P-3115

GRID MONITORING ARCHITECTURE

A PROJECT REPORT

Submitted by

G. RAGHUL 71206104037

S .VIMAL KUMAR 71206104058

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY : CHENNAI- 600 025

APRIL 2010

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"GRID MONITORING ARCHITECTURE"** is the bonafide work of "**G.RAGHUL** and **"S.VIMAL KUMAR"** who carried out the project under my supervision.

[illegible]	[illegible]
SIGNATURE	**SIGNATURE**
Prof.Dr. S. Thangasamy ,	Mrs.S.Rajini,
HEAD OF THE DEPARTMENT,	**SUPERVISOR,**
Department of Computer Science	Department of Computer Science
Kumaraguru College of Technology,	Kumaraguru College of Technology,
Chinnavedampatti Post,	Chinnavedampatti Post,
Coimbatore – 641 006	Coimbatore – 641 006

The candidates with University Register Nos.**71206104037 & 71206104058** were examined by us in the project viva-voce examination held on..15:04:10.......

[illegible]	[illegible]
INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made this possible and whose constant guidance and encouragement crowns all efforts with success.

We are extremely grateful to **Dr. Ramachandran**, Principal, Kumaraguru College of Technology for having given us this opportunity to embark on this project.

We express our sincere and heartfelt thanks to **Dr. S. Thangasamy,** Dean, Department of Computer Science and Engineering, for his kind guidance and support.

We would like to express our sincere thanks to our project coordinator **Mrs. P. Devaki,** for her valuable guidance during the course of the project.

We would also like to thank our class advisor **Mrs. R. Kalaiselvi,** for her constant support and guidance.

We would like to thank our guide, **Mrs.S.Rajini,** without whose motivation and guidance we would not have been able to embark on a project of this magnitude. We express our sincere thanks for her valuable guidance, benevolent attitude and constant encouragement.

We reciprocate the kindness shown to us by the staff members of our college, people at home and our beloved friends who have contributed in the form of ideas, constructive criticism and encouragement for the successful completion of the project.

ABSTRACT

In Large distributed systems such as Computational and Data Grids required a substantial amount of monitoring data to be collected for a variety of tasks such as fault detection, performance analysis, performance tuning, performance prediction and scheduling. Some tools are currently available and others are being developed for collecting and forwarding this data.

The goal of this project is to develop and model a Grid monitoring system, which monitors the services provided by the systems present in gird to its consumers. The desired capabilities of the system are fault detection, error reporting, status reporting of the services provided by the producers.

The above said capabilities of the system are achieved through this project.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

1. INTRODUCTION

The ability to monitor and manage distributed computing components is critical for enabling High-performance distributed computing. Monitoring of data is needed to determine the source of performance problems and to tune the system and application for better performance. Fault detection and recovery mechanisms need monitoring data to determine if a server is down, and whether to restart the server or redirect service requests elsewhere. A performance prediction service might use monitoring data as inputs for a prediction model, which would in turn be used by a scheduler to determine which resources to use.

There are several groups that are developing Grid monitoring systems to address this problem and these groups have recently seen a need to interoperate. In order to facilitate this, we have developed architecture for monitoring components. A Grid monitoring system is differentiated from a general monitoring system in that it must be scalable across wide-area networks, and include a wide range of heterogeneous resources. It must also be integrated with other Grid middleware in terms of naming and security issues. The Grid Monitoring Architecture (GMA) described here addresses these concerns and is sufficiently general that it could be adapted for use in distributed environments other than the Grid. For example, it could be used with large compute farms or clusters that require constant monitoring to ensure that all nodes are running correctly.

2. DESIGN AND IMPLEMENTATION

2.1 DESIGN:

With the potential for thousands of resources at geographically different sites and tens-of-thousands of simultaneous Grid users, it is important for the data management and collection facilities to scale while, at the same time, protecting the data from spoiling.

In order to allow scalability in both the administration and performance impact of such a system, the decision-making as to what is monitored, measurement frequency, and how the data is made available to the public must be widely distributed and dynamic. Thus, instead of a centralized management component, multiple independent management components synchronize their state through a directory service, which may itself be distributed.

Distributing management in this fashion also helps minimize the effects of host and network failure, making the system more robust under precisely the kinds of conditions it is trying to detect. In some models, such as the CORBA Event Service, all communication flows through a central component, which represents a potential bottleneck. In contrast, it is proposed that performance event data, which makes up the majority of the communication traffic, should travel directly from the producers of the data to the consumers of the data. In this way, individual producer/consumer pairs can do "impedance matching" based on negotiated requirements, and the amount of data flowing through the system can be controlled and localized fashion based on current load considerations.

The design also allows for replication and reduction of event data at intermediate components acting as consumer/producer caches or filters. Use that is of interest to many consumers, with subsequent reductions in the network traffic, as the intermediaries can be placed "near" the data consumers. The directory service contains only metadata about the performance events and system components and is accessed relatively infrequently, reducing the chance that it would be a bottleneck.

2.2 ARCHITECTURE:

The GMA architecture supports both a producer/consumer model, similar to several existing Event Service systems such as the CORBA Event Service, and a query/response model. For either model, producers or consumers that accept connections publish their existence in a directory service. Consumers use the directory service to locate one or more producers generating the type of event data they are interested in. Each consumer then subscribes to or queries the matching producer(s) directly. Likewise, a producer may query the directory service to locate consumer(s) that accept and process event data in a given manner – for example, a consumer that archives event data for later analysis. Once the appropriate consumer is identified, the producer would connect to it directly and stream the event data – similar in behaviour to what happens when a consumer subscribes to a producer, but initiated by the producer.

2.3 COMPONENTS:

The architecture consists of the following components, shown in Figure 1:

- **Consumers**
- **Producers**
- **Directory service**

Three interfaces defined are: the consumer to producer interface, the consumer to directory service interface, and the producer to directory service interface, thus the "standard" grid monitoring services that will all inter-operate are built.

2.3.1 DIRECTORY SERVICE:

To locate, name, and describe the structural characteristics of any data available to the Grid, a distributed directory service for publishing this information must be available. The primary purpose of this directory service is to allow information for consumers (users, visualization tools, programs and resource schedulers) to discover and understand their characteristics. In addition, information producers must

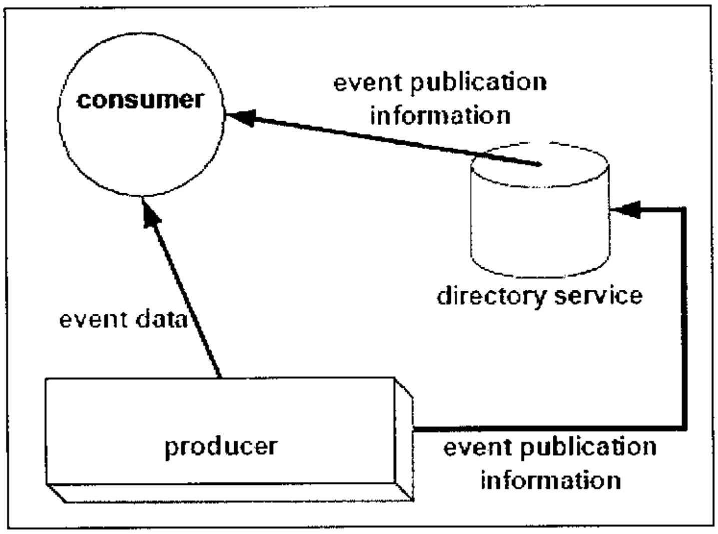


FIGURE1: Grid Monitoring Architecture Components

be able to update the information to reflect the system state. In the context of common operations for both consumers and producers, they will be collectively referred to as *clients*. The directory service contains a listing of all available event data and their associated producers. This allows consumers to discover what event data are currently available (through the producer registration), what the characteristics of the data are, and which producer to contact to receive a given type of event data. The directory service, however, is not responsible for

The names and characteristics associated with dynamic performance data are assumed to change slowly (unlike the performance data itself)i.e, the name and structural characteristics of a data set remain relatively constant while the valid contents of the data set may change dramatically over time.

The functions supported by a directory service are:

1.Authorize-consumer

Establish identity of a consumer, which is in turn mapped to access permissions for the next, or possibly several subsequent, transaction(s).

2. Authorize-producer

<same as Authorize-consumer>, although different mechanisms may be used for the two authorization operations.

3. Search

Perform a search for event data. The client should indicate whether only one result, or more than one result, if available, should be returned. An optional extension would allow the client to get multiple results one element at time using a "get next" query in subsequent searches.

• Preconditions: The client is authorized to perform the search.

• Post conditions: The result(s) of the search are returned, which includes a well-defined null value for searches which did not match in the directory.

4. Add

Add a record to the directory.

• Preconditions: The client is authorized to add the record. The record conforms to the directory's schema. The record is not a duplicate.

• Post conditions: The record is in the directory.

5. Remove

Remove a record from the directory

• Preconditions: The client is authorized to remove the record. The record matches exactly one record in the directory.

• Post conditions: The record is not in the directory.

6. Update

Change the state of a record in the directory.

• Preconditions: The client is authorized to modify the record. The record matches exactly one record in the directory.

• Post conditions: The record now has the new values.

7. Version request

A client may request the current version of the interface Query-optimized directory services such as LDAP, Globus MDS, the Legion Information Base, and the Novell NDS, all provide the necessary base functionality for this service, but only in their fully distributed implementations. Some public-domain implementations of these services do not support distributed implementation.

2.3.2 CONSUMER:

A consumer is any program that receives event data from a producer. Consumers that will accept asynchronous requests from producers will publish this information in the directory service. The functions supported by a consumer are:

1. Authorize to producer

The consumer contacts a producer and proves its identity. This may need to be performed once per "session", or on every request.

2. Authorize from producer

The consumer accepts authorization requests from a producer and verifies its identity. As in *Authorize to producer*, this may be done once per session or on every producer-initiated request.

3. Query

The consumer receives one event or set of events from the producer. Optional extensions are a *filter* to indicate interest in only a subset of events or to perform transformations on event data.

• Preconditions: The consumer is authorized to revive these event(s). The event data is available.

• Post conditions: One or more events are returned, together, in the reply.

4. Consumer-initiated Subscribe

The consumer establishes a connection to the producer to receive events in a stream.

• Preconditions: The consumer is authorized to connect to the producer and receive these event(s).

• Post conditions: Same as for Query, except that in addition to returning the most recent event, on success the producer will either (a) return events in a stream over the connection used for the request or (b) inform the consumer of the location of a new connection from which it can read the stream of events

• Other behaviours: If the consumer closes the established connection, the producer should simply consider the subscription ended (generating no errors). If the underlying source of event data stops producing data, the producer may close the connection without warning, so consumers should be designed to recover gracefully in this instance.

5. Consumer-initiated Unsubscribe

The consumer tells a producer to close the subscription. An optional extension is a "close all" version which closes all subscriptions for this consumer.

• Preconditions: The subscription exists for the producer/consumer pair. The consumer is authorized to end it.

• Post conditions: The subscription is removed. No more data should be sent for this subscription after the producer has confirmed.

6. Producer-initiated Subscribe

The consumer accepts subscriptions from producers who wish to send events.

• Preconditions: The producer is authorized to send events to this consumer.

• Post conditions: A new subscription is created for this producer/consumer pair.

7. Producer-initiated Unsubscribe

The consumer accepts an unsubscribe request from the producer.

• Preconditions: The subscription exists. The producer is authorized to end it.

• Post conditions: The subscription is removed.

8. Authorize to directory

The consumer contacts the directory service and proves its identity. This may need to be performed once per "session" or on every lookup.

9. Lookup

The consumer makes a query to the directory service, of which at least 2 types should be available:

(1) Producer: get data for a producer associated with an event.

(2) Event: get the description of the event.

• Preconditions: Authorization has been performed.

• Post conditions: The directory service is unchanged (read-only operation).

10. Update

The consumer updates records in the directory service regarding events for which this consumer will accept producer-initiated subscriptions.

• Preconditions: Authorization has been performed.

• Post conditions: The directory service has more/less/modified records reflecting the new information. There are many possible types of consumers. These may include:

• **real-time monitor:** This consumer is used to collect monitoring data in real time for use by real-time analysis tools. It checks the directory service to see what data is available, and then "subscribes" to all the events it is interested in. The producers then send the event data to the consumer as it is generated. Data from many sources can then be used for real-time performance analysis.

• **archiver:** This consumer may be used as to collect data for the archive service. It subscribes to the producers, collects the event data, and places it in the archive. A monitoring architecture needs this component, as it is important to archive event data in order to provide the ability to do historical analysis of system performance, and determine when/where changes occurred desirable to archive all monitoring data, it is desirable to archive a good sampling of both "normal" and "abnormal" system operation, so that when problems arise it is possible to compare the current system to a previously working system. It this architecture, the archive is just another consumer.

• **Process monitor**: This consumer can be used to trigger an action based on an event from a server process. For example, it might run a script to restart the processes, send email to a system administrator, call a pager, etc.

• **overview monitor**: This consumer collects events from several sources, and uses the combined information to make some decision that could not be made on the basis of data from only one host.

2.3.3 PRODUCER:

Producers are responsible for providing event data to consumers, either by request or asynchronously. Producers will publish event availability information in the directory service. The functions supported by a producer are:

1. Authorize from consumer

The producer establishes a consumer's identity and access permissions. Authorization may be combined with subscription or query requests, or performed separately with the results stored in a shared "key" of some sort.

2. Authorize to consumer

The producer contacts a consumer and proves its identity. As for *Authorize to producer*, this may need to be performed once per session or on every new request.

3. Query

The producer returns a single set of event(s) in response to a consumer query.

• Preconditions: The consumer is authorized to receive data about the event.

• Post conditions: The event data, if present, is returned.

4. Consumer-initiated Subscribe

Accept consumer requests to establish a stream of event data (subscription). This request should include parameters and filters, etc.

• Preconditions: Consumer is authorized to subscribe to requested event data.

• Post conditions: The subscription is added for a consumer and the producer either

(a) Returns events in a stream over the connection used for the request (or)

(b) Informs the consumer of the location of a new connection from which it can read the stream of events.

5. Consumer-initiated Unsubscribe

This is the normal operation by which a consumer ends its subscription. An optional "unsubscribe all" extension would allow the consumer to cancel all its subscriptions at once. As mentioned in the consumer section, if a consumer summarily closes its connection, the producer should automatically unsubscribe it everywhere.

• Preconditions: The subscription exists for this producer/consumer pair.

• Post conditions: The consumer/producer pair has one less subscription.

6. Producer-initiated Subscribe

A producer asynchronously begins a subscription with a consumer.

• Preconditions: The producer is authorized to send data to the consumer.

• Post conditions: The subscription is added and the producer may now send data.

7. Producer-initiated Unsubscribe

The producer informs a consumer that the subscription is ending. •

•Preconditions: The subscription exists for this consumer/producer pair. The consumer supports this function, allowing producers to asynchronously unsubscribe.

• Post conditions: The subscription is removed. Note that even in the case of failure the subscription may be removed by the producer.

8. Version

A consumer may request the current version of the interface. Producers can service "streaming" or "query" requests from consumers. In streaming mode the consumer makes a single request, then receives events in a stream until an explicit action is taken to end the connection. The producers are also used to provide access control to the event data, allowing different access to different classes of users. Since Grids typically have multiple organizations controlling the resources being monitored there may be different access policies (firewalls possibly), support for different frequencies of measurement, and willingness to allow access to different performance details for consumers "inside" or "outside" of the organization running the resource. Some sites may allow internal access to real-time event streams, while providing only summary data off-site. The producers would enforce these types of policy decisions. This mechanism is especially important for monitoring clusters or computer farms, where there may be a large amount of internal monitoring, but only a limited amount of monitoring data accessible to the Grid. There may also be components that are both consumers and producers. For example a consumer might collect event data from several producers, and then use that data to generate a new derived event data type, which is then made available to other consumers.

2.4 SOURCES OF EVENT DATA:

There are many possible sources of event data, including monitoring *sensors*. The following is a type of sensor which is used in the project to gather event data:

- **Process sensors**: Process sensors generate events when there is a change in process status (for example, when it starts, dies normally, or dies abnormally). They might also generate an event if some dynamic threshold is reached (for example, if the average number of users over a certain time period exceeds a given threshold).

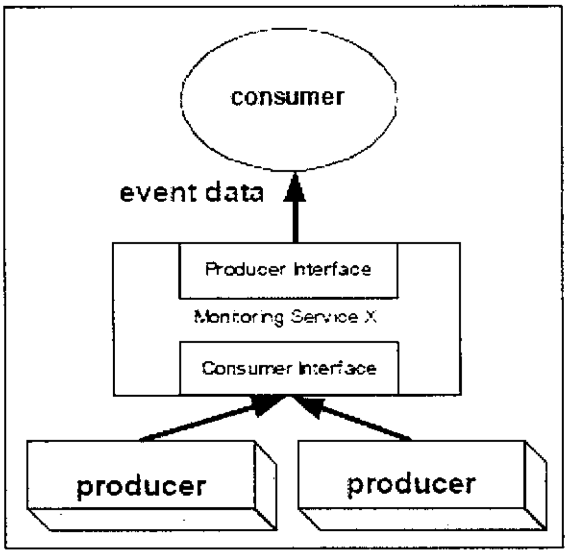


FIGURE 2: Joint consumer/producer

Optional Producer Tasks

There are many other services that producers might provide, such as event filtering and caching. For example, producers could optionally perform any intermediate processing of the data the consumer might require. A consumer might request that a prediction model be applied to a measurement history from a particular sensor, and then be notified only if the predicted performance falls below a specified threshold. The producer might in this case filter the data for the consumer and deliver it according the schedule the consumer determines. Another example is that a consumer might request that an event be sent only if its value crosses a certain threshold. Examples of such a threshold would be if CPU load becomes greater than 50%, or if load changes by more than 20%. The producer might also be configured to compute summary data. For example, it can compute 1, 10, and 60 minute averages of CPU usage, and make this information available to consumers.

Information on which services the producer provides would be published in the directory server, along with the event information.

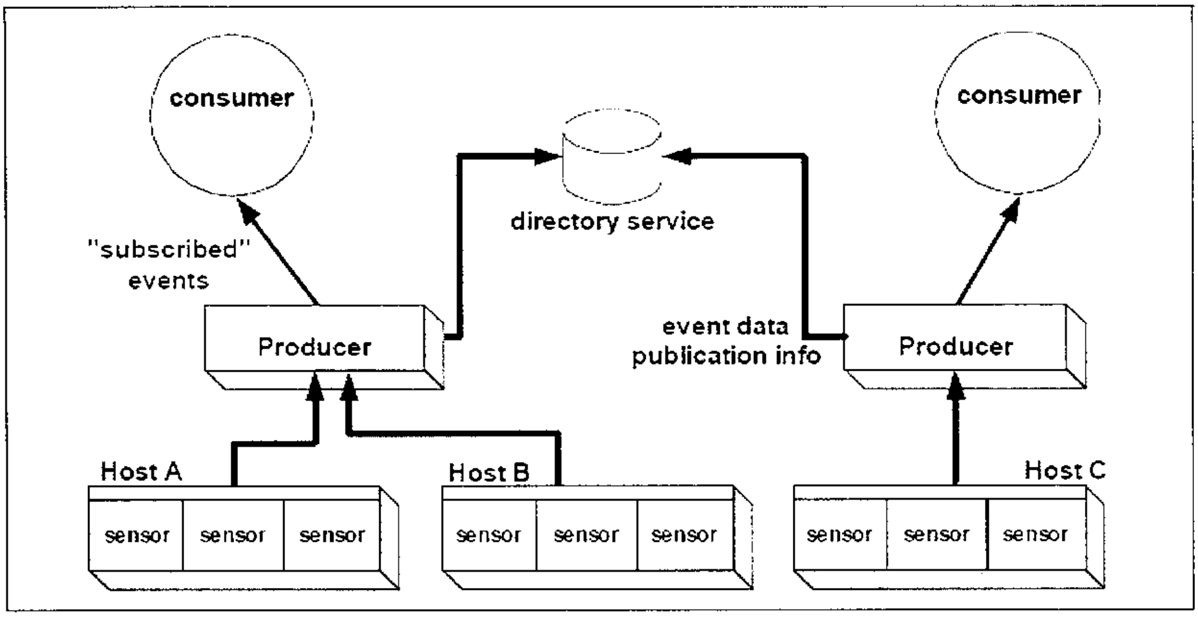


Figure 3: Relationship of producers and sensors

Monitoring service characteristics

The following characteristics distinguish performance monitoring information from other system data, such as files and databases.

Performance information has a fixed, often short lifetime of utility

Most monitoring data may go stale quickly making rapid read access important, but obviating the need for long-term storage. The notable exception to this is data that gets archived for accounting or post-mortem analysis.

• Updates are frequent

Unlike the more static forms of "metadata," dynamic performance information is typically updated more frequently than it is read. Most extant information-base technologies are optimized for query and not update, making them potentially unsuitable for dynamic information storage.

• Performance information is often stochastic

It is frequently impossible to characterize the performance of a resource or an application component using a single value. Therefore, dynamic performance information may carry *quality-of-information* metrics quantifying its accuracy, distribution, lifetime, etc., which may need to be calculated from the raw data.

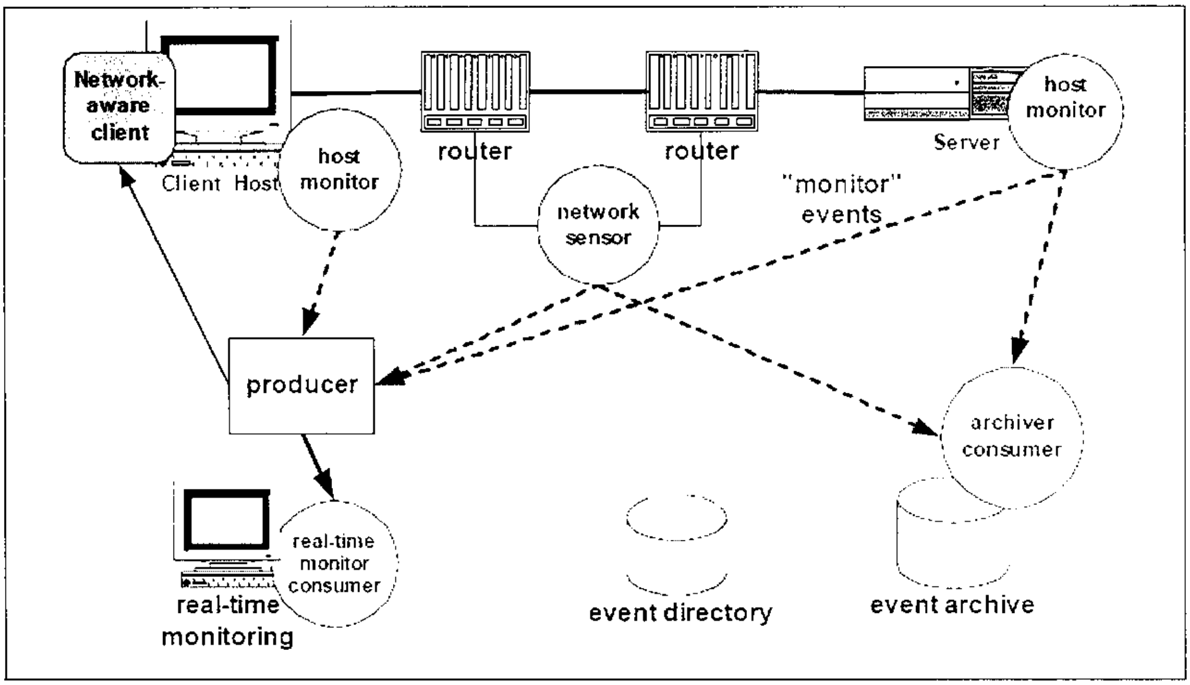


Figure 4: Sample use of monitoring system

• **Data gathering and delivery mechanisms must be high-performance**

Because dynamic data may grow stale quickly, the data management system must minimize the elapsed time associated with storage and retrieval. Note that this requirement differentiates the problem of dynamic data management from the problem of providing an archival performance record. The elapsed time to read an archive, while important, is often not the driving design characteristic for the archival system. We believe that archival data is useful both for accounting purposes and for long-term trend analysis. It is our belief, however, the separate but complimentary systems for managing and archiving Grid performance data respectively are required, each tailored to meet its own set of unique performance constraints.

• **Performance measurement impact must be minimized**

There must be a way for monitoring facilities to be able to limit their intrusiveness to an acceptable fraction of the available resources. If no mechanism for managing performance monitors is provided, performance measurements may simply measure the load introduced by other performance monitors.

General Implementation Strategies

The following are the factors to be considered when implementing a monitoring system.

• **The data management system must adapt to changing performance conditions dynamically**

Dynamic performance data is often used to determine whether the shared Grid resources are performing well (e.g. fault diagnosis) or whether Grid load will admit a particular application (e.g. resource allocation and scheduling). To make an assessment of dynamic performance fluctuation available, the data management system cannot, itself, be rendered inoperable or inaccessible by the very fluctuations it seeks to capture. As such, the data management system must use the data it gathers to control its own execution and resources in the face of dynamically changing conditions.

• **Dynamic data cannot be managed under centralized control**

Having a single, centralized repository for dynamic data (however short its lifespan) causes two distinct performance problems. The first is that the centralized repository for information and/or control represents a single-point-of-failure for the entire system. If the monitoring system is to be used to detect network failure, and a network failure isolates a centralized controller from separate system components, it will be unable to fulfil its role. All components must be able to function when temporarily disconnected or unreachable due to network or host failure. For dynamic data, writes often outnumber reads. That is, performance data may be gathered that is never read or accessed since demand for the data cannot be predicted. Experience has shown that a centralized data repository simply cannot handle the load generated by actively monitored resources at Grid scales.

• **All system components must be able to control their intrusiveness on the resources they monitor**

Different resources experience varying amounts of sensitivity to the load introduced by monitoring. A two megabyte disk footprint may be insignificant within a 10 terabyte storage system, but extremely significant if implemented for a palm-top or RAM disk. In general, performance monitors and other system components must have tuneable CPU, communication, memory, and storage requirements.

• **Efficient data formats are critical**

In choosing a data format, there are trade offs between ease-of-use and compactness. While the easiest and most portable format may be ASCII text including both event item descriptions and event item data in each transmission, this also the least compact. This format may be suitable for cases where a small amount of data is recorded and transmitted infrequently.

However, some sources of event data can generate huge volumes of data in a short amount of time, demanding that a more efficient data format be adopted. Compressed binary representations that can be read on machines with different byte orders is one possibility. Transmitting only the item data values and using a data structure obtained separately to interpret the data is another way to reduce the data volume. XML is an emerging standard that allows the data description to be separated from the data values. The XML schema could be placed in a separate directory server, retrieved, and used in conjunction with the event data values. Another possibility is to send the data descriptor one time when a consumer subscribes to a producer, and send only the data values for each event transmission.

The GMA could support registration of a data format for each event, allowing different events to use the format most appropriate for their needs. Consumers could be provided plug-in modules to convert from one format to another.

Scalability

In addition, for the GMA system to scale, performance monitoring consumers (particularly those that require the cooperation between two or more producers) must coordinate their interactions to control intrusiveness. For example, if network performance is to be monitored between all pairs of hosts attached to a single Ethernet segment, the network probes required to generate end-to-end measurements cannot occur simultaneously.

If they do, both the quality of the readings that are gathered and the network capacity that is available for other work will suffer. If performance monitors are not coordinated in the Grid, the intrusiveness of performance monitoring may strongly impact available performance, particularly as the system scales i.e, if all performance facilities operate their own monitoring sensors, Grid resources will be consumed by the monitoring facilities alone.

Coordinating a Grid-wide collection of sensors is complicated both by the scale of the problem (there are many Grid resource characteristics to monitor) and by the dynamically changing performance and availability of Grid resources that are being used to implement the dynamic data management service.

One recommended producer service that is important for system scalability is that of consumer-specified caching. Often a consumer needs to access only a small subset of the global data pool, and will sacrifice fast access for tight data consistency. An automatic program scheduler, for example, might want the "freshest "data that can be delivered for a specified set of hosts with no more than a one second access delay.

To achieve this functionality at Grid scales, producers must cache the data the consumer will want and deliver whatever data is available at the time of request. Experience with dynamic program scheduling indicates that this type of producer is valuable to scalable performance within the Grid.

SYSTEM TESTING

3. SYSTEM TESTING

3.1 INTRODUCTION:

After finishing the development of any computer based system the next complicated time consuming is system testing. During the time of testing only the development company can know that, how far the user requirements have been met out, and so on.

Following are the some of the testing methods applied to this effective project:

3.2 SOURCE CODE TESTING:

This examines the logic of the system. If we are getting the output that is required by the user, then we can say that the logic is perfect.

3.3 MODULE LEVEL TESTING:

In this the error will be found at each individual module, it encourages the programmer to find and rectify the errors without affecting the other modules.

3.3 INTEGRATION TESTING:

Data can be tested across an interface. One module can have an inadvertent, adverse effect on the other. **Integration testing** is a systematic technique for constructing a program structure while conducting tests to uncover errors associated with interring.

3.5 VALIDATION TESTING:

It begins after the integration testing is successfully assembled. Validation succeeds when the software functions in a manner that can be reasonably accepted by the client. In this the majority of the validation is done during the data entry operation where there is a maximum possibility of entering wrong data. Other validation will be performed in all process where correct details and data should be entered to get the required result.

3.6 PERFORMANCE TESTING:

Performance Testing is used to test runtime performance of software within the context of an integrated system. Performance test are often coupled with stress testing and require both software instrumentation.

3.7 OUTPUT TESTING:

After performing the validation testing, the next step is output testing of the proposed system since no system would be termed as useful until it does produce the required output in the specified format. **Output format** is considered in two ways, the **screen format** and the **printer format.**

3.8 USER ACCEPTANCE TESTING:

User Acceptance Testing is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system users at the time of developing and making changes whenever required

SYSTEM REQUIREMENTS

4.SYSTEM REQUIREMENTS & IMPLEMENTATION

4.1.1 HARDWARE REQUIREMENTS

1.	Processor Type	:	Pentium 4
2.	Processor Speed	:	1.4 MHZ
3.	RAM	:	266 RAM
4.	Hard Disk Capacity	:	20 GB

4.1.2 SOFTWARE REQUIREMENTS

1. Operating System	: Microsoft windows XP, Windows 2000 professional
2. Programming Language	: JAVA

IMPLEMENTATION & RESULTS

5.IMPLEMENTATION&RESULTS

5.1 IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system. The most critical stage is achieving a successful system and giving confidence on the new system for the users that it will work efficiently. The implementation process begins with preparing a plan for the implementation of the system.

The language used to develop this project is JAVA. In JAVA the RMI programming is the main concept behind the development of this project. The process sensors are used to collect event data. The JNDI also plays an important role; it acts as the directory service for this project

5.2 RESULTS:

We executed the project in windows platform using java. We implemented the system using a single system; both client and server are simulated in the same system.

The following results are obtained, while executing

Result 1: Appropriate response obtained

Condition: if all the services are available and working correctly

Result 2: Service out of order

Condition: If a particular service is not available due to server shutdown or some other problems.

Result 3: Service corrupted

Condition: If the service is corrupted due to errors or due to server crash.

CONCLUSION & FUTURE ENHANCEMENTS

6.CONCLUSION AND FUTURE ENHANCEMENTS

We presented a performing and portable implementation of the Grid Monitoring Architecture. Performance was obtained through the use of JAVA as base implementation language; we obtained good results in terms of throughput and response time, both for producers and the directory service. Multiple ready-to-use consumers have been implemented

In the future there is a possibility to record the resource's presence in the directory service, auto install suitable sensor and register with a set of producers (based on the type of monitoring data offered and the resource's location).

Also there is a scope to develop proxy producers which allow collecting sensor data from other monitoring platforms, information providers and use these data in our framework.

APPENDICES

7.1 SAMPLE CODE

CONSUMER:

```
import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.*;

public class GMA extends JFrame implements ActionListener

{

        JMenuBar mbar;

        JMenu File,Help;

        JMenuItem Init,Moniter,Exit,About;

        String [] list={"factser","addser"};

        JComboBox combo;

        public GMA()

        {

        }

        public  void init()

        {

                mbar=new JMenuBar();

                File=new JMenu("File");

                JPanel panel=new JPanel();

                Help=new JMenu("Help");

                //Init=new JMenuItem("Initilize");

                //Init.addActionListener(this);

                //Init.setActionCommand("init");

                Moniter=new JMenuItem("Call Producer");

                Moniter.setActionCommand("monitor");

                Moniter.addActionListener(this);
        Exit.addActionListener(this);
        Exit.setActionCommand("exit");
        //File.add(Init);

        File.add(Moniter);
        File.addSeparator();
        File.add(Exit);
        mbar.add(File);
        mbar.add(Help);
        combo=new JComboBox(list);
        combo.setSelectedIndex(0);
        combo.addActionListener(this);

        JButton but=new JButton("Connect");
        but.addActionListener(this);
        but.setActionCommand("click");
        panel.add(combo);
        panel.add(but);
        this.setJMenuBar(mbar);
        getContentPane().add(panel);
        setSize(400,400);
        show();
}

public void actionPerformed(ActionEvent arg0)
{
        if(arg0.getActionCommand()=="monitor")
        {
```

```
        //System.out.print("clicked");
        //JComboBox cb = (JComboBox)arg0.getSource();
       String petName = (String)combo.getSelectedItem();
        GMAConnect g=new GMAConnect(petName);
        //System.out.print(petName);


        }
        if(arg0.getActionCommand()=="exit")
        {
        System.out.print("clicked");
        //GMAConnect g=new GMAConnect();
        }

    }
}
```

PRODUCER:

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;
public class addserverimp extends UnicastRemoteObject implements addint
{

public addserverimp() throws RemoteException
 {
   }

public int fact(int a)
{
        if(a==0)
        return 1;
        else
        return (a*fact(a-1));
        //return 0;
        }


 public String Swlog() throws RemoteException
  {
        try {
                FileReader file=new FileReader("log.txt");
                BufferedReader br=new BufferedReader(file);
```

```
            return str;
        }

catch (Exception e)
{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;

}

}
```

USER INTERFACE FOR CONSUMER:

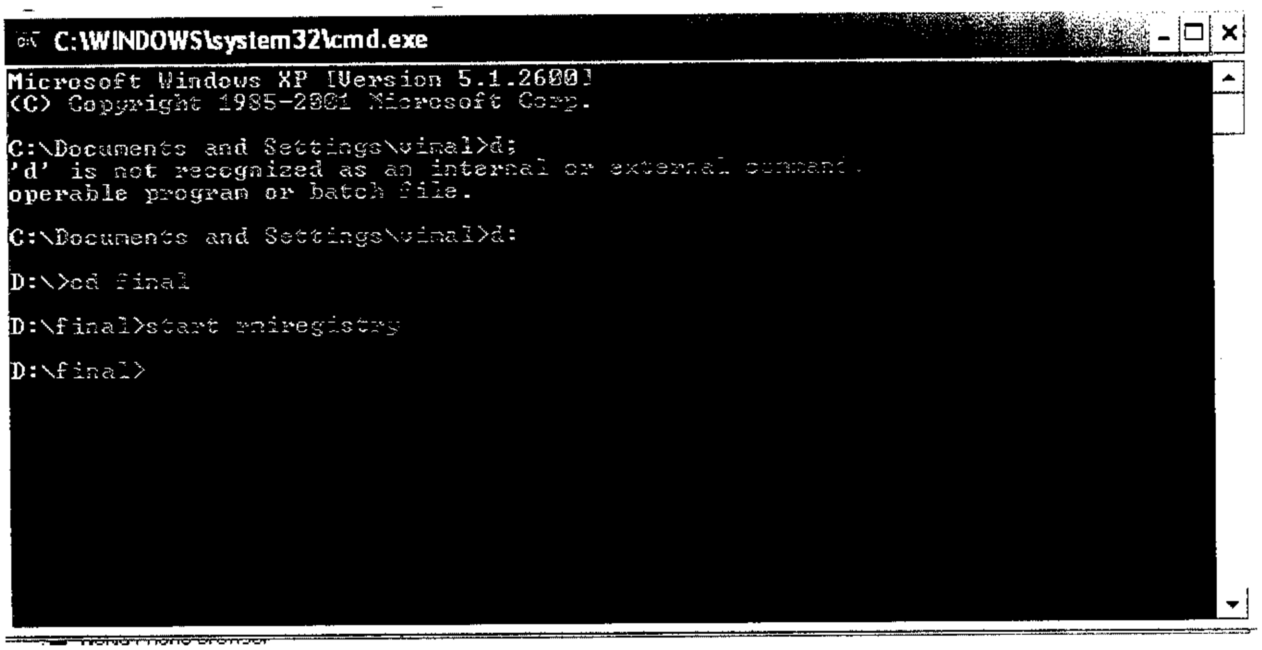
```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
public class GMAConnect extends JFrame implements ActionListener
{
        JLabel label,label1;
        JButton button;
        JTextField tf,tf1;
        String [] list={"factser","addser"};
        JComboBox combo;
        String service;
        public GMAConnect(String ser)
{
                service=ser;
                setLayout(new FlowLayout(FlowLayout.LEFT));
                label=new JLabel("Address");
                tf=new JTextField(10);
                combo=new JComboBox(list);

        combo.setSelectedIndex(0);
        combo.addActionListener(this);
        button=new JButton("Connect");
        button.addActionListener(this);
        label1=new JLabel("Number");
        tf1=new JTextField(10);
        //getContentPane().add(label);
        //getContentPane().add(tf);
        getContentPane().add(label1);
        getContentPane().add(tf1);
        getContentPane().add(button);
        setSize(200,100);
        show();
        }

        public void actionPerformed(ActionEvent e)
{
         cli c=new cli(service,tf1.getText());
         this.setVisible(false);
        }


}


import java.rmi.*;
import java.io.*;
public class cli
{
public cli(String add,String num){
try
{
         int a=Integer.parseInt(num);
         //int a=5;
         String connect="rmi://127.0.0.1/" + add;
        String [] list=Naming.list("rmi://127.0.0.1");
        for(int i=0;i<list.length;i++)
        System.out.print(list[i] + list.length + "\n");
        if(add=="factser"){
        addint ab=(addint)Naming.lookup(connect);
        if(ab.fact(a)==0)
        {
        System.out.print(ab.Swlog());
        }
        else
        System.out.println(ab.fact(a));
        }
        Else
        {
        if(add=="addser"){
        subint sb=(subint)Naming.lookup(connect);
        System.out.println(sb.square(a));
}
}
}
        catch(Exception e){
        System.out.println("unable to connect to producer");
}
}
}
```

ADDING INTERFACE:

```
import java.net.*;
import java.rmi.*;
public class addser
{
public addser(){}
public static void main(String [] s)
{
        Try
        {
        addserverimp obj=new addserverimp();
        addserverimp1 obj1=new addserverimp1();
        Naming.rebind("factser",obj);
        Naming.rebind("addser",obj1);
        }
        catch(Exception e)
        {
        System.out.println(e);
        }
}
}
```

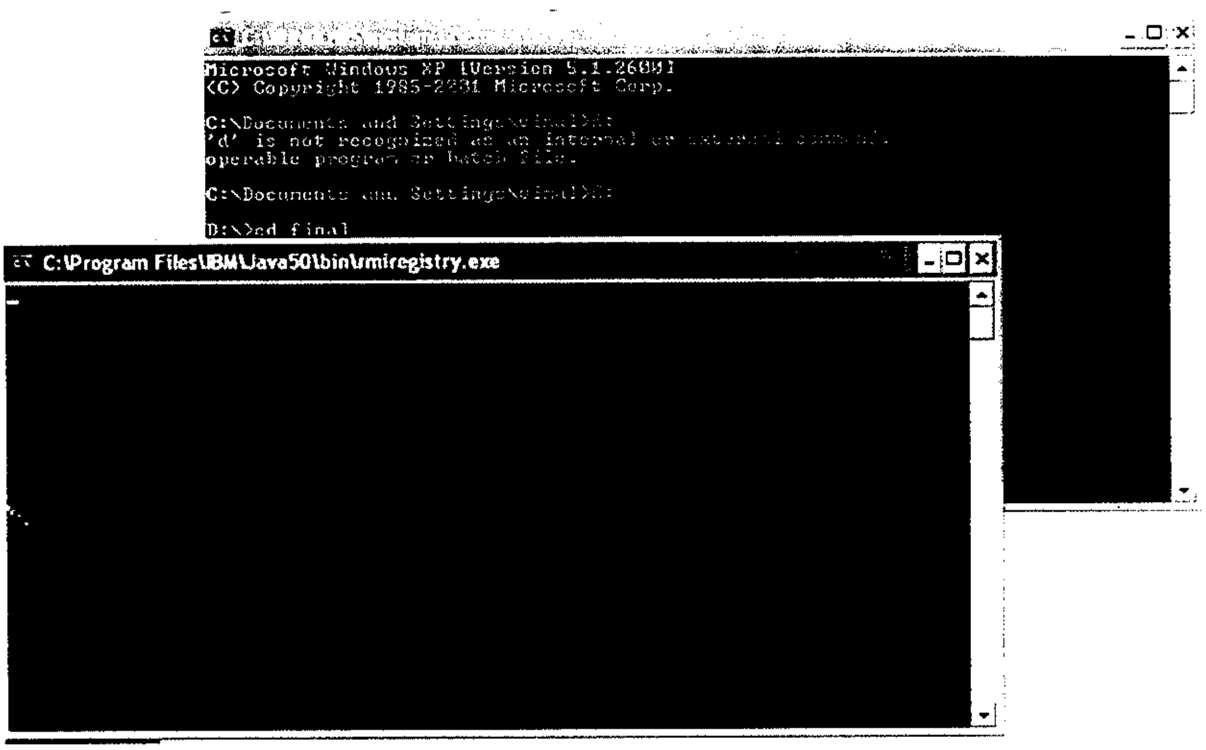
7.2 SCREENSHOTS

STARTING DIRECTORY SERVICE:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\vimal>d;
'd' is not recognized as an internal or external command,
operable program or batch file.

C:\Documents and Settings\vimal>d:

D:\>cd final

D:\final>start rmiregistry

D:\final>
```

DIRECTORY SERVICE STARTED:

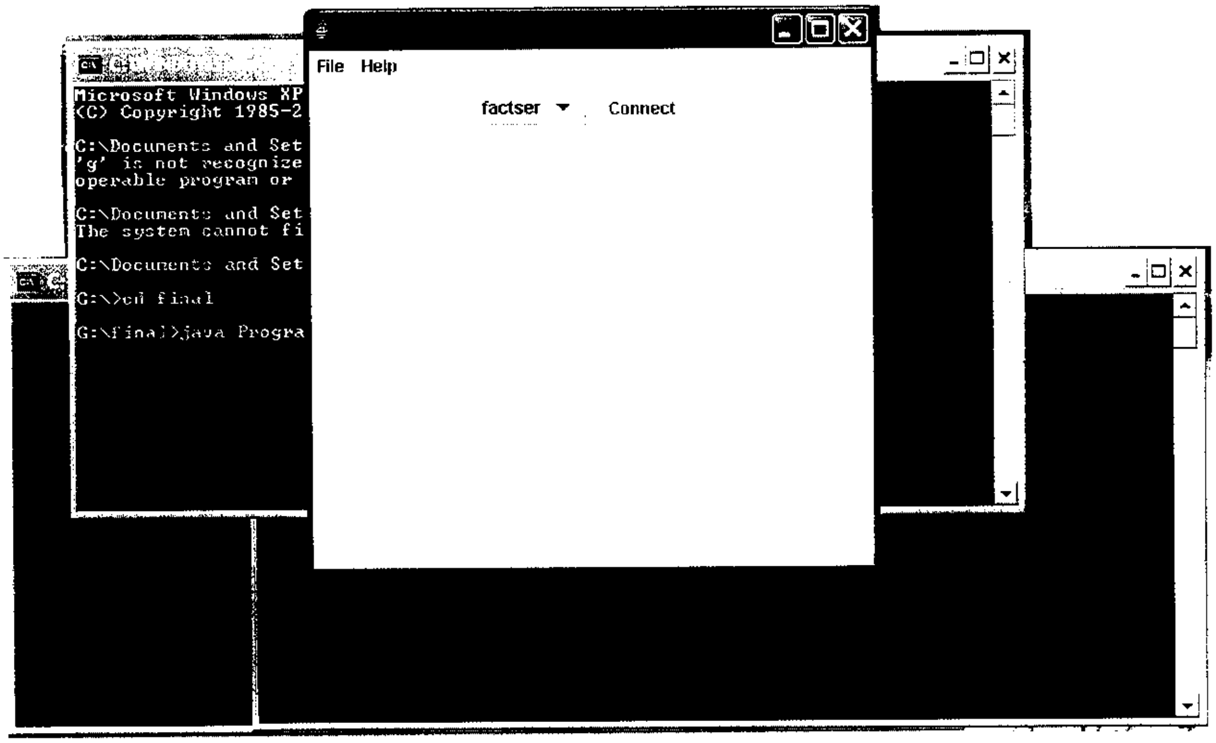


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\[illegible]>d
'd' is not recognized as an internal or external command,
operable program or batch file.

C:\Documents and Settings\[illegible]>d:

D:\>cd final
```

C:\Program Files\IBM\Java50\bin\rmiregistry.exe

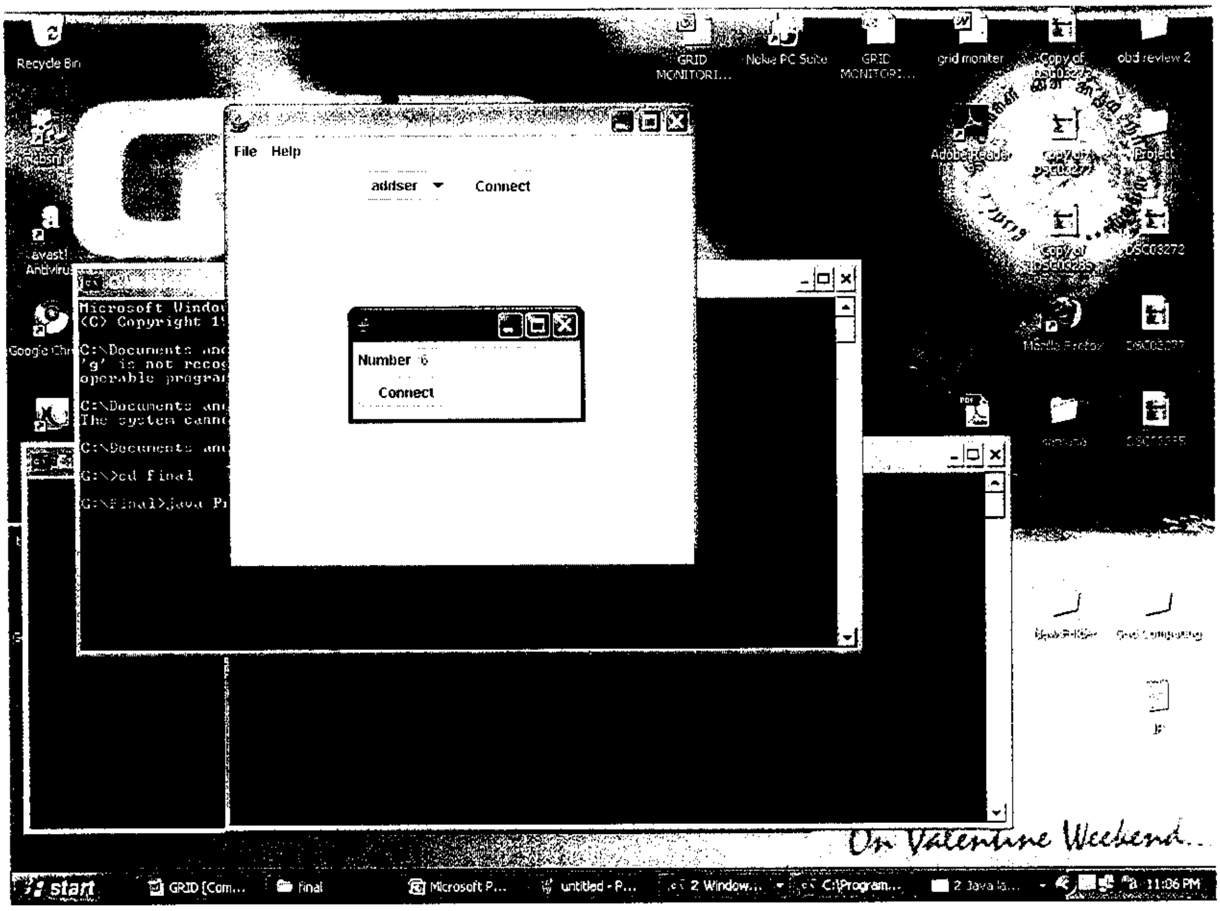
```
_
```

USER INTERFACE FOR CONSUMER:



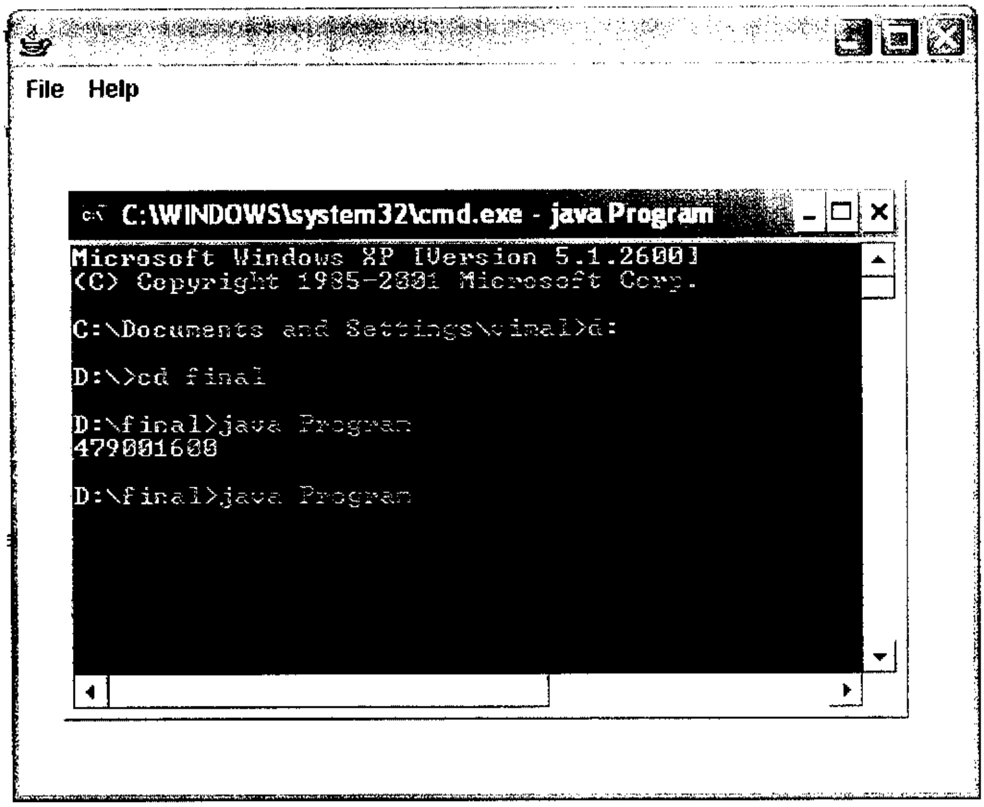
```
Microsoft Windows XP
(C) Copyright 1985-2

C:\Documents and Set
'g' is not recognize
operable program or

C:\Documents and Set
The system cannot fi

C:\Documents and Set

G:\>cd final

G:\final>java Progra
```

CONSUMER REQUESTING:



```
Microsoft Windo
(C) Copyright 1

C:\Documents and
'g' is not recog
operable progra

C:\Documents an
The system cann

C:\Documents an

G:\>cd final

G:\Final>java P
```

RESPONSE:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\vimal>d:

D:\>cd final

D:\final>java Program
479001600

D:\final>java Program
```

REFERENCES

8.REFERENCES

[1] CORBA, "Systems Management: Event Management Service", X/Open Document Number: P437, http://www.opengroup.org/onlinepubs/008356299/

[2] Jini distributed Event Specification, http://www.sun.com/jini/specs/

[3] W. Smith, "Monitoring and Fault Management," http://www.nas.nasa.gov/~wwsmith/mon_fm

[4] A. Waheed, W. Smith, J. George, J. Yan. "An Infrastructure for Monitoring and Management in Computational Grids." In Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems.