The CanonicalProducer: an instrument monitoring component of the Relational Grid Monitoring Architecture (R-GMA)

Rob Byrom[§], Brian Coghlan[**], Andrew Cooke[*], Roney Cordenonsi[‡], Linda Cornwall[¶], Ari Datta[‡], Abdeslem Djaoui[¶], Laurence Field[§], Steve Fisher[¶], Steve Hicks[§], Stuart Kenny[**], James Magowan[†], Werner Nutt[*], David O'Callaghan[**], Manfred Oevers[†], Norbert Podhorszki[‖], John Ryan[**], Manish Soni[§], Paul Taylor[†], Antony Wilson[§] and Xiaomei Zhu[§]
[*]Heriot-Watt, Edinburgh, UK [†]IBM Hursley Laboratory, UK [‡]Queen Mary, University of London, UK [§]PPARC. UK [¶]Rutherford Appleton Laboratory, UK [‖]SZTAKI, Hungary [**]Trinity College Dublin, Ireland

***Abstract*— We describe how the R-GMA (Relational Grid Monitoring Architecture) can be used to allow for instrument monitoring in a Grid environment. The R-GMA has been developed within the European DataGrid Project (EDG) as a Grid Information and Monitoring System. It is based on the Grid Monitoring Architecture (GMA) from the Global Grid Forum (GGF), which is a simple Consumer-Producer model. The special strength of this implementation comes from the power of the relational model. It offers a global view of the information as if each Virtual Organisation had one large relational database. It provides a number of different Producer types with different characteristics; for example some support streaming of information. We describe the R-GMA component that allows for instrument monitoring, the CanonicalProducer. We also describe an example use of this approach in the European CrossGrid project, SANTA-G, a network monitoring tool.**

***Index Terms*— Grids, Instrument Monitoring, Grid Monitoring Architecture, Grid Information and Monitoring Systems, R-GMA, CanonicalProducer, SANTA-G**

I. The R-GMA

The Grid Monitoring Architecture (GMA) [2] of the Global Grid Forum (GGF), as shown in Figure 1, consists of three components: *Consumers*, *Producers* and a directory service, which in the R-GMA is referred to as a *Registry*).

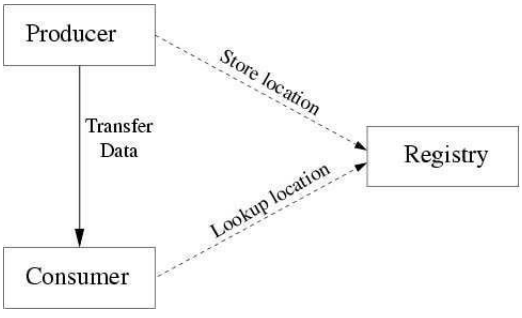

Fig. 1. Grid Monitoring Architecture

In the GMA Producers register themselves with the Registry and describe the type and structure of information they want to make available to the Grid. Consumers can query the Registry to find out what type of information is available and locate Producers that provide such information. Once this information is known the Consumer can contact the Producer directly to obtain the relevant data. By specifying the Consumer/Producer protocol and the interfaces to the Registry one can build interoperable services. The Registry communication is shown on Figure 1 by a dotted line and the main flow of data by a solid line.

The current GMA definition also describes the registration of Consumers, so that a Producer can find a Consumer. The main reason to register the existence of Consumers is so that the Registry can notify them about changes in the set of Producers that interests them. Although the GMA architecture was devised for monitoring, the R-GMA uses it as a basis for a *combined* information and monitoring system. The case for this was argued in [4]; that the only thing which characterises monitoring information is a time stamp, so in the R-GMA there is a time stamp on all measurements, saying that this is the time when the measurement was made, or equivalently the time when the statement represented by the tuple was true.

The GMA does not constrain any of the protocols nor the underlying data model, so the implementation of the R-GMA was free to adopt a data model which would allow the formulation of powerful queries over the data.

R-GMA is a relational implementation of the GMA, developed within the European DataGrid (EDG), which harnesses the power and flexibility of the relational model. R-GMA creates the impression that you have one RDBMS per Virtual Organisation (VO). However it is important to appreciate that the system is a way of using the relational model in a Grid environment and *not* a general distributed RDBMS with guaranteed ACID properties. All the producers of information are quite independent. It is relational in the sense that Producers announce what they have to publish via an SQL CREATE TABLE statement and publish with an SQL INSERT and that Consumers use an SQL SELECT to collect the information they need. For a more formal description of R-GMA see [3].

R-GMA is built using servlet technology and is being migrated rapidly to web services, specifically to fit into an OGSA/OGSI [5] framework.

There have so far been defined not just a single Producer but four different types: a DataBaseProducer, a StreamProducer, a LatestProducer and a CanonicalProducer. All appear to be

Producers as seen by a Consumer, but they have different characteristics.

The producers are instantiated and given the description of the information they have to offer by an SQL `CREATE TABLE` statement and a `WHERE` clause expressing a predicate that is true for the table. Currently this is of the form `WHERE (column_1=value_1 AND column_2=value_2 AND ...)`. To publish data, in all but the CanonicalProducer, a method is invoked which takes the form of a normal SQL `INSERT` statement. The CanonicalProducer, though in some respects the most general, is somewhat different due to the absence of a user interface to publish data via an SQL `INSERT` statement; instead, it triggers user code to answer an SQL query. For more detail see Section III.

A Consumer uses the Registry to find out what type of information is there, and where it is. The R-GMA Registry stores information about all producers currently available. The R-GMA, uniquely, includes a mediator (a kind of broker that is hidden behind the Consumer interface) specifically to make the R-GMA easy to use. The mediator knows that Producers are associated with views on a virtual data base. Currently views have the form:

SELECT * FROM <table> WHERE <predicate>

This view definition is stored in the Registry. When queries are posed by a Consumer, the Mediator uses the Registry to find the right Producers and then combines information from them.

II. R-GMA Architecture

R-GMA is currently based on Servlet technology. Each component has the bulk of its implementation in a Servlet. Multiple APIs in Java, C++, C, Python and Perl are available for user code to communicate with the servlets. The R-GMA makes use of the Tomcat Servlet container. Most of the R-GMA code is written in Java and is therefore highly portable. The only dependency on other EDG software components is in the security area.

Figure 2 shows the communication between the APIs and the Servlets. When a Producer is created its registration details are sent via the Producer Servlet to the Registry (Figure 2a). The Registry records details about the Producer, which include the description and view of the data published, *but not the data itself*. The description of the data is actually stored as a reference to a table in the Schema. In practise the Schema is co-located with the Registry. Once registration is completed, then whenever the Producer publishes data, the data are transferred to a local Producer Servlet (Figure 2b).

When a Consumer is created its registration details are also sent to the Registry although this time via a Consumer Servlet (Figure 2c). The Registry records details about the type of data that the Consumer is interested in. The Registry then returns a list of Producers back to the Consumer Servlet that match the Consumer's selection criteria.

The Consumer Servlet then contacts the relevant Producer Servlets to initiate transfer of data from the Producer Servlets to the Consumer Servlet as shown in Figures 2d-e. The data are then available to the Consumer on the Consumer Servlet, which should be close in terms of the network to the Consumer (Figure 2f).

As details of the Consumers and their selection criteria are stored in the Registry, the Consumer Servlets are automatically notified when new Producers are registered that meet their selection criteria.

The system makes use of soft state registration to make it robust. Producers and Consumers both commit to communicate with their servlet within a certain time. A time stamp is stored in the Registry, and if nothing is heard by that time, the Producer or Consumer is unregistered. The Producer and Consumer servlets keep track of the last time they heard from their client, and ensure that the Registry time stamp is updated in good time.

III. The CanonicalProducer

If we have to deal with a large volume of data it may not be practical to convert it all to a tabular storage model. Moreover, it may be inefficient to transfer the data to a Producer servlet with SQL INSERT statements. It may be judged better to leave the data in its raw form at the location where it was created. The CanonicalProducer is able to cope with this by accepting SQL queries and using user supplied code to return selected information in tabular form when required.

In general the R-GMA producers are sub-classes of the Insertable class, the class that provides the insert method. The insert method is used by the producers to send data to the servlets as an SQL INSERT string. The CanonicalProducer is different however; it is a subclass of the Declarable class. This means that it inherits the methods for declaring tables, but not inserting data. The user's producer code is responsible for obtaining the data requested. Figure 3 shows the communication between the servlets for a CanonicalProducer. When the other producer types publish data, the data is transferred to a local producer servlet via a SQL INSERT. The CanonicalProducer Servlet, however, is never sent raw data, which is instead retained local to the user's CanonicalProducer code.

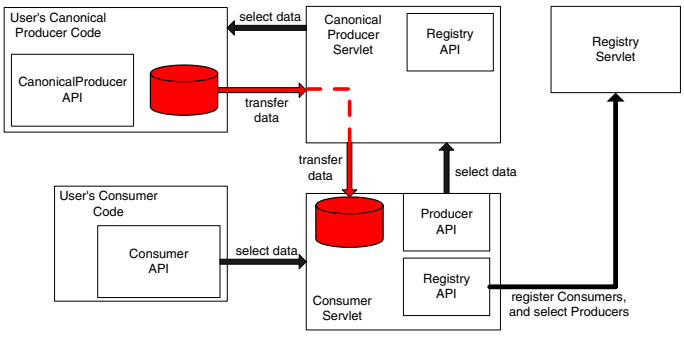

Fig. 3. CanonicalProducer Servlet Communication

A CanonicalProducer is instantiated by calling the API constructor method:

```
CanonicalProducer myProducer =
new CanonicalProducer
( 8998, CanonicalProducer.HISTORY );
```

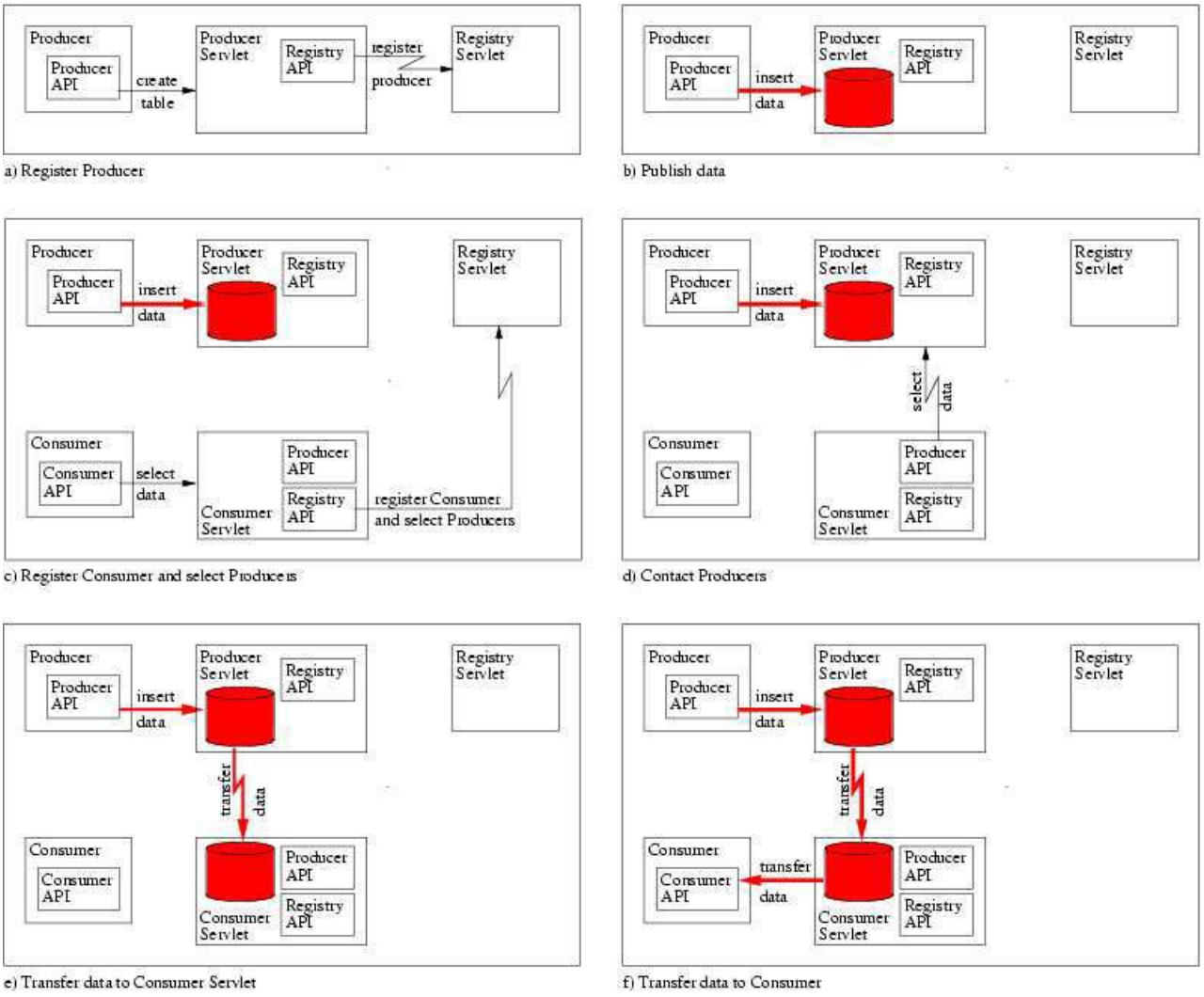

Fig. 2. Relational Grid Monitoring Architecture

This creates a new CanonicalProducer object, which registers itself with the CanonicalProducerServlet. The first parameter is a port number. The CanonicalProducerServlet expects to be able to connect, by way of a socket connection, to the CanonicalProducer code on this port in order to satisfy SQL queries. The second parameter describes the type of query that this producer code can satisfy, HISTORY or LATEST.

The table, or tables, that this producer publishes are then declared using the `declareTable` method.

```
myProducer.declareTable
( "cpuLoadUsage",
  " WHERE (ipAddress='"
  + this.ipAddress + "')",
  "CREATE TABLE cpuLoadUsage(
  ipAddress VARCHAR(50)
  NOT NULL PRIMARY KEY,
  cpuLoad REAL)"
)
```

When the servlet receives a query it opens a socket connection on the given port number to the CanonicalProducer code and forwards the SQL SELECT query to the producer code. The producer code must then execute the query, in whatever way it likes, and return a ResultSet to the servlet. The servlet can then return this ResultSet to the consumer. With the other producer types the producer is never aware of the SQL SELECT queries, they simply push the data to the servlet, and it is the servlet that carries out the SQL query. With a CanonicalProducer, however, the servlet has only the very minimum functionalty. To satisfy the query, it simply acts as an intermediary, forwarding the query to the correct CanonicalProducer instance and waiting for results to be returned.

A typical implementation of CanonicalProducer code would consist of several components, as shown in Figure 4. Although the figure shows the data being collected by an instrument and stored in log files, the data source could be anything.

CanonicalProducer Code would be the main class implemented by the user, which would use the CanonicalProducer API to instantiate a producer object, and declare the tables that the

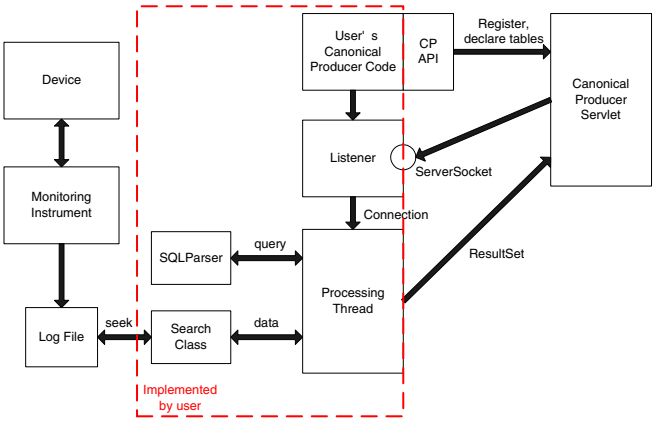

Fig. 4. An example of CanonicalProducer user code

producer publishes. It would then start a Listener to wait for connections from the servlet.

Listener — would be created by the main class. It would need to create a ServerSocket and then listen on this socket for connections from the servlet. When a connection is obtained it would be passed to a processing thread to execute the query and then continue listening for new connections.

Processing Thread — would receive the connection to the servlet from the Listener. The processing thread would read the SQL SELECT query from the socket connection, and process it over the available data. When the results had been accumulated they would then be returned to the servlet, over the same socket connection.

SQL Parser — Some additional classes would have to be used by the processing thread. A class would be needed to parse the SQL SELECT received from the servlet.

Search Class — A class would also be needed to search the data for the required results to satisfy the query. This class might, for example, perform seek operations on a binary log file to find the data, or possibly invoke a script to collect the data.

Results should be returned to the servlet as XML ResultSets. The form of these is as follows:

```
<?xml version = '1.0' encoding='UTF-8'
"standalone='no'?>
<edg:XMLResponse
xmlns:edg='http://www.edg.org'>
<XMLResultSet>
<rowMetaData>
<colMetaData>ColumnName</colMetaData>
</rowMetaData>
<row><col>ColumnValue</col></row>
</XMLResultSet>
</edg:XMLResponse>
```

An important issue with the CanonicalProducer is the following. For the other producer types one can estimate how often the producer will contact the servlet, as it should be regularly inserting data. This is not the case with the CanonicalProducer. Because the CanonicalProducer never actually inserts data, the servlet will never be informed as to whether the producer is still alive, and therefore will not inform the registry. After the R-GMA *termination interval* the CanonicalProducer would be presumed to be dead and its details would be removed from the registry. To avoid this a CanonicalProducer implementation should ensure that it regularly sends a sign of life to the servlet. This can be achieved by a thread that periodically, at intervals less than the termination interval, contacts the servlet.

Because the user must write the code to parse and execute the query, the CanonicalProducer can be used to carry out any type of query on any type of data source.

IV. Example Use of the CanonicalProducer: SANTA-G

SANTA-G (Grid-enabled System Area Networks Trace Analysis) is a generic template for ad-hoc, non-invasive monitoring with external instruments, see Figure 5. The template allows for the information captured by external instruments to be introduced into the Grid Information System. It is possible for these intruments to be anything, from fish sonars to PCR Analysers. The enabling technology for the template is the CanonicalProducer. The demonstrator of this concept, developed within the CrossGrid [11] project, is a network tracer that allows a user to analyse the Ethernet traffic at a site. The information obtained is useful for both the validation and calibration of intrusive monitoring systems and also for performance analysis.

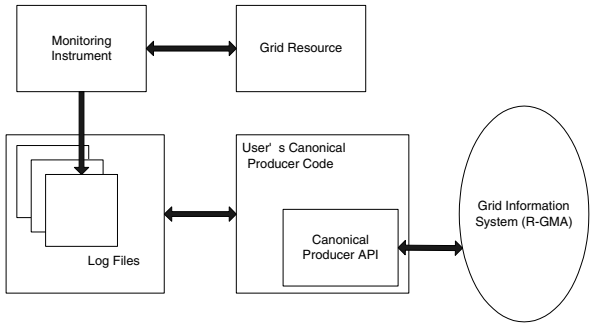

Fig. 5. SANTA-G monitoring framework

The SANTA-G NetTracer is composed of three components that allow for the monitoring data to be accessed through the R-GMA: a Sensor (which is installed on the node(s) to be monitored), a QueryEngine, and a Viewer GUI, see

Figure 6. The Sensor invokes Tcpdump (an open-source packet capture application), and then monitors the log files created. The Sensor notifies the QueryEngine when new log files are detected. The QueryEngine records these events in a database, which is published to users through the R-GMA (by using the LatestProducer API). The QueryEngine also includes the interface to the R-GMA by using the CanonicalProducer API. Data is viewed via the R-GMA by submitting an SQL SELECT statement, as if querying a relational database. Through the CanonicalProducer this query is forwarded to the QueryEngine, which then parses the query, searches the appropriate log file to obtain the data required to satisfy the query, and returns the dataset to the GUI through the R-GMA.

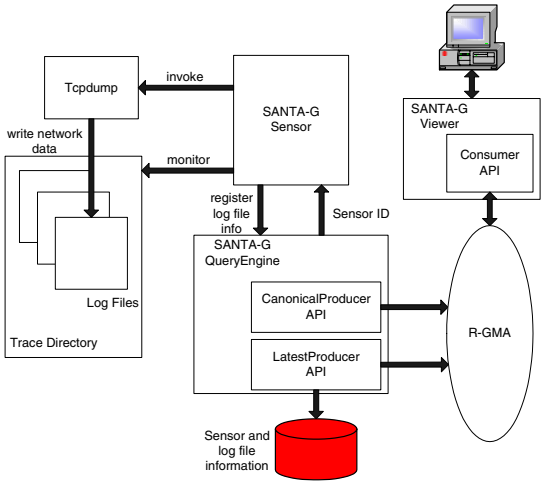

Fig. 6. SANTA-G NetTracer

It is the SANTA-G QueryEngine that implements the components of the CanonicalProducer code as described in Section III. Figure 7 shows how the QueryEngine executes a SQL query received from the R-GMA (i.e. from the CanonicalProducerServlet). The QueryEngine listens on a socket, waiting for connections from the Servlet. When a connection is made the SQL query is read from the socket and passed to an SQLParser class. The parser breaks the query into three separate lists; a select list that contains the network header fields to be read, a from list that contains the table the fields belong to, and a where list that contains the values used to match the packets to. The Search class searches the log file for network packets that match the WHERE predicates specified in the query, and extracts the required packet header fields from them. The data that satisfies the query is accumulated into a ResultSet in XML format and returned to the Servlet over the socket connection. For example, the following query:

```
SELECT source_address, destination_address,
packet_type
FROM Ethernet
WHERE sensorId = 'some.machine.com:0'
AND fileId = 0
AND packetId < 100
```

would return the source address, destination address, and packet type fields of the Ethernet header for the first 100 packets in the log file assigned ID 0 and stored on 'some.machine.com'.

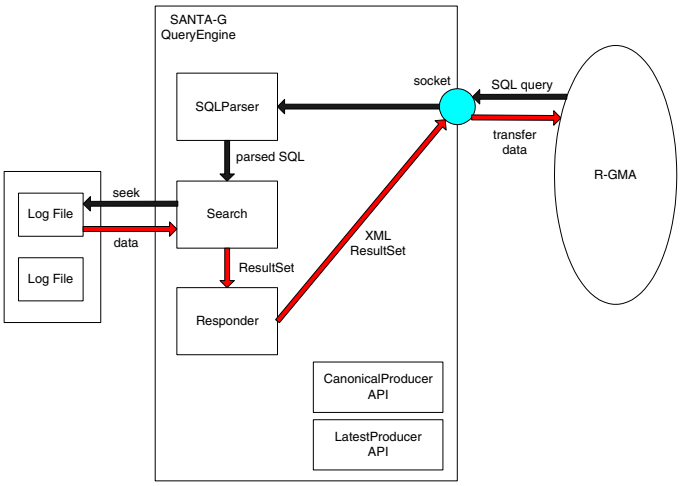

Fig. 7. SANTA-G QueryEngine Query Processing

The SANTA-G Viewer provides a graphical user interface, which makes use of the R-GMA Consumer API, to allow users to graphically view network packets in the log files, and also to build and submit SQL queries that will be carried out on the log files.

V. Conclusion

The R-GMA is a relational implementation of the GMA architecture. It is built using servlet technology. In the R-GMA most producers of information publish data by transferring the data to servlets. This may not always be suitable for all applications. When dealing with instruments which produce a large volume of data it may not be practical to convert it all to a tabular storage model nor efficient to transfer the data to the servlets. It may be preferrable to leave the data where it was created, and only transfer it across the network when specifically requested by a user. In order to allow for this a special type of producer was included in R-GMA, the CanonicalProducer. This allows a user to customize the way the producer responds to a user request, i.e. a SQL query. The SANTA-G network monitoring tool developed within the CrossGrid project demonstrates the CanonicalProducer concept by publishing Ethernet trace data.

References

[1] Andrew Cooke, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Ari Datta, Roney Cordenonsi, Rob Byrom, Laurence Field, Steve Hicks, Manish Soni, Antony Wilson, Xiaomei Zhu, Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Norbert Podhorszki, Brian Coghlan, Stuart Kenny David O'Callaghan, John Ryan. RGMA: First Results After Deployment CHEP03, La Jolla, California, March 24-28, 2003.

[2] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. *A Grid monitoring architecture.* Global Grid Forum Performance Working Group, March 2000. Revised January 2002.

[3] Andy Cooke, Alasdair J G Gray, Lisha Ma, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Rob Byrom, Laurence Field, Steve Hicks, Jason Leake, Manish Soni, Antony Wilson, Roney Cordenonsi, Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Norbert Podhorszki, Brian Coghlan Stuart Kenny, David O'Callaghan. *R-GMA: An Information Integration System for Grid Monitoring* Proceedings of the Tenth International Conference on Cooperative Information Systems, 2003.

[4] Brian Coghlan, Abdeslem Djaoui, Steve Fisher, James Magowan, Manfred Oevers. *Time, Information Services and the Grid* 31st May 2001.
[5] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, P. Vanderbilt. *Grid Service Specification* http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-04_2002-10-04.pdf, 2003.
[6] The DataGrid Project. http://www.eu-datagrid.org
[7] DataGrid WP3. *DataGrid Information and Monitoring Final Evaluation Report* https://edms.cern.ch/document/410810/4/DataGrid-03-D3.6-410810-4-0.pdf
[8] Brian Coghlan, Stuart Kenny. *SANTA-G Software Design Document*
[9] Brian Coghlan, Stuart Kenny. *SANTA-G First prototype Description* http://www-eu-crossgrid.org/Deliverables/M12pdf/CG3.3.2-TCD-D3.3-v1.1-SANTAG.pdf
[10] CrossGrid WP3. *Deliverable D3.5, Report on the Results of the 2nd and 3rd Prototype* http://www-eu-crossgrid.org/Deliverables/M24pdf/CG3.0-D3.5-v1.2-PSNC010-Proto2Status.pdf
[11] The CrossGrid Project http://www.eu-crossgrid.org
[12] DataGrid WP3 Information and Monitoring Services http://hepunx.rl.ac.uk/edg/wp3/
[13] Global grid forum. http://www.ggf.org
[14] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 2: Computational Grids, pages 15–51. Morgan Kaufmann, 1999.
[15] I. Foster, C. Kesselman, and S. Tuecke. *The anatomy of the Grid: Enabling scalable virtual organization.* The International Journal of High Performance Computing Applications, 15(3):200–222, 2001.
[16] Globus Toolkit. http://www.globus.org