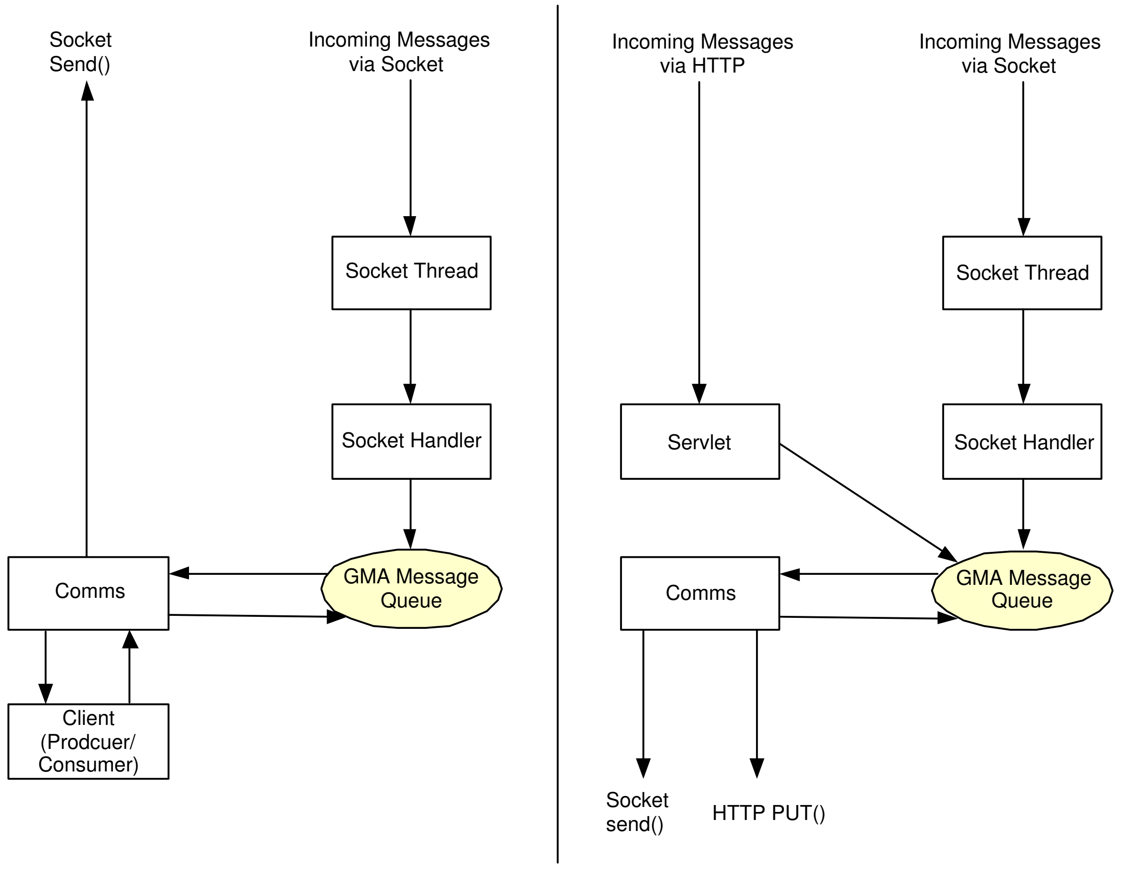


Figure 3.3: The internal structure of a jGMA client and Producer/Consumer servlet

3.4.1 RMI

The first jGMA implementation used bi-directional RMI for communications. The reason for using RMI was that it enabled us to rapidly develop the message passing parts of the initial software. However, it was found that the communication performance was too slow [21].

3.4.2 Sockets (LAN)

In order to speed up the sending of messages Java sockets were used in place of RMI. This gave better control of the send and receives methods, which allowed further optimisations, these are described later in Section 3.4.4. Java NIO [22] may be considered for this layer in the future.

3.4.3 HTTP (WAN)

The current GMA specification does not define the message format and protocol for WAN communication. jGMA uses HTTP [23] for its WAN communication. We do not have to ASCII encode binary messages before sending them because the HTTP protocol allows binary data to be sent in the body of a HTTP POST request to permit uploading through HTML forms [24]. Using 'multipart/form-data' and 'multipart/mixed' it is possible to send more than one binary message in one request, or mix ASCII and binary in one request. If the GGF defines a message format for WAN communication in the future, jGMA can easily send GGF compliant messages. Thus, using HTTP POST minimises the effort required to adapt jGMA to comply with a GGF standard message format later. Currently, jGMA only sends one message per request, but it may take advantage of the HTTP's ability to send multiple messages in future versions.

3.4.4 Objects

In order to reduce message latency we needed an efficient means of passing data between jGMA components. The normal Java programming practice of using objects was replaced with static methods, which manipulated byte arrays. Our objective here was to reduce the number of times Java copied the internal data structures and limit the use of expensive calls to the high-level Java API. The flow of data using the byte arrays method is shown in Figure 3.4.

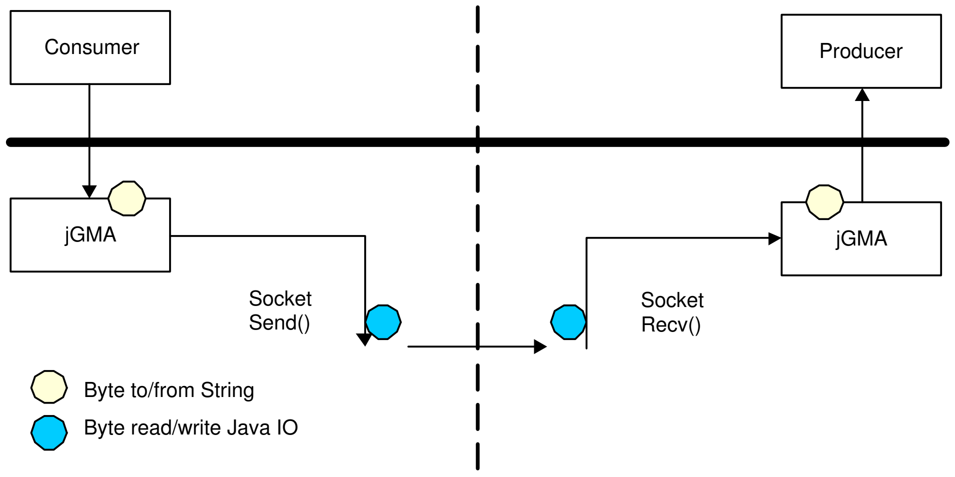


Figure 3.4: Minimising the number of copies

3.4.5 Naming

The initial version of jGMA used a ”pseudo unique” naming scheme to identify consumers and producers. Here the address was made up of several components used for routing messages. While there was no guarantee that the name would be unique, the naming scheme made it unlikely that there will be a clash.

3.4.6 The Registry

The overall purpose of the registry in GMA is to match consumers with one or more producers (or visa versa). This is achieved by producers (or consumers) publishing information about themselves and then consumers (or producers) searching through the registry until they find the relevant match and thereafter the two communicate directly with each other.

The information published in the registry typically includes the unique address of a producer or consumer, and potentially their attributes and capabilities. In addition, to limit the retention of stale information, some sort of Time To Live (TTL) tag should be associated with the registration.

An implementation of the GMA should be capable of scaling to global proportions. This implies that there should multiple registries and the registry information should be replicated for fault tolerance purposes.

GridRM uses jGMA to provide a messaging infrastructure between its gateways. GridRM gateways hold detailed information about its producers and what data they can provide. A GridRM client could search through one or more gateways for the producers that it is interested in. However, as the number of producers and gateways becomes large this would produce an unacceptable load on the overall system and also means that a query could take a significant amount of time. It is clear that a meta-level registry service is necessary. Such a service would hold high-level information about producers and gateways that could be interrogated first by a client before doing a low-level and detailed search on individual gateways.

jGMA registries can provide this meta-level registry service for GridRM. Ideally, jGMA registries should be able to ”slot” together to form a virtual registry. Such a registry, from a client’s point of view, would appear as one large shared entity. To create the virtual registry requires that the physical registries are distributed and the information they hold is replicated.

jGMA originally used a volatile registry, which stored the names of producers and consumers in memory. This registry was designed to provide the limited functionality needed to build and test the original jGMA implementation. The jGMA registry contains a parser that allows queries based on a simplified SQL syntax. Maintaining a high-level of abstraction via the registry API (and SQL syntax) has made it possible to create a jGMA registry interface that can plug-in to a variety of potential persistent repositories, including XML and relational databases, as well as other systems such as the Globus MDS and R-GMA.

Originally, producers or consumers in jGMA registered just their address in the registry. The revised registry API not only permits consumers or producers to register, but it also allows an associated XML document be uploaded, which describes features and capabilities of the registered entities. This additional feature means that the developers using jGMA can publish as much information as they wish, and consequently have control over the granularity of the virtual registry service.

For the purposes of GridRM, the jGMA registry service will remain minimal, we do not intend to store anything other than the information required to provide the GMA-like functionality and use of the extra XML information will be limited.

The objective of this phase of jGMA has been to produce an abstract registry API that can interact with a number of persistent data stores, include relational and XML databases, or a simple flat ASCII file. The current jGMA registry service has been prototyped using volatile storage. Later we plan to test Berkeley DB Java Edition [25] and an ASCII text file as registry components.

3.5 The jGMA First Release

3.5.1 Benchmarks

This section reports on the initial testing the performance and functionality of jGMA over the local area. The overall aim of this stage was to optimise the communication overheads, assess its impact, and confirm overall functionally. We provide an overview of the tests with some key results highlighted, a more thorough analysis can be found in a DSG technical report [27].

A Java implementation of the traditional Ping-Pong network test was used to measure point-to-point performance between a producer and consumer. We were careful to omit extra overheads, such as internal processing of jGMA messages. By comparing the measurements taken when timing jGMA, to the raw performance, it was possible to analyse performance overheads.

Test 1: Non-blocking I/O - A Ping-Pong between a single producer and consumer. This test involved executing both consumer and producer on the same host, and then with the producer and consumers on different hosts connected via Fast Ethernet.

Test 2: Blocking I/O - A Ping-Pong between a single producer and consumer. These tests were executed both over the network and on the same machine in the same way as the non-blocking tests.

Test 3: Scalability Tests - The cluster's head node runs the Producer/Consumer servlets, Registry servlet, and one consumer. The producers are distributed over the cluster so that each node can run up to two. N producers are started and the consumer instructs them to send messages to it over Fast Ethernet, the number of messages the consumer handles per second is recorded. Non-blocking I/O was used as this was felt a more realistic simulation than blocking I/O for large numbers of events.

Test 4: Wide-Area Communications - A wide-area environment was simulated on the DSG cluster by running two Producer/Consumer servlets - one for the consumer and one for the producer. The test machines communicate via Fast Ethernet. The test measures the latency and bandwidth of sending jGMA messages over HTTP via a Producer/Consumer servlet for a range of message sizes. For an explanation of the steps in wide-area communications see Section 3.3 and Figure 3.2.

3.5.2 Results

The performance tests (for details see [27]) showed that for blocking communications there is an extra 8-millisecond overhead compared to raw sockets for Ethernet messages under $<$ 256 Kbytes. This overhead is due to processing a blocking message, which we are continuing to investigate. There is the possibility of changing to an eager-reader paradigm here, but this may produce an excessive impact on the host. The overhead currently limits the peak bandwidth, which is 33% of the raw socket bandwidth. For non-blocking communication using messages $<$ 256 Kbytes, jGMA produces an overhead of 1.4 milliseconds compared to raw sockets for Ethernet, and the peak bandwidth is 67% of the raw socket performance.

It requires between 7 to 9 producers (depending on the message size) to saturate a jGMA consumer. When more producers are added, the number of messages the consumer can handle does not fall - which indicates that even under a heavy load (1100 x 32 Kbytes messages per second) jGMA is stable. When the number of messages being received by a consumer reaches its peak, new messages begin to queue up in the send buffers of the producers, eventually a producer will consume all available resources and will not be able to add any new messages to its send buffer. It is unlikely that GridRM will generate messages at this rate, but it indicates the throughput that GridRM can expect jGMA to be able to handle reliably.

It became evident that some kind of throttling was needed for the sending mechanisms of jGMA. Currently a producer can generate as many messages as the memory allocated to it by the Java VM can contain. This was desirable when testing the scalability because it allowed stress testing of the jGMA implementation. This, however, does have a side effect, a consumer can only process a certain number of messages per second. When multiple producers are sending at the same time, the consumer will reach a maximum throughput and messages will start to accumulate in the send buffers at each producer.

3.6 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.

3.6.1 Event Driven API

jGMA follows the event-driven programming paradigm since it is not known when a message will be generated. This is similar to the way a typical GUI operates, where little happens until the user interacts with the GUI. In jGMA the program only executes background house keeping tasks, such as cache flushing, until a producer or consumer generates an event. This creates a problem for software, which tries to control the flow of execution (waiting for a message reply).

In initial version of jGMA an attempt was made to provide a combined event driven and traditional blocking API, this created complex software, which was hard for the developer to debug. A neater solution is proposed which places an optional blocking wrapper layer around a stand-alone event driven API, illustrated in Figure 3.5. This allows developers to integrate jGMA with programs, which expect to control the flow of messages while allowing access to a simple non-blocking event API for event driven programming.

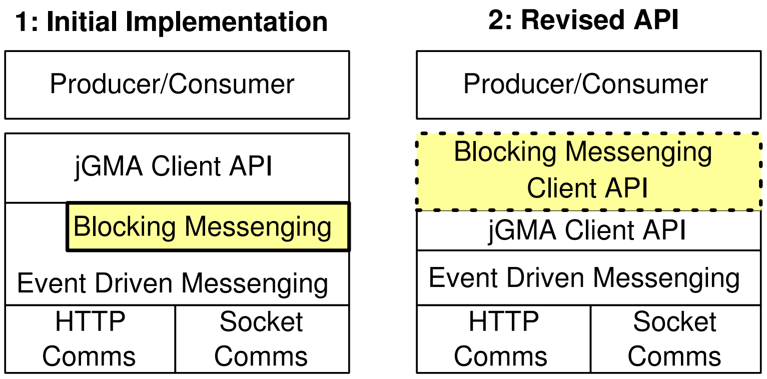


Figure 3.5: Revisions to the jGMA API

3.6.2 Revised Naming

The initial addressing and naming scheme (see section 3.4.5) in jGMA was based on implementation decisions rather than design, as the API matured parts of the old naming system became redundant so a new scheme was designed from scratch.

Addressing Requirements of jGMA

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 for instance. 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. The PC servlet is acting a multiplexer and de-multiplexer for jGMA communications over the wide area.

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.

LAN Addressing

When a PC servlet or a client is communicating to a client directly over the LAN it needs to know either the hostname or IP address and the socket number of a client. It is possible to have more than one machine on a LAN with the same fully qualified domain name and it is also possible to have more than one machine with the same IP address on a LAN (in some non-standard configurations). It is less likely that there will be machines sharing IP addresses than having the same hostname, so for jGMA the assumption is made that the IP is unique on the LAN, this is a compromise between complexity and functionality. Since it is not possible to have more than one program bound to a TCP socket the combination of IP address and socket number is unique on the LAN.

Information required to contact a client on the same LAN:

- IP
- Socket Port Number

(i.e. 192.168.0.100:5678)

Additional Naming Requirements

The combination of a PC servlet URL, an IP address and a port number provides enough information to create a globally unique address for each producer or consumer. However, it is possible to embed more information into a name, than just the address, such as some human readable information to help developers map a jGMA name to a client they are working on. Since this information is sent with every message this extra information represents an overhead for every message, so a balance must be found between useful information while keeping the size of the address as small as possible. Embedding extra information along with the address allows some functionality to be provided by the jGMA API by parsing the name without having to query a registry.

It is useful for the developer to be able to tell from a jGMA name whether a client is a producer or a consumer. The API could also filter consumers and producers by parsing the name, without having to query the registry.

Allowing the developer to use a human readable label in the name allows them to potentially to distinguish between clients without having to query the registry. This makes debugging and understanding the flow of message in jGMA easier since the raw jGMA name contains enough information to associate a name with a end point instance. Again, it would be possible for the jGMA API to make use of this extra information, for example filtering for consumers, which have a specific name.

Combing this information allows us to create a valid URL for LAN addressing within jGMA (i.e. socket://192.168.0.100:5678/consumer/testconsumer).

WAN Addressing

The PC servlet component runs in a Tomcat container, this provides a HTTP URL to access the servlet by. As inter PC servlet communication is done via HTTP this URL is the only information required to initiate a connection to a remote servlet. Creating a hash from the client LAN address and appending it to the servlet URL provides a unique WAN address.

Information required to contact a client over the WAN:

- URL to PC servlet,
- Hash mapping to the LAN address.

(i.e. http://dsg.port.ac.uk:8080/jGMA/PC?234623)

Two Tier Naming Structure

Following this addressing design a client has two addresses: a LAN address used for socket communications and a WAN address used for inter-servlet communications using HTTP(s).

The PC servlets were modified to allow the automatic translation between the LAN and WAN addresses using a hash-map to link the two addresses. 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, are hidden from remote sites, only the URL to the PC servlet and the hash are exposed.

3.6.3 Client and Servlet Liveliness

Should a consumer or producer fail to cleanly un-register themselves (possible with a poorly implemented client or a network outage) a stale registration will be left in the registry. Similarly if a PC servlet is no longer reachable for some reason any consumers and producers behind it become un-reachable, but their information is still 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 signalling 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.

In the test environment the PC servlets and registry were left running most of the time but the clients were constantly being started and stopped (possibly without using the API to signal the registry) so the client to servlet tests, were implemented first as this kept stale records out of the registry during testing. 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 limit 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.

3.7 Summary

The jGMA library has been revised since the first implementation, after the benchmarking 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 use a standard URL format and the two-tier approach to addresses minimises the amount of local network 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.

Proposed future implementation and design work is described in Chapter 5.

Chapter 4

jGMA web-cam demo

There is often a need to demonstrate the jGMA software in an interactive way during presentations; it is also desirable to have a permanent online demonstration, which interested parties can execute later to see how the overall system works. However, jGMA is middleware, it provides services transparently for other software, and there is only a log file to show that jGMA is working, which is not typically very interesting.

Such a demonstration should have the following features:

- Event driven (the user interacts with the GUI which creates jGMA events),
- Accessible via the Internet through a web browser,
- Graphical,
- Show both producers and consumers,
- Demonstrate some of the underlying aspects of what jGMA is doing behind the scenes,
- Have the scope to be extended to show more complicated functionality.

A web-cam browser was chosen as the example application for the demonstration. The demonstration consists of a Web browser displaying web-cam images served from jGMA producers. The images are fetched by a jGMA consumer, which is coupled, to the web interface.

4.1 The Design

4.1.1 The Architecture

An important feature for the web-cam demonstration architecture is that it has to use two different sites. The jGMA PC servlets are executing one these two sites that are connected by the Internet. This set up enables us to show all aspects of jGMA communication (Local, LAN and inter-site WAN). Figure 4.1 shows the schematic layout of the demonstration.

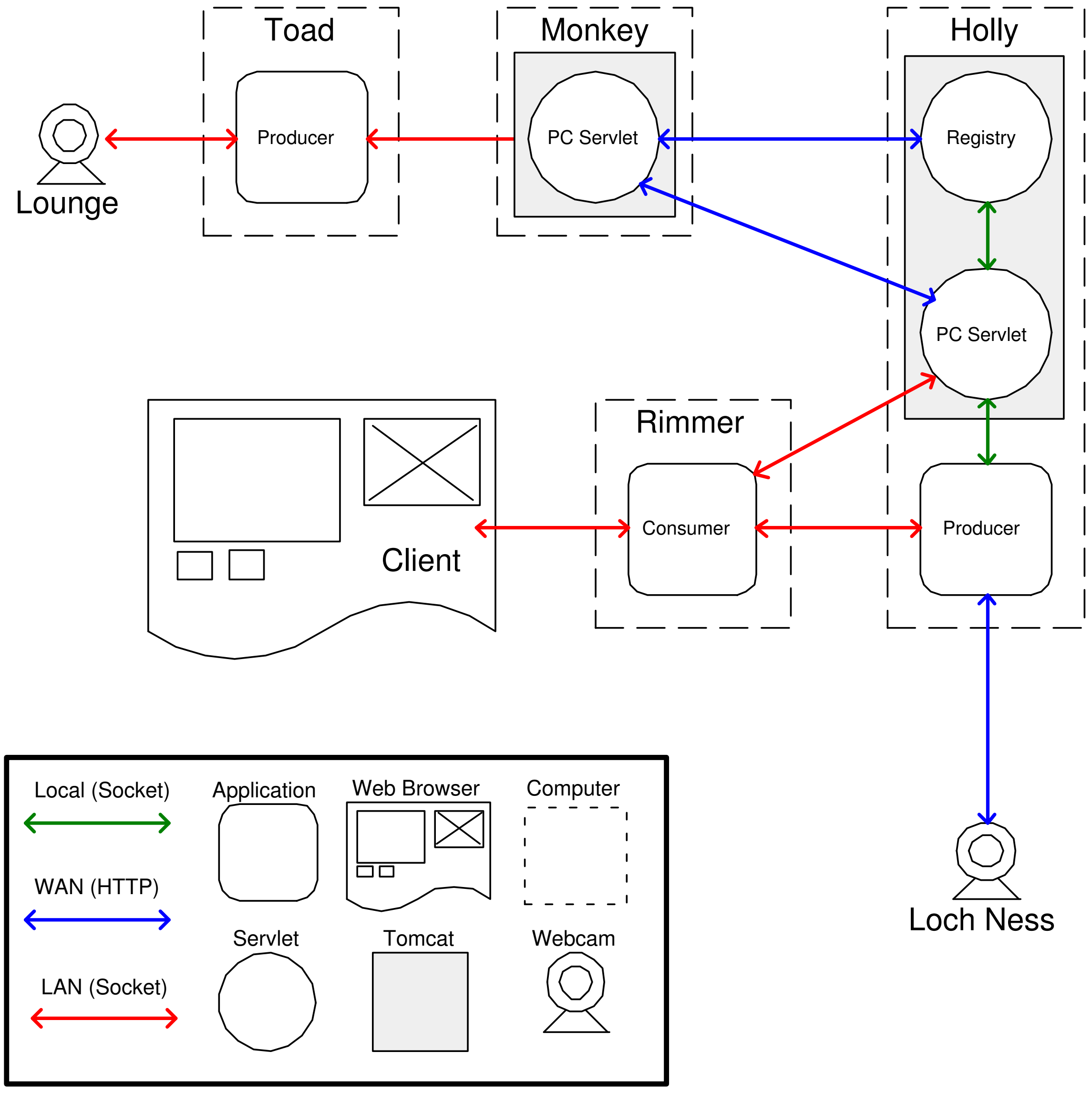


Figure 4.1: A schematic showing the layout of the jGMA web-cam demonstration

The demonstration provides an interactive graphical interface (Web browser) to consumers and producers executing within the jGMA framework. Producers fetch images from web-cams either directly from the local machine or via HTTP. Consumers in turn can discover and retrieve these images using the jGMA framework.

Two simple operations were implemented for the web-cam demonstration: `fetch()` and `pull()`. `fetch()` retrieves an image from a producer and `poll()` retrieve images at predefined intervals. In the future we will add more sophisticated features such as video streaming.

4.1.2 The Web-cam GUI

There are three components that make up the GUI:

1. A box with a list of available web-cams discovered by querying the registry.
2. A box with textual log showing what jGMA is doing behind the scenes.
3. An area where images are displayed.

These components are laid out in Figure 4.2.

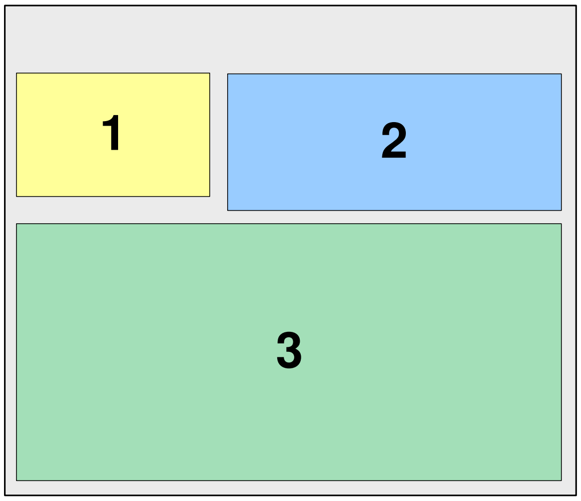


Figure 4.2: The Web browser GUI design

4.1.3 The Web-cam Implementation

As the demonstration interface is within a Web browser there is one key technical difficulty to overcome. More than one component may need to be refreshed at the same time.

The log window (label 2 in Figure 4.2) needs to be updated each time a jGMA operation is performed, i.e. refreshing the web-cam producer list (label 1 in Figure 4.2), and the web-cam image needs to be displayed (label 3 in Figure 4.2) when a producer is selected.

In order to just use the Web browser to deploy the interface, the GUI must be created from HTML widgets, i.e. drop-down combo boxes and buttons and the it must be able to refresh more than one component after a user action. Based on the author's previous experience using HTML to write GUIs it was not known if this was possible. If it could not be achieved then the interface would have to be written as a Java Applet and embedded in the Web browser; this adds the extra requirement of a working JVM to view the demo. Since we felt that it was important to reach as many people as possible, the use of an Applet was not desirable and the HTML implementation was pursued.

User actions from each page element could require one or all of the other elements to be updated. The options for doing this in HTML are:

- Client side Java Script dynamically altering the page elements,
- Using one page and refreshing the whole page when ever an action is performed,
- Splitting the page into four separate components using IFrames (inline frames).

Using the HTML IFrame component, allows for the best control of the page elements, minimising the amount of content, which must be served to update the interface, i.e. only the changed elements are served. Since the most widely used web browsers (Internet Explorer and Mozilla) use different Document Object Models [28], writing a generic Java Script that can work on both browsers is difficult. One solution is to minimise the use of client side scripting so that the demonstration will work in as many browsers as possible.

4.1.4 A jGMA web-cam Producer and Consumer

A jGMA web-cam producer was created that would respond to queries from jGMA consumers. The web-cam producer fetches an image from a URL (provided as a command line parameter) and sends the image as binary information to the consumer. Web-cams were used, which publish their images via HTTP servers as standard JPEGS. This meant that a producer can fetch an image from a geographically remote web-cam, which allows a diverse and interesting set of web-cams be used for the demonstration.

The web-cam consumer provides two functions, which can be invoked by command line parameters. If the consumer is passed a jGMA address, it attempts to send an event to the producer requesting a web-cam image; if the operation is successful, it writes the image to a file and prints the file name to the standard output (`stdout`). If no command line parameters are issued, the consumer queries the registry and prints out a list of web-cam producers.

4.1.5 Writing the Interface

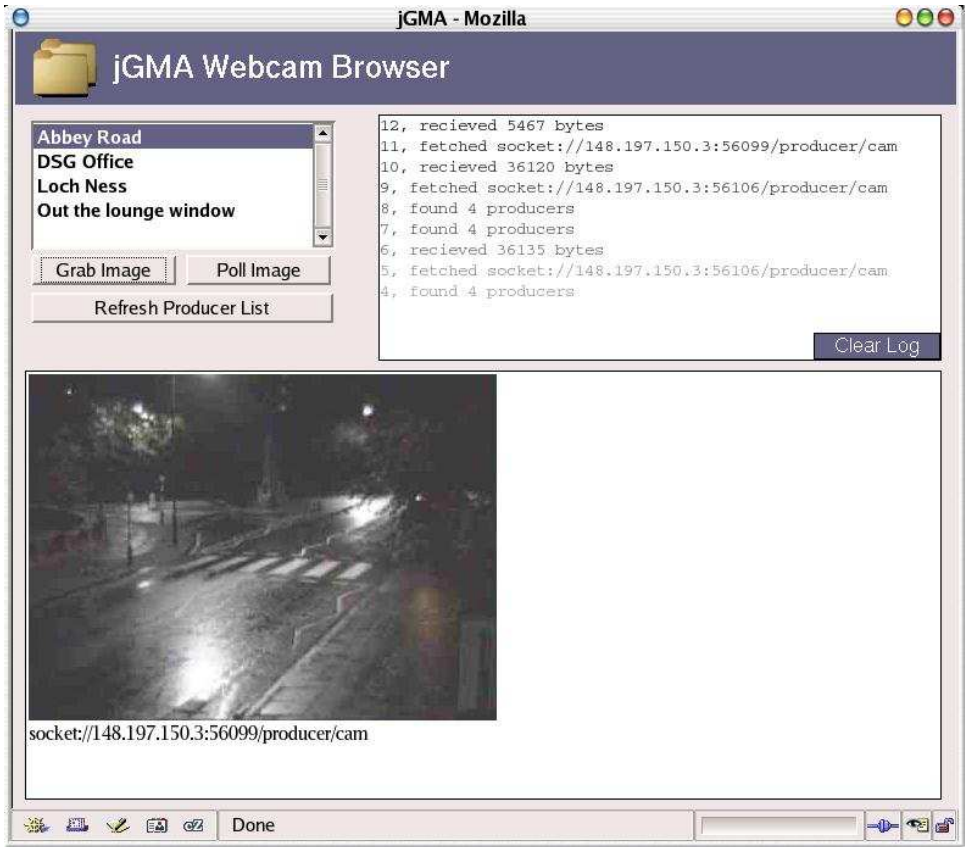
PHP4 was used to implement the server side interface, the language was chosen as it was already installed on the departmental Web Server. It should be noted that any server side language would have worked. The script executes the jGMA consumer and interprets the collected output. HTML and CSS is dynamically written to build the interface with one line of client-side Java Script to allow more than one element to be refreshed if required. For example the log element (label 2 in Figure 4.2) and the web-cam display element (label 3 in Figure 4.2). Every time the script is called an entry is placed in the log file, which provides some insight into some of the actions jGMA is undertaking behind the scenes.

Figure 4.3 shows a screen shot of the demonstration. The three buttons allow a user to retrieve a new image from a web-cam producer, poll a producer for images, i.e. automatically invoke the consumer at a predefined interval to retrieve a new image, and finally the list of producers can by refreshed.

A BASH shell script is used to start the demonstration or publish a new version of the producer to the servers.

4.2 Summary

The web-cam demo has been successfully deployed across two sites using four web-cams. It was demonstrated live at the UK E-Science All Hand Meeting 2004 [29] as part of a presentation on jGMA. The demo meets the requirements described in the introduction to this chapter and allows the jGMA software to be interactively demonstrated in a manor suitable for presentations or viewed over the Internet using any modern web browser.



```
12, recieved 5467 bytes
11, fetched socket://148.197.150.3:56099/producer/cam
10, recieved 36120 bytes
9, fetched socket://148.197.150.3:56106/producer/cam
8, found 4 producers
7, found 4 producers
6, recieved 36135 bytes
5, fetched socket://148.197.150.3:56106/producer/cam
4, found 4 producers
```

socket://148.197.150.3:56099/producer/cam

Figure 4.3: A Screen shot from the running demo

4.2.1 Problems with the Current Implementation

The PC servlet to registry leasing had not been implemented when the demonstration was developed it is possible to leave producers in the registry which could not be interacted with (stale producers). The static nature of the HTML interface means that if a user leaves the browser open for a long period the information it displays may be out of date. For example producers may have been added or removed from the registry which will make the producer list inconsistent.

If the PHP script calls the Java consumer with the address of a producer that is no longer active the PHP script will hang, as the jGMA consumer will never return an image. This requires the demonstration to be restarted. The work around for this is to always refresh the producer list before using the demonstration. Three issues must be addressed to fix this behaviour. When the PC servlet to registry leasing is implemented it will be less likely that there will be stale producers in the registry, as they will be removed if a PC servlet fails to renew its lease. The jGMA consumer needs trap and handle the condition of a producer not sending it an image within a timeout period. And finally the PHP script must provide a finite execution time for any consumer it invokes in order to protect against programming errors in either the jGMA framework or the web-cam producers and consumers.

Unfortunately the final point cannot be implemented using PHP4, as the functions to execute external programs do not include a way to limit the execution time of forked processes. However, PHP5 does have this kind of external process handling. When PHP5 has reached production quality the interface may be ported from PHP4.

4.2.2 Future Work

Apart from the alterations already described, the demonstration could be extended in the future to show more complex GMA behaviour such as:

- Allowing a consumer to subscribe for images from a producer, which could send a new image based on motion detection. This would demonstrate subscribing for events and a producer pushing events to a consumer.
- We could make either video or sound producers to create a larger amount of traffic than single JPEG images. This would demonstrate the robustness of jGMA when working with large events.

Adding video streaming or animating images in a slide show would let the demonstration constantly show something happening rather than just displaying single image. This would be more interesting for an audience as there is more to see, and stress jGMA capabilities further by creating more network traffic.

Chapter 5

Conclusions and Future Work

In this report we have described and then discussed jGMA, a reference GMA implementation written in Java. We were motivated to produce jGMA due 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 other Grid service and application.

jGMA, whilst being functional, is at an early stage of development, there are a number of outstanding implementation issues to solve (described in Chapters 3) and a great deal of further study is required before the Virtual Registry can be implemented. In this report we have presented the results of simple benchmarks that provide us with some insight into the expected performance and capabilities of jGMA, however this will change as we extend and optimise the implementation further. Although jGMA is still evolving it has been made available [30] as a binary release to developers interested in investigating and further enhancing its capabilities.

5.1 Summary of Immediate Research Goals

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.

5.2 Future Research Directions

In this section we focus on potential future research directions of work with jGMA. Chapter 3 described some implementation issues of the current jGMA consumer/producer PC servlet and considers a number of enhancements. Additionally, Chapter 3 also outlined the components, such as PC servlet to registry leasing, that need to be completed. The most interesting of these implementation issues are discussed in more depth in this chapter. There is also the possibility to add debugging and monitoring tools to make the system easier to use by end users. However, the focus for the next stage of research will be to design and implement the jGMA registry. An initial design of the distributed registry is described in Section 5.2.3.

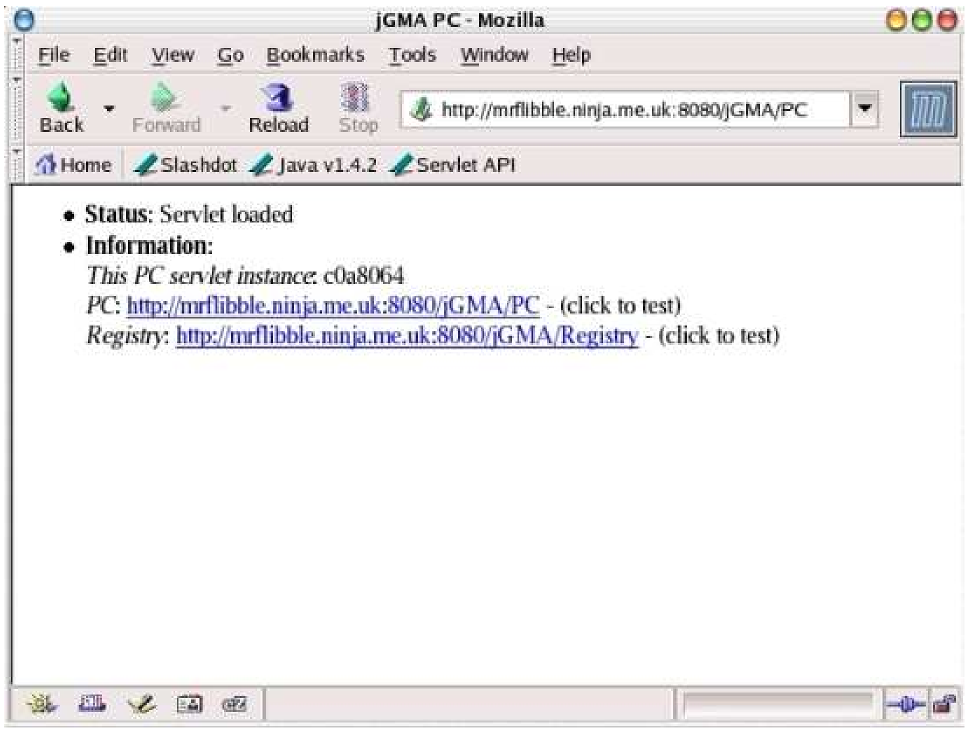
5.2.1 Soft-state PC Servlet to Registry Leasing

As described in the design and implementation Chapter 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 [31]), between the PC servlet and the Registry (over the wide-area). The infrastructure implemented for the ping-pong tests, which tests communications between the PC servlet and Consumers/Producers is not ideal for testing the WAN communications because it typically has a higher latency and lower bandwidth than a LAN, so the number of messages used to test connectivity should be kept to a minimum to make most efficient use of the bandwidth.

Two values must be determined - the length of the lease and how long before (or after) the lease expired before attempts should be made to renew it. A simple solution is to choose values based on the developers previous experience. However, more accurate time values may be chosen through either experimentation or empirical studies.

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.

5.2.2 jGMA Monitoring


jGMA PC - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://mrflibble.ninja.me.uk:8080/jGMA/PC
Home Slashdot Java v1.4.2 Servlet API

- **Status:** Servlet loaded
- **Information:**
  This PC servlet instance: c0a8064
  PC: http://mrflibble.ninja.me.uk:8080/jGMA/PC - (click to test)
  Registry: http://mrflibble.ninja.me.uk:8080/jGMA/Registry - (click to test)

Figure 5.1: The Happy jGMA Web Page

The PC and registry servlets already provide a limited interface for debugging the installation of jGMA. The inspiration for this simple message (see Figure **??**) was the Happy AXIS page, which can be used to validate an installation of the Apache's projects implementation of SOAP [32]. While this helps to check that jGMA is installed correctly, we would also like to provide some tools 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. There is a need for more intuitive tools to help monitor, potentially test and debug a system using jGMA.

We have identified four ways that information could be extracted from an executing jGMA system. These are listed below. The most important issue is to minimise the impact and intrusiveness on jGMA, if the additional tools are being used; these action should not affect the behaviour or performance of the system.

1. Provide an API to access copies of events (messages) as they flow through each PC servlet.

2. Provide tools that can be used by the consumers and producers to pass meta-events through the framework describing what is happening.
3. Standardise the existing jGMA logging to allow an external program to utilise the information.
4. Create tools to trace and visualise the flow of events through jGMA. This would require instrumentation code to be added to the jGMA software at every level of the API

While it is not clear which solution is best to use yet, the log method would be the easiest to implement, it also has the advantage of not requiring the monitoring software to be integrated into jGMA, as the log parser can run as a separate program.

5.2.3 The Virtual Registry (VR)

Chapter 3 describes the current implementation of the jGMA's registry implementation, which is volatile and centralised. jGMA requires a fault tolerant distributed registry to provide a fault tolerant and robust system. We discuss the two main areas that will be studied to understand the underlying design issues in the following sections.

VR System Requirements

The jGMA system was designed to run with minimal configuration needs and tries to be as flexible as possible in terms of functionality; the registry component should follow this design objective. This means that a design requirement for the VR is that it should add as little complexity as possible to the overall installation 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.

5.2.4 Standalone jGMA

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.

Proposed VR Architecture

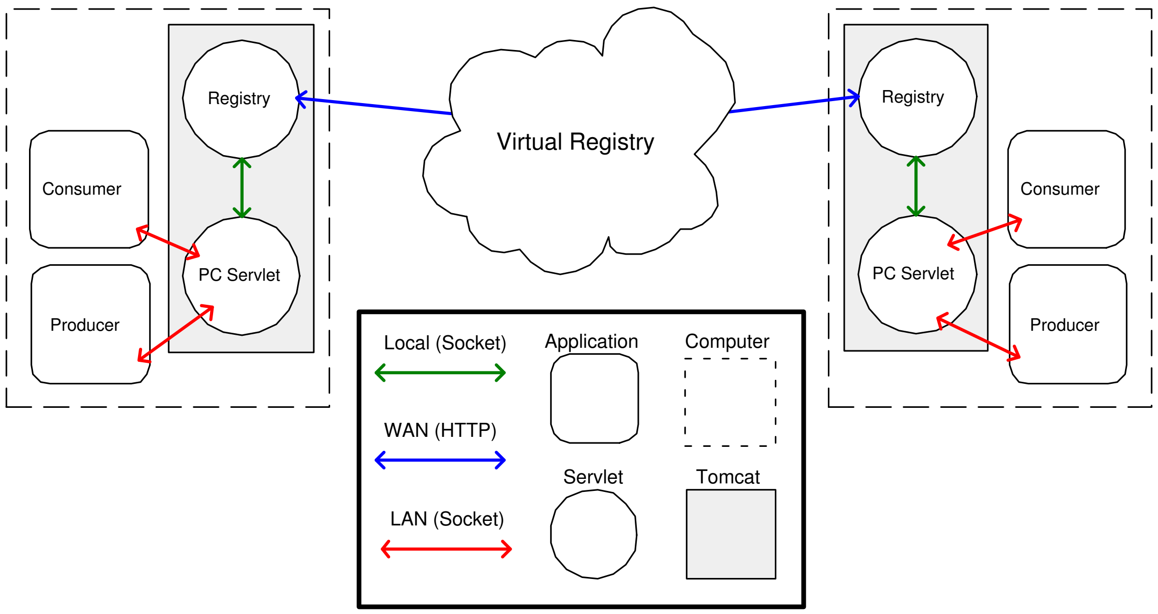


Figure 5.2: The revised jGMA architecture; here each PC servlet has its own executing registry component

5.2.5 The New 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 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 from every site. That is each PC servlet had the address of the registry hard-wired into it. By including a registry component in the PC servlet the name discovery problem becomes an issue for the registry because the PC servlet does not have to discover the registry if it has one included in it but the registry still needs to find a way to join the jGMA network. Other systems have addressed the problem in different ways, two common approaches are manually hardwiring remote addresses in each node or using a number of address caching services.

An example of a centralised hard-wired registry is Jini when used over the wide area. Jini uses multicast packet to discover the lookup service (its registry) when used on a LAN, but when multicast is not available the, a hardwired unicast is used to interact with lookup services.

The Gnutella protocol [33] uses a network of peer services with static addresses, which cache the addresses of end points. Each peer is hardwired with the static addresses of the caching services; when the peer joins the network it fetches a list of entry points by from the caching services. The caching services are replicated to add redundancy to the system. This solution is a practical way to provide nodes with sufficient information to join the system; the current version of GridRM uses the same solution.

These are two examples of how different distributed systems handle the boot strapping problem. This area requires further investigation to understand the existing solutions and find an optimal way to solve this problem in the jGMA VR.

Registry Communication Topology

The second issue when designing the registry infrastructure is to select an optimum topology for jGMA registries for efficient query routing. A query must search the information stored in the 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 it must be self-healing, by this we mean should thief one or more of the registries fail, it should have as minimal affect on the overall performance of the VR.

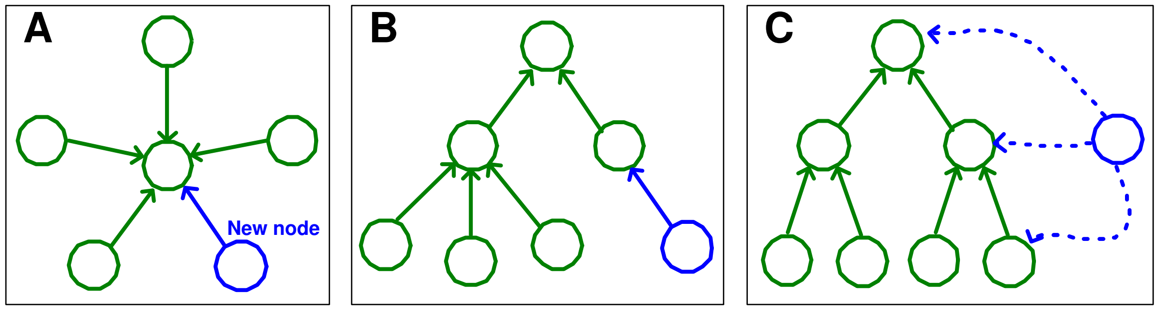


Figure 5.3: A new node joining the system in three different communication topologies.

Figure 5.3 shows three possible node topologies. Each circle (node) could represent a jGMA registry component.

- A) Centralised.
- B) Static tree.
- C) Dynamic tree.

A centralised system (Figure 5.3 A) is not acceptable for jGMA, as it represents a single point of failure and is not scalable. As the number of nodes increases the performance of the network decreases as the central node becomes overloaded. UDDI [34] can be configured to use this topology.

In a static tree topology (Figure 5.3 B) the parent to child relationship is predefined. An example of this is DNS [**?**]. The main problem with this topology is the number of nodes at each level must be predefined in order to create an efficient hierarchy. If we use DNS as an example, if there are too many sub-domains for a given domain the name space becomes difficult to manage as the number of records increases.

In a dynamic tree (Figure 5.3 C) an algorithm determines the position of a new node in the hierarchy. The Gnutella peer-to-peer protocol uses this mechanism to try and create an efficient tree [36]. The topology attempts to arrange itself into an efficient structure at runtime, based on trying to minimise the number of hops required to traverse the tree.

5.2.6 Summary

The two main issues when investigating the VR are how to design a scalable and efficient communications topology. We need to determine the best topology and overcome problems associated with boot strapping and efficient searching. These issues will be the main focus of jGMA development in the near future.

5.3 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 Chapter 3.

5.3.1 Encrypted Communications

Data passing through a medium beyond the physical control of an organisation can be passively intercepted, read without changing the data. The integrity of the data can also be compromised while it is in transit, 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 [37] or RSA [38], does not prevent the interception of data, but stops it from being altered on route and prevents a third party from interpreting the payload.

Currently HTTP is used for the WAN communication of jGMA, moving to the industry standard HTTPS [39] would prevent the messages from being read or altered on route.

5.3.2 Authentication and Access Control

Some kind of access control is required to restrict which producers and consumers can communicate with each other. It is unclear at what layer of jGMA this should be implemented, whether it should be at the client level in the consumers/producers or provided as part of the jGMA API. There are various grid standards for access control and security one of which is Grid Security Infrastructure (GSI) [40], which was created as part of the Globus project. It is desirable to use a standards-based approach for jGMA and GSI seems a likely candidate because of its wide spread acceptance within the area of grid arena.

5.3.3 Security Features of the jGMA Public API

Part of the contract provided by the jGMA API is to prevent itself from being used in ways that it was not designed for. Specifically, care has been taken to implement and ascertain parameters that will not allow the misuse of the API. Moreover, a layer of abstraction between the users of the API and the implementation of the API enforces the order in which the methods can be called. For instance, the API guarantees that a consumer cannot send a message to the registry to unregister a different consumer.

5.4 Blocking API layer

As described in Chapter 3, the initial implementation of jGMA had a combined blocking and non-blocking API, that later had the blocking functionality removed to create a simpler overall system. It was proposed that a blocking API be provided as new layer between the client and the jGMA non-blocking API. This functionality is useful for using jGMA within system, which expect to control the flow of execution. However, it is not essential for the further development of jGMA, since its consumers and producers all follow the event-driven (non-blocking) paradigm. The new blocking API will be implemented after more important work, such as the VR, is completed.

5.5 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.

5.6 A Grid Gaming Framework

Online distributed gaming has become increasingly popular with the wide spread uptake of broadband. 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 statistic 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 provide an infrastructure for distributed online gaming using jGMA as a communications layer.

5.7 Summary

The Virtual Registry (VR) is the most important component yet to be developed for jGMA; it is also the most interesting component from a research perspective.

As the GMA implementation develops and matures the GGF may outline the necessary API that will allow the GMA implementations to interoperate. jGMA has been designed to be as flexible as possible; this will allow modifications for inter-operation with other implementation to be made without major re-engineering.

R-GMA and pyGMA are both being changed to Web Services' technologies. With the widespread adoption of Web Services throughout the community it seems quite likely that it will be used in a formal GMA protocol specification. jGMA has been written to be efficient and fast. Whereas SOAP uses XML, which implies larger and more verbose messages that will create additional overheads, consequently slowing communications. If jGMA where to interact with other GMA implementations via Web Services a separate translation gateway would probably be the efficient way of inter-operating without slowing down jGMA messaging.

In the chapter we have discussed some implementation issues, which need to be addressed to complete the jGMA system. In the near future research into the infrastructure will focus on the development of the VR, which is the most important jGMA component yet to be developed.

References

[1] Foster, C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International J. Supercomputer Applications, 15(3), 2001.

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

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

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

[5] GGF Performance Working Group, A Grid Monitoring Architecture, http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf, 2002

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

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

[8] R-GMA testbed, http://hepunx.rl.ac.uk/edg/wp3/testbed.html

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

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

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

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

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

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

[15] AutoPilot, http://www-pablo.cs.uiuc.edu/Project/Autopilot/Autopilot Overview.htm‘

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

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

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

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

[20] Beowulf Cluster, http://www.beowulf.org/

[21] More Efficient Serialization and RMI for Java (1999), Michael Philippsen, Bernhard Haumacher, Christian Nester, Concurrency: Practice and Experience volume 12

[22] Java NIO, http://java.sun.com/j2se/1.4.2/docs/guide/nio/

[23] RFC2616, Hypertext Transfer Protocol, HTTP/1.1, http://www.w3.org/Protocols/rfc2616/rfc2616.html

[24] RFC1867, Form-based File Upload, HTML, http://www.ietf.org/rfc/rfc1867.txt

[25] Berkeley DB Java Edition, http://www.sleepycat.com/products/je.shtml

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

[27] jGMA: A lightweight implementation of the Grid Monitoring Architecture, DSG Technical Report, http://dsg.port.ac.uk/ mjeg/jGMA/jgma_report2004.pdf

[28] W3C Document Object Model, http://www.w3.org/DOM/

[29] UK E-Science All Hands Meeting, http://www.allhands.org.uk/

[30] Download jGMA, http://dsg.port.ac.uk/projects/jGMA/software/index.php

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

[32] Apache AXIS, http://ws.apache.org/axis/

[33] Gnutella, http://www.gnutella.com/

[34] Universal Description Discovery and Integration (UDDI), http://www.uddi.org/

[35] Domain Names - Implementation and Specification, http://www.faqs.org/rfcs/rfc1035.html

[36] Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems (2002), Matei Ripeanu, Ian Foster, IEEE Internet Computing Journal Volume 6

[37] Data Encryption Standard (DES) , Federal Information Processing Standards Publication 46-2 (1993), http://www.itl.nist.gov/fipspubs/fip46-2.htm

[38] Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, http://www.faqs.org/rfcs/rfc3279.html

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

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

Appendix A

Formal Training

A.0.1 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 website 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.

A.0.2 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 Sept. 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