TASK 3.3 SOFTWARE REQUIREMENTS SPECIFICATION

DRAFT

Task 3.3 Grid Monitoring

Document Filename:	**CG-3.3-DOC-0001-SRS**
Work package:	**WP3 New Grid Services and Tools**
Partner(s):	**TCD, CYFRONET, ICM**
Lead Partner:	**TCD**
Config ID:	**CG-3.3-TEM-0000-0-8-DRAFT-A**
Document classification:	**PUBLIC**

Abstract: This document specifies the software requirements for CrossGrid Task 3.3 'Grid Monitoring'.

Delivery Slip

	Name	Partner	Date	Signature
From				
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Author
0-0-8-DRAFT-A	9 Mar 2002	Draft version	Brian Coghlan and Bartosz Balis

CONTENTS

1. INTRODUCTION 4

1.1. PURPOSE 4
1.2. SCOPE 4
1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 4
1.4. REFERENCES 5
1.5. OVERVIEW 5

2. OVERALL DESCRIPTION 6

2.1. PRODUCT PERSPECTIVE 6
2.1.1. System interfaces 10
2.1.2. User interfaces 10
2.1.3. Hardware interfaces 10
2.1.4. Software interfaces 10
2.1.5. Communications interfaces 10
2.1.6. Memory constraints 11
2.1.7. Operations 11
2.1.8. Site adaptation requirements 11
2.2. PRODUCT FUNCTIONS 11
2.2.1. APPLICATION MONITORING (OCM-G) 12
2.2.2. NON-INVASIVE MONITORING (SANTA-G) 14
2.2.3. Jiro-BASED MONITORING 15
2.3. USER CHARACTERISTICS 16
2.4. CONSTRAINTS 16
2.5. ASSUMPTIONS AND DEPENDENCIES 17
2.6. APPORTIONING OF REQUIREMENTS 20

3. SPECIFIC REQUIREMENTS 21

3.1. EXTERNAL INTERFACES 21
3.2. FUNCTIONS 21
3.2.1. OCM-G FUNCTIONS 22
3.2.2. SANTA-G FUNCTIONS 23
3.2.3. Jiro-BASED MONITORING FUNCTIONS 25
3.3. PERFORMANCE REQUIREMENTS 26
3.4. LOGICAL DATABASE REQUIREMENTS 26
3.5. DESIGN CONSTRAINTS 26
3.6. STANDARDS COMPLIANCE 26
3.7. SOFTWARE SYSTEM ATTRIBUTES 27

4. APPENDIXES 28

4.1. MDS SCHEMA 28
4.1.1. Top level schema definition in GOS form 28
4.1.2. WP1 schema in GOS form 28
4.1.3. WP5 schema in GOS form 30
4.1.4. WP6 schema in GOS form 32
4.1.5. WP7 schema in GOS form 32
4.2. R-GMA 35
4.2.1. R-GMA ARCHITECTURE 35
4.2.2. R-GMA PROTOCOLS 36
4.2.3. R-GMA IMPLEMENTATION 37

5. INDEX 38

1. INTRODUCTION

1.1. PURPOSE

This document specifies the software requirements for CrossGrid Task 3.3 'Grid Monitoring', including those for the OCM-G monitoring system for Grid applications, the additional services for non-invasive monitoring, and the Jiro-based services for Grid-infrastructure monitoring. The intended audience is both the Task itself and and dependent tasks.

1.2. SCOPE

This task will extend the Grid information system content to include three of the major sources of performance data: applications, instruments and infrastructure.

The products of Task 3.3 are:

(a) an OMIS-based application monitoring system, OCM-G,
(b) additional services, SANTA-G, for non-invasive monitoring, and
(c) Jiro-based services for Grid-infrastructure monitoring.

OCM-G is a distributed monitoring system for obtaining information on and manipulating parallel distributed applications. The purpose of this sytem is to provide a basis for building tools supporting parallel application development. The benefit of using it is that it constitutes an autonomous monitoring infrastructure accessible via a standardised interface, on top of which various tools can be based. This approach provides *abstraction* and increases *modularity* – the tools themselves can be developed independently from the monitoring system.

SANTA-G services are a specialized non-invasive complement to other more intrusive monitoring services. The application of these services will be in performance analysis as well as calibration of both intrusive monitoring systems and systemic models. The objectives are to allow information captured by external monitoring instruments to be introduced into the Grid information system, and to support analysis of performance using this information. The benefits are that other users of the Grid information system can then access this detailed performance data.

The Jiro-based services for Grid-infrastructure monitoring are intelligent components for obtaining information from and manipulating Grid hardware devices. The application of the software is to gather information from hardware devices, make autonomous decisions based on this information and take necessary actions. The objectives are to allow user specify desirable logic for managing hardware. The benefits are that management effort is transferred (partly) from the user to the system.

The addition of these three components will greatly expand the quality and quantity of the Grid information system content.

1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

CrossGrid	The EU CrossGrid Project IST-2001-32243
DataGrid	The EU DataGrid Project <project no.>
Jiro	SUN Jiro

OMIS	On-line Monitoring Interface Specification
OCM-G	Grid-enabled OMIS-Compliant Monitor
RDBMS	Relational Database Management System
R-GMA	DataGrid relational Grid monitoring architecture
SANTA-G	Grid-enabled System Area Network Trace Analysis
SQL	Structured query language
HTTP	Hypertext transport protocol
HTTPS	Secure hypertext transport protocol
XML	Extended markup language

1.4. REFERENCES

CrossGrid	CrossGrid Project Technical Annex *CROSSGRIDANNEX1_V0.1.DOC*
DataGrid	DataGrid Project Technical Annex *DataGridPart_B_V2_51.doc*
Jiro	http://www.jiro.com/
OMIS	*OMIS – On-line Monitoring Interface Specification.* Version 2.0. Lehrstuhl für Rechnertechnik und Rechnerorganisation Institut für Informatik (LRR-TUM), Technische Universität München. http://wwbode.informatik.tu-muenchen.de/~omis/
R-GMA	DataGrid Project Deliverable 3.2 <document no.>
SQL	Structured query language
HTTP	Hypertext transport protocol
XML	Extended markup language

1.5. OVERVIEW

This subsection should
a) Describe what the rest of the SRS contains;
b) Explain how the SRS is organized.

This document provides the software requirements for the OCM-G, SANTA-G and Jiro-based monitoring services.

Section 2 provides …

Section 3 provides ...

2. OVERALL DESCRIPTION

This section of the SRS should describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3 of the SRS, and makes them easier to understand.
This section usually consists of six subsections, as follows:
a) Product perspective;
b) Product functions;
c) User characteristics;
d) Constraints;
e) Assumptions and dependencies;
f) Apportioning of requirements.

2.1. PRODUCT PERSPECTIVE

This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.
A block diagram showing the major components of the larger system, interconnections, and external inter-faces can be helpful.
This subsection should also describe how the software operates inside various constraints. For example, these constraints could include any (maybe all) of the following aspects:

The Grid monitoring services and tools provide information to all dependent subsystems. These subsystems use this information to establish the current and past state of the Grid. This may be to adapt their behaviour accordingly, or to predict future state, or any other similar functionality. The monitoring services and tools gather the information from its sources, and as such are dependent on whatever subsystems host those sources.

In the context of the larger CrossGrid system, there is a primary known: CrossGrid will use the Globus software, and therefore the initial testbed will use the Globus MDS information system, which is due to become obselete at the end of 2002. There is also a primary unknown: how much will CrossGrid use the DataGrid software, and hence how much will it use the DataGrid R-GMA information system? Thus CrossGrid may start with one or two existing information systems, and the monitoring services and tools may or may not use one or both of these. Let us now try to assess the risks involved.

From the dependency tables in Section 2.5 it can be seen that a significant proportion of the information required by the information users is already available via the Globus MDS or DataGrid R-GMA. For example, the DataGrid MDS schema definition (see Appendix) for a computing element includes:

```
CEId :: single-valued, cis,
   {The identifier of the CE}
GlobusResourceContactString :: single-valued, cis,
   {The Globus resource contact string}
GRAMVersion :: single-valued, cis,
   {The GRAM version}
Architecture :: single-valued, cis,
   {The architecture of the hosts composing the CE}
OpSys :: single-valued, cis,
   {The operating system of the hosts composing the CE}
MinPhysicalMemory :: single-valued, integer,
```

```
    {The minimum value of the physical memory among the hosts associated to the CE}
MinLocalDiskSpace :: single-valued, integer,
   {The minimum local disk footprint}
TotalCPUs :: single-valued, integer,
   {The number of total processors associated to the CE}
FreeCPUs :: single-valued, integer,
   {The number of free processors}
NumSMPs :: single-valued, integer,
   {The number of SMP hosts}
MinSPUProcessors :: single-valued, integer,
   {The minimum number of SPU processors (for SMP hosts)}
MaxSPUProcessors :: single-valued, integer,
   {The maximum number of SPU processors (for SMP hosts)}
TotalJobs :: single-valued, integer,
   {The number of jobs submitted to the CE}
RunningJobs :: single-valued, integer,
   {The number of currently running jobs submitted to the CE}
.. and so on ..
```

DataGrid have expended a lot of effort in formulating their MDS schema and populating it for their Testbed 1 (DG-TB1), and in creating a compatible R-GMA (that can also access the base MDS information) that will become the primary information system for their Testbed 2 (DG-TB2). Replication of this seems a waste of time and effort.

Task 3.3 proposes three more ways to create information about the current state of the Grid.

The first (Task 3.3.1) will monitor Grid applications at runtime in an OMIS compliant manner, principally for use at runtime by OMIS compliant tools from Workpackage 2, but also for archival of a selected subset for subsequent post-processing for trend analysis and pattern recognition (thereby creating new prediction result information). The third (Task 3.3.2) will non-invasively monitor Grid components using external instruments, and create a relational trace database of this information, again OMIS compliant, and again for post-processing. A final activity (Task 3.3.3) will monitor infrastructure components using Jiro technology.

The resulting information flows could be as per Figure 2.1.1:

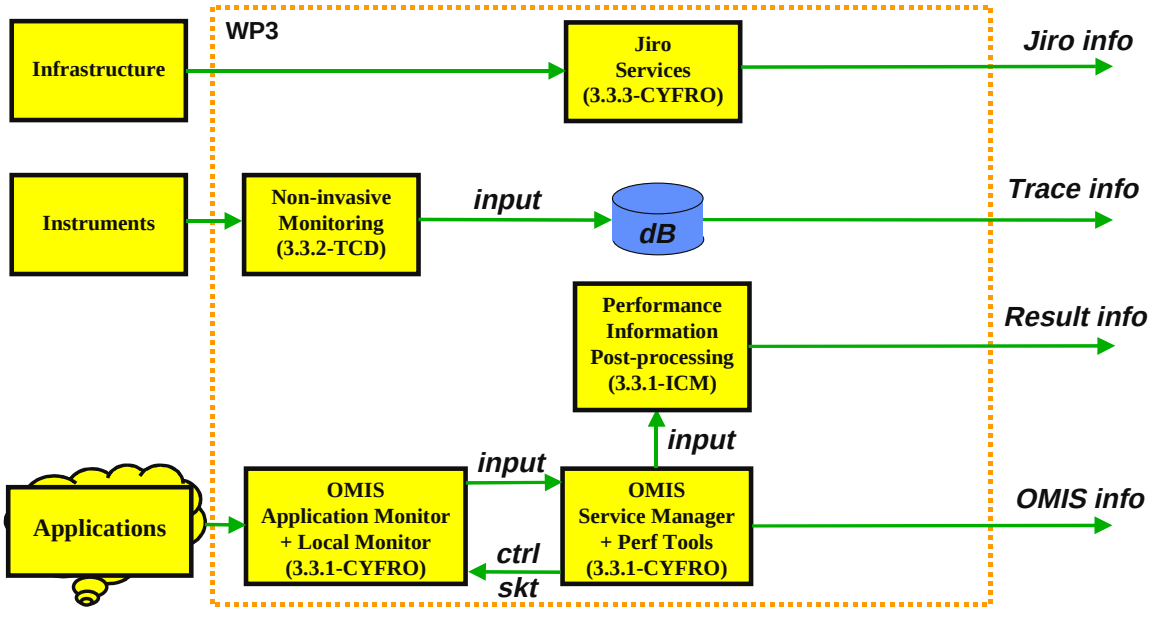

Figure 2.1.1 Internal information flows within Task 3.3

Here we suggest that these flows should supplement the existing information system content rather than involve design of a new information system, i.e. that the effort should concentrate on adding value. Thus Task 3.3 suggests that for the MDS an appropriate extension to the DataGrid MDS schema should be formulated and populated with details of the CrossGrid Testbed 1 (CG-TB1). This should also be done for the R-GMA, again for CG-TB1.

The consequence of this is that, at least for CG-TB1, a significant proportion of the information required by the other tasks and workpackages can then be satisfied by the existing information systems. A further consequence is that these information systems become available for use by the new information sources. The OMIS, result and trace information could be routed through them, and the result and trace information could use the persistent database provisions of the R-GMA. The Jiro information could also be reflected into them, in much the same way as MDS currently is reflected into R-GMA. If these arrangements are made, then the resulting information flows could be as per Figure 2.1.2:

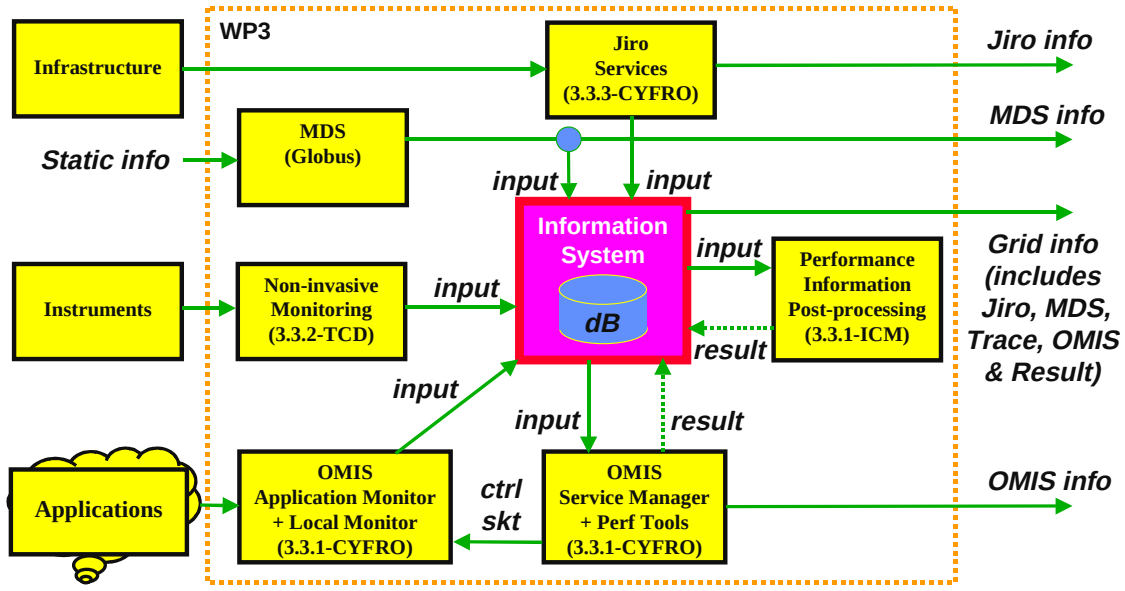

Figure 2.1.2 Can an existing information system be used within Task 3.3 ?

The risks are as follows. Firstly the MDS will be obselete by the end of 2002, therefore we can only consider the R-GMA in relation to the new information sources. Secondly it is likely there will be difficulties of some form, but this applies whatever is proposed, and to be optimistic: ‘where there is a will there is a way’. Thirdly it places great reliance on the DataGrid R-GMA, but this has at least been released in its first revision, DataGrid themselves are keen to assist Task 3.3, and the creation of a larger body of expert R-GMA developers is reassuring. Fourthly the initial R-GMA release is missing many essential functions, in particular the all-important security, a substantive way of finding the best source of an item of information (i.e. a Mediator), a full consideration of scaling and performance, and any measure to guarantee robustness (e.g. replication); all these functions are, however, in the DataGrid development plan. Finally it renders CrossGrid hostage to DataGrid’s timetables, but at least those are 12-15 months further advanced.

At this time, the nett risk might be considered to be acceptable relative to the benefits. The position should be re-evaluated over time as new Grid information systems become available, e.g. OGSA

compliant systems (note that DataGrid plan to modify R-GMA to be OGSA compliant). For the present one should assume exploitation of the DataGrid R-GMA information system.

There is still, however, the question of how to best exploit this for each new information source:

OCM-G

The OCM-G is able to gather information from user applications and provide it to higher level software components, typically tools such as performance analysis ones. Parts of OCM-G are stand-alone components, while other parts reside directly in the application address space. Some parts of OCM-G have rather specialized functional and performance requirements, and may have to be completely independent of the information system.

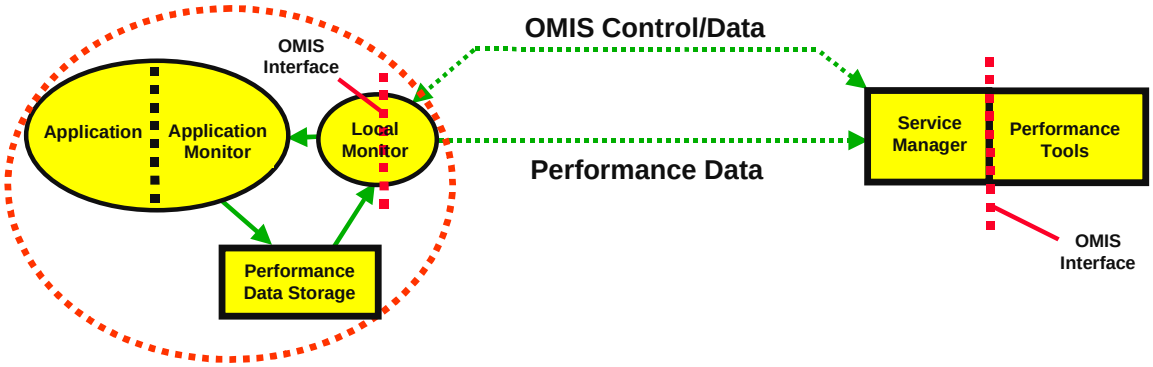

Figure 2.1.3 OCM-G

Figure 2.1 shows the monitoring environment. The OCM-G parts are *Service Manager*, *Local Monitor*, and *Application Monitor*. A service manager is the part of OCM-G to which tools can submit requests and from which they receive replies. Local monitors reside on each node of the target system. A local monitor receives OMIS requests from a service manager and its task is to execute the request and pass the reply back to the service manager. An application monitor is embedded in each process of the monitored application. Its role is to execute performance-critical requests directly in the application context and buffer performance data before passing it to higher layers.

OCM-G shall interoperate with the Grid information system. At this point it is not clear what kind of interoperability this should be. OCM-G will probably use the information system to manage the startup of the monitoring infrastructure. OCM-G might also be enabled to put the monitoring data into the information system if a client demands it. This can be useful for tools that perform statistical observations and need historic data (e.g. ‘what files have been most frequently accessed by an application during its many executions’).

SANTA-G

SANTA-G is specifically intended to introduce information captured by external monitoring instruments into the Grid information system, and to avail of the information system for subsequent performance analysis. All of the SANTA-G services are only dependent on the information system.

Jiro-BASED MONITORING

Jiro is totally independent of the Grid information system; it is composed of stand-alone services. These services are responsible for gathering information from the hardware devices as well as analysing these information and making decisions based on it. Services responsible for gathering information can communicate directly with devices or through some legacy API like SNMP. Services responsible for making autonomous decision let the user specify appropriate logic for it. Moreover, the Jiro information can easily be reflected into it in much the same way as MDS currently is reflected into R-GMA, thereby adding value to the information system content

2.1.1. System interfaces

This should list each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system.

The system interfaces …

2.1.2. User interfaces

This should specify the following:

I. The logical characteristics of each interface between the software product and its users. *This includes those configuration characteristics (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.*

II. All the aspects of optimizing the interface with the person who must use the system. *This may simply comprise a list of do's and don'ts on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be verifiable, e.g., "a clerk typist grade 4 can do function X in Z min after 1 h of training" rather than "a typist can do function X." (This may also be specified in the Software System Attributes under a section titled Ease of Use.)*

The user interfaces …

2.1.3. Hardware interfaces

This should specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by line support.

The hardware interfaces …

2.1.4. Software interfaces

This should specify the use of other required software products (e.g., a data management system, an operating system, or a mathematical package), and interfaces with other application systems (e.g., the linkage between an accounts receivable system and a general ledger system). For each required software product, the following should be provided:

- Name;
- Mnemonic;
- Specification number;
- Version number;
- Source.

For each interface, the following should be provided:

- Discussion of the purpose of the interfacing software as related to this software product.
- Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

The software interfaces …

2.1.5. Communications interfaces

This should specify the various interfaces to communications such as local network protocols, etc.
The communications interfaces …

2.1.6. Memory constraints

This should specify any applicable characteristics and limits on primary and secondary memory.
The memory constraints …

2.1.7. Operations

This should specify the normal and special operations required by the user such as
- The various modes of operations in the user organization (e.g., user-initiated operations);
- Periods of interactive operations and periods of unattended operations;
- Data processing support functions;
- Backup and recovery operations
NOTE: This is sometimes specified as part of the User Interfaces section.

OCM-G

There are three primary operations that can be performed in interaction with OCM-G:

- *Init* open connection to the monitoring system
- *Request* send OMIS request
- *Finalize* close connection to the monitoring system

These operations will be available to clients by means of an appropriate API.

SANTA-G

The operations that can be performed in interaction with SANTA-G are yet to be determined.

Jiro-BASED MONITORING

The operations that can be performed in interaction with Jiro-based monitoring are yet to be determined.

2.1.8. Site adaptation requirements

This should
- Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode (e.g., grid values, safety limits, etc.);
- Specify the site or mission-related features that should be modified to adapt the software to a particular installation.
The data must be …
The initialization sequence is …
Specific sites must …

2.2. PRODUCT FUNCTIONS

This subsection of the SRS should provide a summary of the major functions that the software will perform. For example, an SRS for an accounting program may use this part to address customer account maintenance, customer statement, and invoice preparation without mentioning the vast amount of detail that each of those functions requires.
Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product. Note that for the sake of clarity
a) The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.
b) Textual or graphical methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product, but simply shows the logical relationships among variables.

This section assumes the target information system is the DataGrid R-GMA. This assumption should be re-evaluated over time as new Grid information systems become available.

2.2.1. APPLICATION MONITORING (OCM-G)

OCM-G (Task 3.3.1) will monitor Grid applications at runtime, principally for use at runtime by the tools from Workpackage 2, which assumes the application-level monitoring environment to be developed will comprise an autonomous monitoring system. Efficiency of application monitoring will be ensured by specialised application monitors, embedded in the application address space, and by efficient performance-data storage. The issue of scalability will be addressed by distributing the monitoring system into service managers (an intermediate layer between tools and local monitors), local monitors (for direct control of local application processes), and application monitors (for local handling of performance-critical actions in co-operation with local monitors). The OMIS specification will be used to build the communication layers from tools-to-service managers and service managers-to-local monitors. The system will be extendible to provide new, additional functionality. If the control and information flows are separated then control could be exercised over channels built via Globus sockets (with GSI security), whilst data could flow via the R-GMA. This might be implemented as shown in Figure 2.2.1, although at this stage this is speculation.

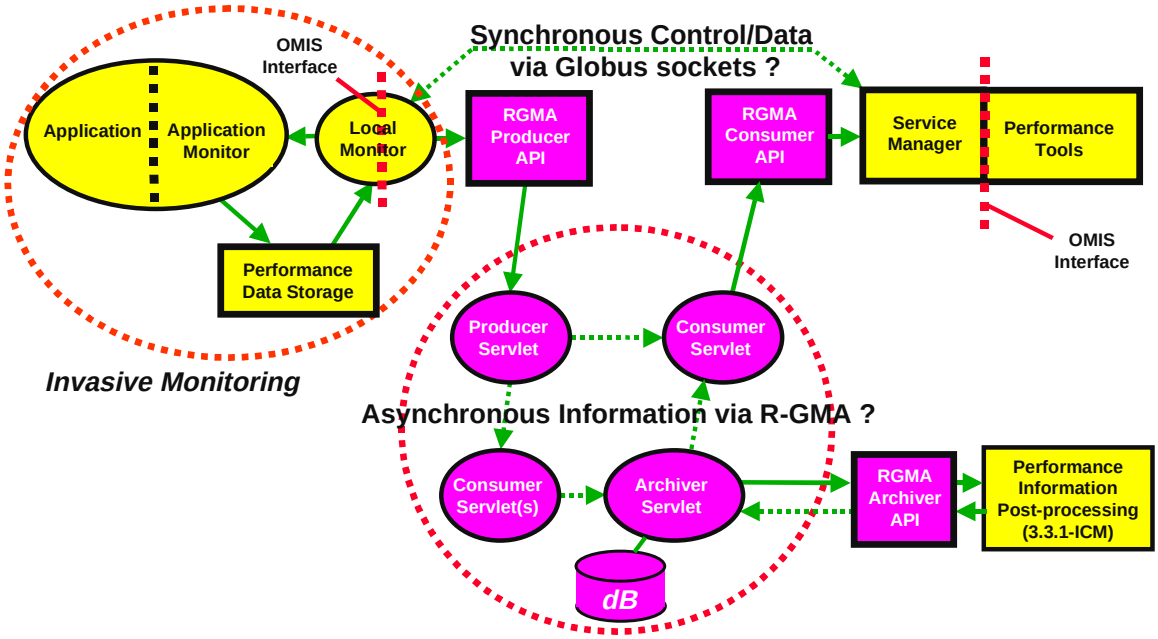

Figure 2.2.1 OCM-G implementation

Note that there may be many Local instances interacting with a single Service Manager, and there may be many Service Managers. Note also that information from the Application Monitor can be stored in logfiles in the Performance Data Storage, and subsequently retrieved via the Local Monitor by issuing appropriate SQL queries to the Producer. Synchronization of control and data could be guaranteed by ensuring the control and data flow through the one executing process, which has both Globus socket and R-GMA interfaces.

Another activity in this subtask will archive a selected subset of this information and post-process it for trend analysis and pattern recognition, and thereby create new prediction result information. Here one or more *Consumer* servlets could be used by an *Archiver* servlet to gather selected information that the Application Monitor has stored in the Performance Data Storage, where again the selection can be specified via SQL queries (again see Fig.2.2.1). An Archiver has a built-in *DBProducer* servlet to store the information persistently using a RDBMS, such as MySQL. The post-processing application could then take its input from the DBProducer and return its results to new tables in the same (or another) DBProducer.

The OCM-G will accept all requests compliant with OMIS 2.0. This includes two type of requests: *unconditional* and *conditional*.

The *unconditional* requests have a single effect on the target system and yield an immediate response. They are composed of one or more *actions* which specify either information to be returned or manipulations to be performed. For example request *stop process p_1* will attempt to stop process identified as *p_1*, while request *return process list for node n_1* will return a list of attached (see section 3.2) processes residing on node identified as *n_1*. All requests return a status information (whether the request succeded or not).

The *conditional* requests specify an event and a list of actions which shall be executed whenever the event occurs. Thus, responses to conditional requests can be produced multiple times, any time the specified event takes place. Example: *when function X is invoked, return the timestamp of this event.*

For a detailed description of OMIS services, syntax and semantics of requests and specification of data format returned from the monitoring system refer to [OMIS].

2.2.2. NON-INVASIVE MONITORING (SANTA-G)

SANTA-G (Task 3.3.2) will non-invasively monitor Grid components using external instruments, and create a relational trace database of this information. This is especially desirable for validation of invasive toolsets and simulation models. External sensor instruments will generate local logfiles that can be accessed via the DataGrid R-GMA. Specific R-GMA components will be needed to produce and consume the non-invasive performance monitoring information, as also will be associated performance analysis tool support. This might be implemented as shown in Figure 2.2.2. This arrangement will use the R-GMA to gather selected information from the logfiles, and store it persistently in a database. A subsequent analysis will take its input from the R-GMA and return its results to the R-GMA. An OMIS-compliant interface will enable interaction with the OCM-G toolset.

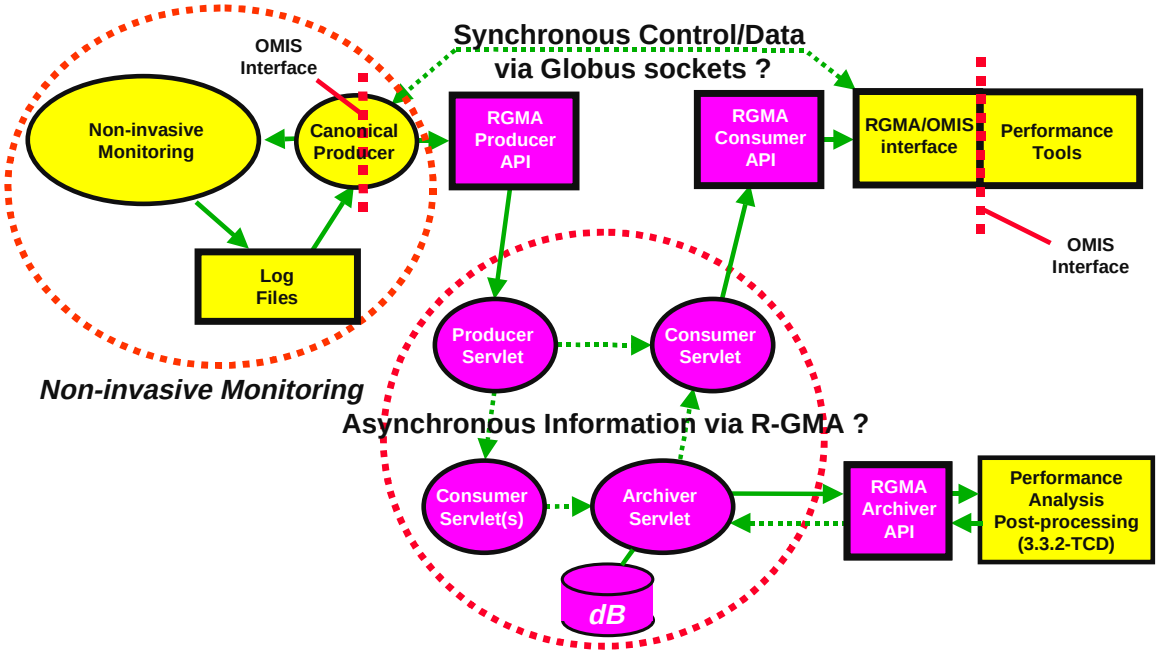

Figure 2.2.2 SANTA-G implementation

Thus SANTA-G tools will provide a set of services for non-invasively monitoring grid resources and subsequent performance analysis. This assumes that source of monitoring data will be a set of specialized, typically commercially available, instruments. Most relevant trace instrument hardware is controllable by suitable software. Acquisition of only the desired information, from within the complete set of information being generated, is usually determined by setting appropriate filter and trigger patterns. For a network trace, for example, these may specify packet types, source and destination addresses, etc.

A relational database will be employed to store and analyse the trace data acquired by the instrument. Since the raw trace data collected by the instrument will be written to binary trace files, these may need to be pre-processed into a format which is suitable for insertion into the database.

A number of useful attributes can be added by the trace hardware during tracing to assist subsequent analysis of the data. Each monitoring run will associated with a particular trace identifier, allowing for the identification of individual monitoring sessions; this will form part of the primary key. A timestamp will be appended to the data during the non-intrusive acquisition of the trace data. The database will make these timestamps available for further analysis - consequently trace data will provide accurate information about the temporal behaviour of the system under test. An example of some of the fields provided in a database for a network trace could be:

PKT.packet_id	*Ethernet packet identifier*
PKT.timestamp_s	*integer timestamp part*
PKT.timestamp_us	*fractional timestamp part*
ETH.destination_address	*Ethernet destination address*
ETH.source_address	*Ethernet source address*

Subsequent to the trace data acquisition, R-GMA components will be used to provide for the persistent storage of the trace data. The trace files will be accessed by using a *Producer* (in this case the *CanonicalProducer*, which is being developed by TCD for DataGrid). This will then in turn use a R-GMA *Archiver* to store the trace data in a relational database.

In order to access this data, users will also make use of R-GMA *Consumers* and the Archiver. Users will create consumers, specifying SQL query statements for the data they wish to collect. The consumers will contact the Archiver, which will return the result set.

Further functions will provide performance analysis post-processing. This will be done using the same Archiver, from which the performance analysis tools will be able to retrieve the data required.

Interaction with OCM-G will be via an OMIS-compliant interface, supporting a subset of the functionality described in Section 2.2.1.

2.2.3. Jiro-BASED MONITORING

A final activity (Task 3.3.3) will monitor infrastructure components using Jiro technology, which is a Java language-based implementation of the Federated Management Architecture. Jiro is designed for enterprise-sized applications, and supports network management protocols such as SNMP, thus it seems to be an appropriate approach to the network-related issues within Grid monitoring. Jiro components, such as the Logging Service, will be used for infrastructure monitoring, i.e. collecting information about user and node activity, network and node loads and network configuration tracing. Therefore this approach will complement the OMIS view. An important feature of the approach is that the management components can communicate with managed objects using SNMP, WBEM or some proprietary protocols. This enables one to create a solution that can leverage the existing "legacy" systems without a need to convert them to Java, so the proposed solution is flexible and extensible. The Jiro information can be published into the R-GMA, in the same way as with MDS. This may be implemented as shown in Figure 2.2.3.

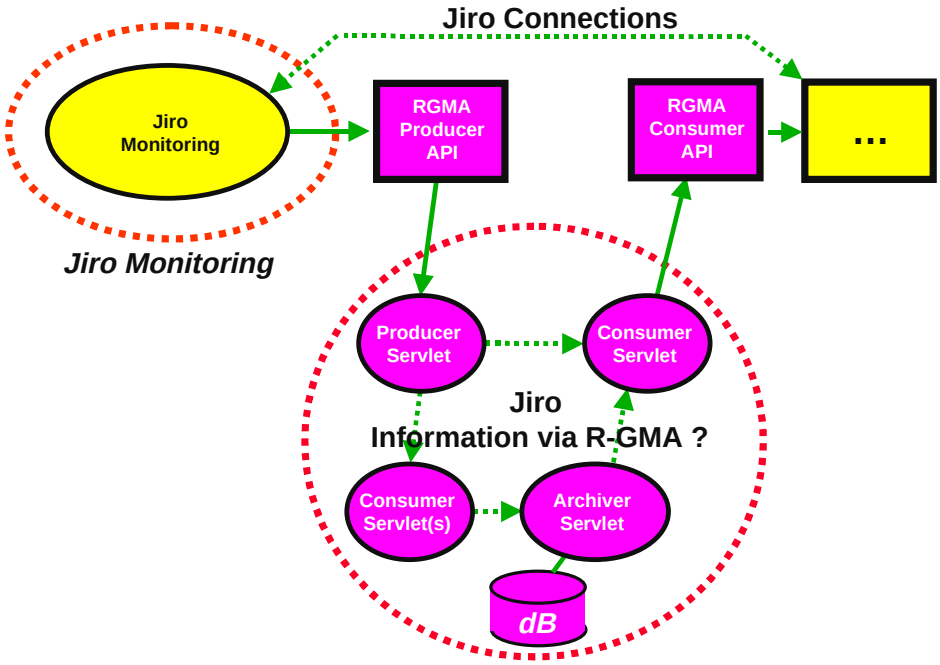

Figure 2.2.3 Jiro-based monitoring implementation

<<< COMMENTS GRATEFULLY RECEIVED >>>

2.3. USER CHARACTERISTICS

This subsection of the SRS should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. It should not be used to state specific requirements, but rather should provide the reasons why certain specific requirements are later specified in Section 3 of the SRS.

The user characteristics are …

2.4. CONSTRAINTS

Contraints include
a) Regulatory policies …
b) Hardware limitations (e.g., signal timing requirements) …
c) Interfaces to other applications …
d) Parallel operation …
e) Audit functions …
f) Control functions …
g) Higher-order language requirements …
h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK) …
i) Reliability requirements …
j) Criticality of the application …
k) Safety and security considerations …

2.5. ASSUMPTIONS AND DEPENDENCIES

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

Firstly let us look at the kinds of information that will be required both within Workpackage 3 and in other workpackages. From the CrossGrid Technical Annex, the information flows involving Task 3.3 and other Workpackages are confined to interactions with Workpackage 2 as in Figure 2.5.1.

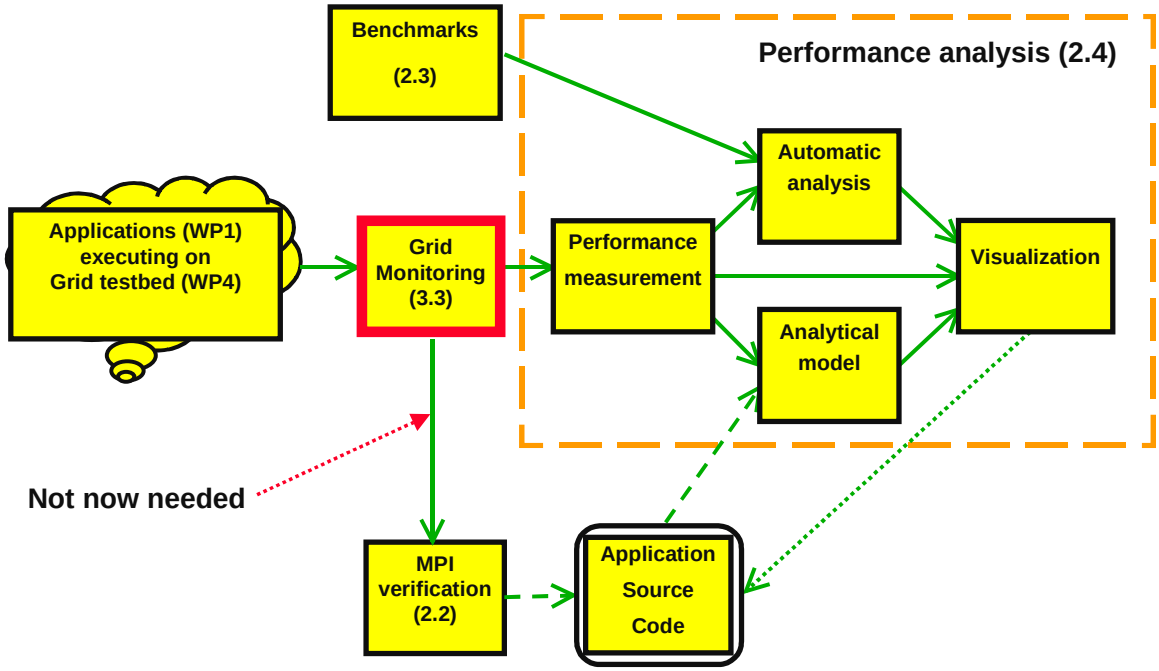

Figure 2.5.1 Information flows between Task 3.3 and other WPs as per the Technical Annex

Again from the CrossGrid Technical Annex, the information flows between Task 3.3 and the remainder of Workpackage 3 are as in Figure 2.5.2.

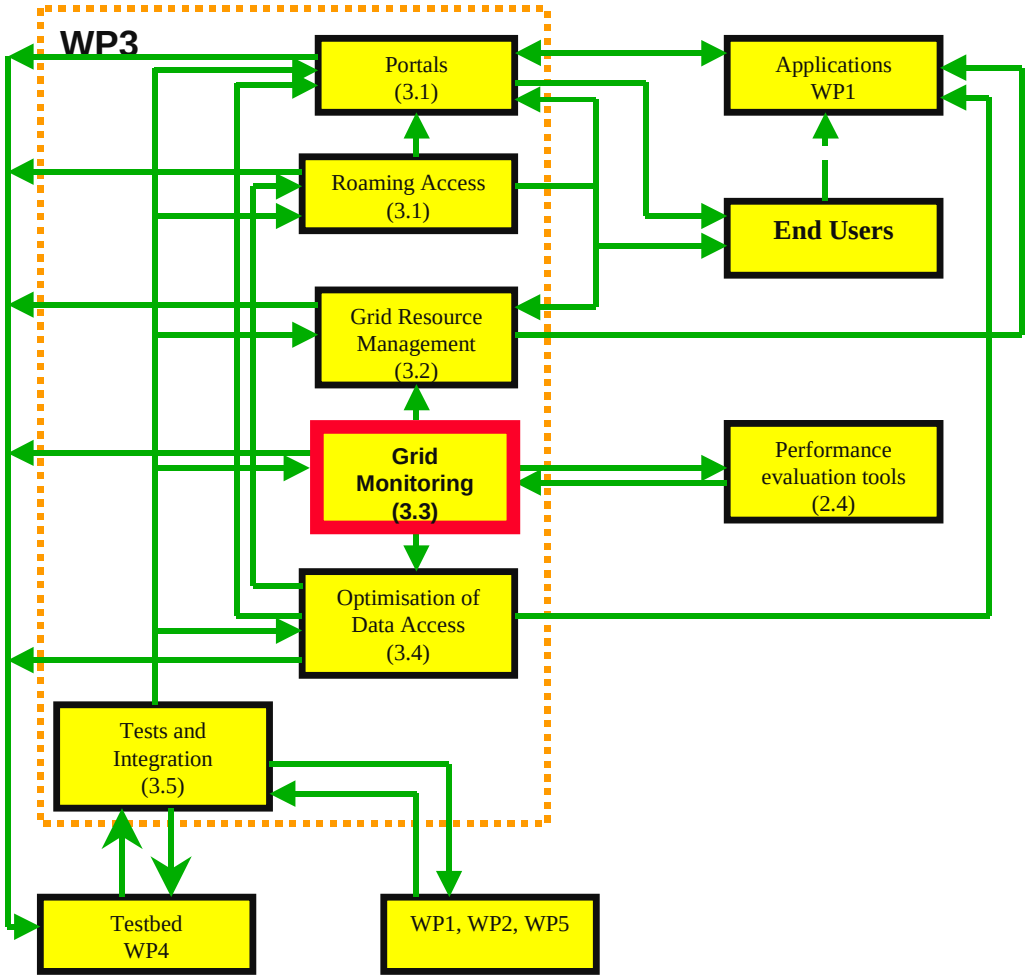

Figure 2.5.2 WP3 internal information flows as per the Technical Annex

Figure 2.5.3 sumarises the known external information flows involving Task 3.3.

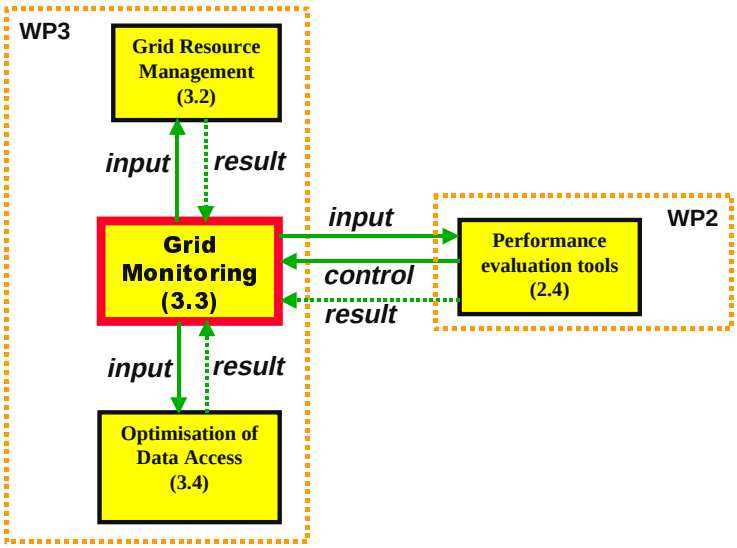

Figure 2.5.3 Summary of internal information flows involving Task 3.3

The depencency tables below indicate where information may be most easily acquired from: either the Globus MDS, the DataGrid R-GMA, or unique Task 3.3 sources.

Task	Flow	Information	Source
T1.x [Applications]	← T3.3	unknown	
	→ T3.3	test objects software environment (runtime libraries, etc) software requirements specification ? other info ? no critical dependency	JDL? / MDS?
	Queries:	what info do T1.x need ? What info do T1.x publish ?	
T2.2 [MPI Verification]	← T3.3	what info ?	
	→ T3.3	outputs software requirements specification ? no critical dependency	
	Queries:	what input from T3.3 ?	
T2.3 [Benchmarks]	← T3.3	Nil	
	→ T3.3	Nil	
	Queries:	what input from T3.3 ?	
T2.4 [Performance Evaluation Tools]	← T3.3	Data transfers Synchronization I/O delay CPU utilization Network utilization Storage utilization (Raw?) data [for automatic perf analysis] other info ?	R-GMA? T3.3? R-GMA? R-GMA R-GMA MDS / R-GMA T3.3 ?
	→ T3.3	software requirements specification	
	Queries:	what other info might be required from T3.3 ?	
T3.1 [Portals]	← T3.3	Unknown	
	→ T3.3	Unknown No critical dependency	
	Queries:	where does T3.1 get its info from ?	
T3.1 [Roaming]	← T3.3	Unknown	
	→ T3.3	Unknown No critical dependency	
	Queries:	where does T3.1 get its info from ?	
T3.2	← T3.3	Heterogeneity Load average Availability Location of data files Availability of replicas Network delays Static application characteristics Dynamic application characteristics Architecture OS Static memory config Dynamic memory config	MDS R-GMA MDS RC / MDS RC / MDS MDS / R-GMA JDL T3.3 MDS MDS MDS R-GMA / T3.3

		Resident libraries Required libraries Invoked shared libraries Other infrastructure Configuration History [+ predictions]	MDS JDL T3.3 T3.3 T3.3 T3.3
	→ T3.3	Outputs initial location of processes (to locate processes) migration decisions (to locate monitored processes) [predictions ?] no critical dependency	
	Queries:	Nil as yet	
T3.4 [Optimization of Data Access]	← T3.3	Current user state I/O info Network info MSMS load Queue length Number of drives Drive throughput File size	T3.3? R-GMA? / T3.3 ? R-GMA T3.3? T3.3? MDS T3.3? T3.3?
	→ T3.3	outputs no critical dependency	
	Queries:	current user state ???	
T3.5 [Prototypes]	← T3.3	Specifications User guide Anything else ?	
	→ T3.3	Something ?	
	Queries:	what input from T3.5 ?	

2.6. APPORTIONING OF REQUIREMENTS

This subsection of the SRS should identify requirements that may be delayed until future versions of the system.

3. SPECIFIC REQUIREMENTS

This section of the SRS should contain all of the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output. As this is often the largest and most important part of the SRS, the following principles apply:
a) Specific requirements should be stated in conformance with all the characteristics described in 4.3.of [IEEE-Std 830-1998].
b) Specific requirements should be cross-referenced to earlier documents that relate.
c) All requirements should be uniquely identifiable.
d) Careful attention should be given to organizing the requirements to maximize readability.
Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in 3.1 through 3.7.

3.1. EXTERNAL INTERFACES

This should be a detailed description of all inputs into and outputs from the software system. It should complement the interface descriptions in 2 and should not repeat information there. It should include both content and format as follows:
a) Name of item;
b) Description of purpose;
c) Source of input or destination of output;
d) Valid range, accuracy, and/or tolerance;
e) Units of measure;
f) Timing;
g) Relationships to other inputs/outputs;
h) Screen formats/organization;
i) Window formats/organization;
j) Data formats;
k) Command formats;
l) End messages.

3.2. FUNCTIONS

Functional requirements should define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as "shall" statements starting with "The system shall…" .

These include

a) Validity checks on the inputs
b) Exact sequence of operations
c) Responses to abnormal situations, including
1) Overflow
2) Communication facilities
3) Error handling and recovery
d) Effect of parameters
e) Relationship of outputs to inputs, including
1) Input/output sequences
2) Formulas for input to output conversion

It may be appropriate to partition the functional requirements into subfunctions or subprocesses. This does not imply that the software design will also be partitioned that way.

3.2.1. OCM-G FUNCTIONS

The target system will be viewed by the OCM-G as a hierarchical set of objects. These will include nodes, processes, threads, etc. The exact object hierarchy for Grid <TBD>. The objects will be identified by unique *tokens*, for example *p_1*, *n_1*, etc.

The OCM-G will provide a set of monitoring services for requesting information from and doing manipulations on an application. A typical monitoring session between a tool and the monitoring system consists of the following stages (from the tool's viewpoint):

- Connect to the monitoring system
- Attach to objects (typically application processes) you want to monitor
- Send monitoring requests (and process returned information)
- Disconnect from the monitoring system

Thus, each tool has its associated *scope* of the system being observed, i.e., the set of objects it wants to monitor. Monitoring services can be requested only on the explicitly attached objects, otherwise the request is erronous.

For each monitoring request, a list of objects is is always specified to which the request should be applied to. For example *Stop processes [p_1, p_2, p_3]*.

The monitoring system ensures that:

- The request is distributed to each object involved
- The returned data is assembled and returned to a tool as a single reply

For exmaple, request *Return CPU load on nodes n_1, n_2, n_3* will return a list of three elements containing information on CPU load on nodes n_1, n_2, and n_3.

To support performance analysis of applications, OCM-G can detect and associate actions with the following two events:

1. *Invocation of function X has started*
2. *Invocation of function X has ended*

These two events can be combined with proper actions to obtain performance measurements such as the volume of data transferred between processes or the delay of data transmission. Example of a complex request: *When invocation of function X has started in process [p_1, p_2, p_3] return the token of the process in which the event took place and the time stamp of the occurrence.* This request is applied to processes identified as *p_1*, *p_2* and *p_3*, whereas the result of each execution will be the particular process identifier (i.e. one of *p_1*, *p_2*, *p_3*) and a timestamp.

To support efficient performance analysis, the OCM-G will provide:

1. Ability to buffer gathered data in the application's address space and request it on demand.

2. Efficient data structures to store preprocessed data.
   - *Counters* for storing information representable on integer values.
   - *Integrators* for storing information that needs floating-point representation.

Counters and integrators are objects which can be created in both local (application process) and global (monitoring session) contexts, and will be idenfiable by appropriate tokens (e.g. *c_g_1*, *c_l_1*). Various operations on these objects will be defined (e.g. *Increment counter c_g_1 by n*).

<MORE – TBD>

3.2.2. SANTA-G FUNCTIONS

SANTA-G services are based on a relational model of the acquired information, supported by the DataGrid R-GMA. See Appendix for a brief description of the R-GMA functionality and terminology.

In response to filter and trigger patterns the relevant trace instrument will collect appropriate data into trace files. These trace files may require pre-processing, which will be carried out by software running on the node that hosts these files.

These trace files will be accessed by a *sensor application*, also running on that host, which will contact a *CanonicalProducer Servlet* in order to create the necessary *Canonical Producers* [CP, one for each table], to represent the database structure. A CP is created by specifying a SQL create table statement, which defines the table the producer provides. Two sensor applications will be defined, one for network tracing and another for SCI [IEEE-1596 Scalable Coherent Interface].

For network tracing, the CP tables will be created by a SQL statement of the form:

```
CREATE TABLE  PKT (
        pkt_id …
        file_id …
        ts_s …
        ts_us …
        cap_len …
        act_len …
);
```

For SCI tracing, the SQL statement will be of the form:

```
CREATE TABLE  SCI_Blink (
        TraceId …
        PacketId …
```

code_sendId …
code_sinkId …
code_vd …
code_transId …
priority …
post_reserved …
post_less …
post_bad …
post_crcc …
reserved …
parity …
);

The full definition of these tables is yet to be determined.

The R-GMA will enter each CP in a R-GMA *Registry*, which will allow *Consumers* to locate it. This action is hidden within the R-GMA.

A SANTA tool running on another host can now use an R-GMA *Archiver* to acquire data. It calls the Archiver API, which contacts an Archiver servlet. In the spirit of R-GMA, a SQL query statement is used to define what data is to be acquired. For network tracing the SQL query [SELECT] will be of the form:

SELECT PKT.pkt_id,
PKT.file_id,
PKT.ts_s,
PKT.ts_us,
PKT.cap_len,
PKT.act_len
FROM PKT;

The Archiver uses one or more internal consumers, each of which contacts a R-GMA registry to locate the CPs which hold the relevant tables. The consumers then query these CPs. Each CP in response to this query contacts a *CanonicalProducer Query Engine*, which runs on the node that hosts the trace files. All these propagate the SQL SELECT statement, or a derivative thereof. It is the Query Engine that actually performs seek operations on the trace files and returns the data in the form of a result set to the CP, and thence to the Archiver's consumer. The data can now be inserted into the Archiver's *DBProducer* and persistently stored in a RDBMS database, where it can be accessed by further SANTA tools.

In cases where the entire logfiles are transferred, unfiltered, to the Archiver, it may be sensible to reduce this generic process to simply using a standalone DBProducer, rather than an Archiver.

Other SANTA tools may now access the trace database using their own consumer interface. Again, a SQL query (a SELECT statement) is used to define what information is to be accessed. A call is made to the Consumer API, which contacts a Consumer Servlet. The servlet contacts a R-GMA registry to locate the Archiver which holds the trace database. The consumer then queries the Archiver, which in turn queries the RDBMS database and returns the result set via the R-GMA to the SANTA tool.

Thus the trace data becomes an integral part of the Grid information system content.

3.2.3. Jiro-BASED MONITORING FUNCTIONS

The Jiro-based monitoring systems consist of two layers:

- *Management Facade* components which are Jiro services representing hardware devices
- *Management Policies* components which are Jiro services holding management logic

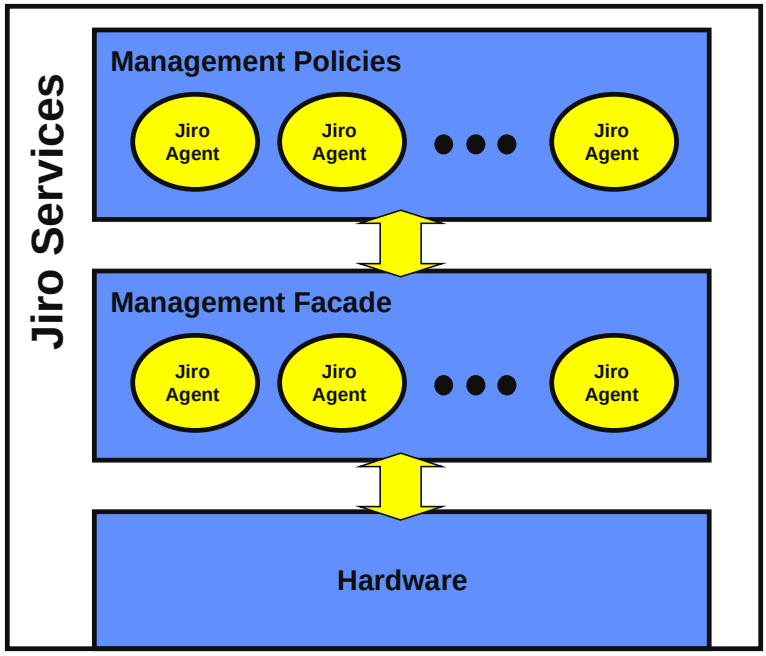

Figure 3.2.1 Jiro monitoring services

Management Facade services should be developed for all the types of hardware devices which exist on the Grid. Services can interact directly with hardware or through SNMP, WBEM, JDMK or other APIs. Users can get information directly from these services.

Management Policies services provide general configurable components for embedding management logic. Users are able to define logic using *sensors*, which provide *information* and *actions*. Sensors are Management Facade services. Actions are defined in the system. Users are able to write conditions on which actions take place.

Below is an example of typical scenario:

- A user creates a Management Policies component with logic: “computer1.freeSpace < 100 then sendEvent(‘warnings.freespace)’”, where *computer1* is the name of a Management Facade service, *freeSpace* is the name of a parameter, and *sendEvent* is the name of an action which accepts one parameter named *topic*. The sendEvent action sends an event using the Jiro *Event Service*.
- The Management Policies work on one of the Jiro *Management Stations*

If the free space on the disk reaches a threshold value an event is sent to the user.

3.3. PERFORMANCE REQUIREMENTS

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole. Static numerical requirements may include the following:
a) The number of terminals to be supported;
b) The number of simultaneous users to be supported;
c) Amount and type of information to be handled.
Static numerical requirements are sometimes identified under a separate section entitled Capacity.
Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.
All of these requirements should be stated in measurable terms.
For example,
95% of the transactions shall be processed in less than 1 s.
rather than,
An operator shall not have to wait for the transaction to complete.
NOTE: Numerical limits applied to one specific function are normally specified as part of the processing subparagraph description of that function.

3.4. LOGICAL DATABASE REQUIREMENTS

This should specify the logical requirements for any information that is to be placed into a database. This may include the following:
a) Types of information used by various functions;
b) Frequency of use;
c) Accessing capabilities;
d) Data entities and their relationships;
e) Integrity constraints;
f) Data retention requirements.

3.5. DESIGN CONSTRAINTS

This should specify design constraints that can be imposed by other standards, hardware limitations, etc.

3.6. STANDARDS COMPLIANCE

This subsection should specify the requirements derived from existing standards or regulations. They may include the following:
a) Report format;
b) Data naming;
c) Accounting procedures;

d) Audit tracing.
For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

3.7. SOFTWARE SYSTEM ATTRIBUTES

There are a number of attributes of software that can serve as requirements. It is important that required attributes be specified so that their achievement can be objectively verified. A partial list of examples can include: reliability, availability, security, maintainability and portability.

4. APPENDIXES

4.1. MDS SCHEMA

For quick reference, below is included the DataGrid schema definitions for the MDS as downloaded from http://marianne.in2p3.fr/datagrid/documentation/MDS-Deployment-PM9-WP3.doc on 2-FEB-2002.

Top level schema and schema required by the various workpackages have been delivered for inclusion in the DataGrid Testbed-1.

Although they are defined as belonging to various workpackages, many are based on the requirements of more than 1 workpackage. For example, the Storage Element schema, defined as WP5 contains fields specified as required by WP1.

The schema for month 9 have been defined using GOS.

A tool has been provided to convert this to ldap – called gos2ldap, which also systematically 'fills in' the OID. This tool is provided as part of the DataGrid WP3 delivery, by the WP3 team. (It is not supplied by Globus.)

In some cases, after the gos2ldap script had produced the schema files, some other fields were found to be needed. In this case, the fields were added by hand. If fields needed to be removed, they were removed by hand. This is so that the OID's of other fields did not change.

If the standard installation procedures are followed the schema files are located in

/opt/edg/info/mds/schema

Below are the Schema delivered at M9.

4.1.1. Top level schema definition in GOS form

```
NAMESPACE DataGrid {

OBJECTCLASS DataGridTop  {
        OID     { ? }
       DESCRIPTION { Base class for DataGrid }
       KIND { ABSTRACT }
       INHERITSFROM { Top }
}
}
```

4.1.2. WP1 schema in GOS form

The schema delivered by WP1 specify the Computing element. The storage element which is close to the computing element is also defined, this is the CloseStorageElement.

```
NAMESPACE DataGrid {
  OBJECTCLASS ComputingElement {
  OID {??}
  DESCRIPTION {
```

```
            A Computing Element for the DataGrid Project
}
KIND { STRUCTURAL }
INHERITSFROM { DataGridTop }
KEY {CEId}
MUSTCONTAIN {
  CEId :: single-valued, cis,
     {The identifier of the CE}
  GlobusResourceContactString :: single-valued, cis,
     {The Globus resource contact string}
  GRAMVersion       :: single-valued, cis,
     {The GRAM version}
  Architecture :: single-valued, cis,
     {The architecture of the hosts composing the CE}
  OpSys :: single-valued, cis,
     {The operating system of the hosts composing the CE}
  MinPhysicalMemory :: single-valued, integer,
      {The minimum value of the physical memory among the hosts associated to
       the CE}
  MinLocalDiskSpace :: single-valued, integer,
     {The minimum local disk footprint}
  TotalCPUs :: single-valued, integer,
     {The number of total processors associated to the CE}
  FreeCPUs :: single-valued, integer,
     {The number of free processors}
  NumSMPs :: single-valued, integer,
     {The number of SMP hosts}
  MinSPUProcessors :: single-valued, integer,
     {The minimum number of SPU processors (for SMP hosts)}
  MaxSPUProcessors :: single-valued, integer,
     {The maximum number of SPU processors (for SMP hosts)}
  TotalJobs :: single-valued, integer,
     {The number of jobs submitted to the CE}
  RunningJobs :: single-valued, integer,
     {The number of currently running jobs submitted to the CE}
  IdleJobs :: single-valued, integer,
     {The number of idle jobs submitted to the CE}
  MaxTotalJobs :: single-valued, integer,
     {The maximum number of jobs (running and idle) allowed for the CE}
  MaxRunningJobs :: single-valued, integer,
     {The maximum number of running jobs allowed for the CE}
  WorstTraversalTime :: single-valued, cisfloat,
     {The worst traversal time for jobs submitted to the CE}
  EstimatedTraversalTime :: single-valued, cisfloat,
     {The estimated traversal time for jobs submitted to the CE}
  Active :: single, boolean,
     { Defines if the CE is active}
  Priority    :: single-valued, integer,
     {The priority associated to the CE}
  MaxCPUTime :: single-valued, cisfloat,
     {The maximum CPU time allowed for jobs submitted to the CE}
  MaxWallClockTime :: single-valued, cisfloat,
     {The maximum wall clock time allowed for jobs submitted to the CE}
  AverageSI00 :: single-valued, cisfloat,
     {The average of the SpecInt2000 benchmark of the nodes associated to the
       CE}
  MinSI00 :: single-valued, cisfloat,
     {The minimum value of the SpecInt2000 benchmark of the nodes associated
       to the CE}
  MaxSI00 :: single-valued, cisfloat,
     {The maximum value of the SpecInt2000 benchmark of the nodes associated
       to the CE}
  AuthorizedUser :: multi-valued, cis,
     {The list of subjects of X509 user certificates for users authorized to submit
       jobs to the CE}
  RunTimeEnvironment :: multi-valued, cis,
     {List of softwares/packages installed on this CE}
  AFSAvailable :: single-valued, boolean,
     {Defines if AFS is installed}
  OutboundIP :: single-valued, boolean,
     {Defines if outbound connectivity is allowed}
  InboundIP :: single-valued, boolean,
```

```
        {Defines if inbound connectivity is allowed}
        }
  MAYCONTAIN {
    QueueName :: single-valued, cis,
        {The name of the queue in the LRMS}
    LRMSType :: single-valued, cis,
        {The type of LRMS}
    LRMSVersion :: single-valued, cis,
        {The version of LRMS}
  }
}

OBJECTCLASS CloseStorageElement {
  OID {??}
  DESCRIPTION {
              A Storage Element close to a Computing Element
  }
  KIND { STRUCTURAL }
  INHERITSFROM { DataGridTop }
  MUSTCONTAIN {
    CloseSE :: single-valued, cis,
        {The identifier of the SE}
    CEId :: single-valued, cis,
        {The identifier of the CE}
  }
  MAYCONTAIN {
    MountPoint:: single-valued, cis,
        {The mount point of this SE from this CE}
  }
}
}
```

4.1.3. WP5 schema in GOS form

The schema delivered by WP5 describe the following:--

The Storage Element - The static information defining the Storage element itself.

The Storage Element Protocol – This defines access method to the storage element

The Storage Element Status – The dynamic information defining the storage element – i.e. the current status.

The File Element – Information on an individual file.

```
NAMESPACE DataGrid {

OBJECTCLASS StorageElement {
  OID {??}
  DESCRIPTION {
              A Storage Element for the DataGrid Project
             }
  KIND { STRUCTURAL }
  INHERITSFROM { DataGridTop }
  KEY {SEId}
  MUSTCONTAIN {
    SEId :: single-valued, cis,
        {The identifier of the SE (the hostname for the PM9 release)}
    CloseCE :: multi-valued, cis,
        {The list of identifiers of CEs close to the SE}
    SEtypearchitecture::single-valued,cis,      {Architecture or type of storage element}
    SEsize::single-valued, integer, {Size of element MB}
    SEResourceContactString::single-valued, cis, {Resource Contact String}
  }
  MAYCONTAIN {
    SEmaxfilesize::single-valued, integer, {max size of file that may be stored - MB}
```

```
    SEmaxdata::single-valued, integer,    {max amount of data that may be stored by 1 job -
MB}
    SEmaxnumfile::single-valued, integer,   {max no of files which may be stored by 1 job}
  }
}

OBJECTCLASS StorageElementProtocol {
  OID {??}
  DESCRIPTION {
              A protocol spoken by a SE
             }
  KIND { STRUCTURAL }
  INHERITSFROM { DataGridTop }
  KEY {SEProtocol}
  MUSTCONTAIN {
    SEProtocol :: single-valued, cis,
        {The name of the protocol}
    SEId :: single-valued, cis,
        {The identifier of the SE (the hostname for the PM9 release)}
  }
  MAYCONTAIN {
    Port :: single-valued, integer,
        {The port number}
  }
 }


OBJECTCLASS StorageElementStatus {
        OID { ? }
        DESCRIPTION {
                Contains current information on the element itself. Updated frequently.
        }
        KIND {STRUCTURAL }
        INHERITSFROM {DataGridTop}
        MUSTCONTAIN {
                SEfreespace::single-valued, integer, {amount of free space available MB}
        }
        MAYCONTAIN {
                SEaccesstime::single-valued, integer, {approx time in seconds taken to access
storage element}
                SEfilequeuelen::single-valued, integer, {no of jobs in queue for the storage
element}
        }
}



OBJECTCLASS FileElement {
        OID { ? }
        DESCRIPTION {
                properties of a file.
        }
        KIND {STRUCTURAL}
        INHERITSFROM {DataGridTop}
        MUSTCONTAIN {
                localfilename::single-valued, ces, {local file name}
        }
        MAYCONTAIN {
                filecreatedate::single-valued, cis,          {date file was created}
                filemodifydate::single-valued, cis,          {date file was last modified}
                fileaccessdate::single-valued, cis,          {date file was last accessed}
                filesize::single-valued, integer, {size of file -bytes}
                filelatency::single-valued, integer, {time taken to access file - in seconds}
                filelifetime::single-valued, integer,  {time file will stay on disk}
                fileownname::single-valued, cis, {name or uid of file owner}
        }
}


}
```

4.1.4. WP6 schema in GOS form

WP6 schema is a site information summary.

```
NAMESPACE DataGrid {

OBJECTCLASS SiteInfo  {
        OID    { ? }
        DESCRIPTION { Site Information Summary }
        KIND { STRUCTURAL }
        INHERITSFROM { DataGridTop }
        MUSTCONTAIN {
                siteName :: single-valued, cis,
                { Name of site }
                sysAdminContact :: single-valued, cis,
                { Contact details for local system administrator }
                userSupportContact :: single-valued, cis,
                { Contact details for local user support }
                dataGridVersion :: single-valued, ces,
                 {Overall DataGrid software version }
                installationDate :: single-valued, time,
                 { Date of installation }
                cpuResourceDescription :: single-valued, cis,
                { Description of available CPU resources}
                diskResourceDescription :: single-valued, cis,
                { Description of available disk resources}
        }
        MAYCONTAIN {
                 supportedFileSystem :: multi-valued, ces,
                 { Supported file systems AFS, NFS, local }
                 batchSystem :: multi-valued, ces,
                 { Available batch system,  PBS, LSF, BQS, Condor, fork etc}
                massStorageDescription :: single-valued, cis,
                { Description of available mass storage}
                experimentalSoftware :: multi-valued, ces,
                { Availablity of software, ATLAS, CMS, ESA etc }
        }
}
}
```

4.1.5. WP7 schema in GOS form

WP7 schema defines the network information.

```
NAMESPACE DataGrid {

OBJECTCLASS NetworkMonitorRTT {
        OID { ? }
        DESCRIPTION {
                Contains attributes defining RTT. These are values against a source-ip
                (if no transfers are found per hn)
        }
        KIND { STRUCTURAL }
        INHERITSFROM { DataGridTop }
        MAYCONTAIN {
                rttmax:: single-valued, cisfloat, { Maximum rtt   }
                rttavg :: single-valued, cisfloat, { Average rtt     }
                rttmin :: single-valued, cisfloat, { Minimum rtt    }
                rttmdev :: single-valued, cisfloat, { Mdev specified by some versions of ping }
```

```
                rttlastmodified       :: single-valued, cisfloat, { Time rtt was last measured
}
        }
}

OBJECTCLASS NetworkMonitorRTTPacketSize {
        OID { ? }
        DESCRIPTION { Stores min/max/avg etc. RTT for each packet size }
        KIND { STRUCTURAL }
        INHERITSFROM { NetworkMonitorRTT }
        MUSTCONTAIN {
                rttpacketsize::single-valued, integer, { Packet size in bytes }
        }
}

OBJECTCLASS NetworkMonitorThroughput {
         OID { ? }
        DESCRIPTION {
                Contains attributes defining the Throughput (eg from iperf or netperf).
                These are values against a source-ip (if no transfers are found per hn)
        }
        KIND { STRUCTURAL }
        INHERITSFROM { DataGridTop }
        MAYCONTAIN {
                throughputtransmax:: single-valued, cisfloat, { Maximum transfer throughput  }
                throughputtransavg:: single-valued, cisfloat, { Average transfer throughput  }
                throughputtransmin:: single-valued, cisfloat, { Minimum transfer throughput  }
                throughputlastmodified::single-valued, cisfloat, { Time last throughput
measurement was made }
        }
}

OBJECTCLASS NetworkMonitorThroughputStreams {
        OID { ? }
        DESCRIPTION { Contains throughput information per number of streams }
        KIND { STRUCTURAL }
        INHERITSFROM { NetworkMonitorThroughput }
        MUSTCONTAIN {
                throughputnumstreams::single-valued, integer, { Number of streams used in
transfer }
        }
}

OBJECTCLASS NetworkMonitorThroughputBufferSize {
        OID { ? }
        DESCRIPTION {
                Contains throughput information per buffer size,
                classified by number of streams by directory structure
        }
        KIND { STRUCTURAL }
        INHERITSFROM { NetworkMonitorThroughput }
        MUSTCONTAIN {
                throughputbuffersize::single-valued, integer, { Buffer size in bytes used in
transfer }
        }
}

OBJECTCLASS NetworkMonitorLoss {
         OID { ? }
        DESCRIPTION {
                Contains attributes defining the Loss. These are values against a source-ip
        }
        KIND { STRUCTURAL }
         INHERITSFROM { DataGridTop }
        MAYCONTAIN {
                lossmin :: single-valued, cisfloat, { Minimum loss  }
                lossmax :: single-valued, cisfloat, { Maximum loss  }
                lossavg :: single-valued, cisfloat, { Average loss }
                losslastmodified :: single-valued, cis, { Last time loss measurement was made }
        }
}
```

```
OBJECTCLASS NetworkMonitorLossPacketSize {
        OID { ? }
        DESCRIPTION {
                Contains attributes defining the loss for a given packet size.
                These are values against a source ip
        }
        KIND { STRUCTURAL }
        INHERITSFROM { NetworkMonitorLoss }
        MUSTCONTAIN {
                losspacketsize::single-valued, integer, { Packet size in bytes?bits? }
        }
}

OBJECTCLASS NetworkMonitorHops {
        OID { ? }
        DESCRIPTION {
                Contains attributes defining the number of hops to a source ip
        }
        KIND { STRUCTURAL }
        INHERITSFROM { DataGridTop }
        MAYCONTAIN {
                hopslastmodified :: single-valued, cis,{Time that last traceroute was performed
}
                hopslast:: single-valued, integer, {Last measured number of hops}
                hopsmin:: single-valued, integer, {Maximum number of hops recorded}
                hopsmax:: single-valued, integer, {Minimum number of hops recorded}
        }
}

OBJECTCLASS NetworkMonitorHost {
        OID { ? }
        DESCRIPTION {
                The host to which measurements were made
        }
        KIND { STRUCTURAL }
        INHERITSFROM { DataGridTop }
        MUSTCONTAIN {
                rhn::single-valued,cis,{ Hostname of monitored ip/dns }
        }
}

OBJECTCLASS NetworkMonitorOU {
        OID { ? }
        DESCRIPTION {
                The organisational unit (read: network) to which measurements were made
        }
        KIND { STRUCTURAL }
        INHERITSFROM { DataGridTop }
        MUSTCONTAIN {
                rou::single-valued,cis,{ The remote ou measured to }
        }
}

OBJECTCLASS NetworkMonitorTool {
        OID { ? }
        DESCRIPTION {
                Standard attributes used to indicate the type of tool used
        }
        KIND { STRUCTURAL }
        INHERITSFROM { DataGridTop }
        MUSTCONTAIN {
                tool::single-valued,cis,{ Tool used to make measurements }
        }
        MAYCONTAIN {
                toolname::single-valued,cis,{ The name of the tool }
                toolversion::single-valued,cis,{ The version of the tool }
        }
}

}
```

4.2. R-GMA

The R-GMA is not a general distributed RDBMS system, but a way to use relational model in a distributed environment where ACID (Atomicity, Consistency, Isolation and Durability) properties are not considered essential. However, the R-GMA can be viewed as one huge logical database, partitioned according to certain criteria (specified by a WHERE clause as a predicate).

4.2.1. R-GMA ARCHITECTURE

The architecture consists of three objects, Producers, Consumers, and a Registry, as shown in Figure 4.2.1. Producers announce that they can publish specific information into the R-GMA by subscribing (announcing) to the Registry. This they do by issuing a SQL 'CREATE TABLE' specifying the structure of the table that they can publish. Consumers retrieve this information via SQL 'SELECT', which triggers discovery of the location of the information, i.e. a look-up of the Registry (currently this is done rather crudely – DataGrid are developing a Mediator to optimize the discovery). A canonical producer doesn't publish anything until queried, it just announces where it can be accessed and how the information is structured and then waits for a query. More elaborate producers may do more; they may even begin publishing once they are instantiated. Actual publishing is via a SQL 'INSERT', which may take place only in response to a query (and then as either a single item or a stream of items), or may be take place to a buffer or database in expectation of subsequent queries. Thus:

Producers	announce:	SQL 'CREATE TABLE'
Consumers	collect:	SQL 'SELECT'
Producers	publish:	SQL 'INSERT'

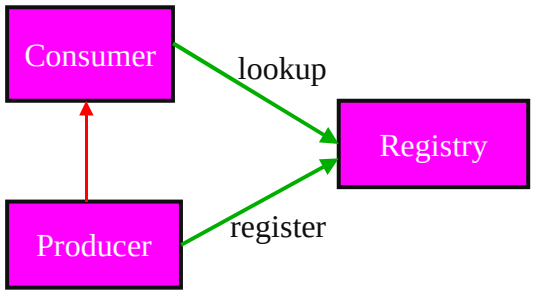

Figure 4.2.1 R-GMA Architecture

The R-GMA is written in Java, and is based on servlet technology, as shown in Figure 4.2.2.

A Producer is a class that communicates with a ProducerServlet object (nearby but not necessarily on the same host) via sockets. The class introduces insignificant overhead. The servlet, on the other hand, requires the Tomcat server, lots of memory, and significant overhead, but then it can reside on another host. When the class is instantiated the SQL 'CREATE TABLE' is sent to the ProducerServlet and

thence to the RegistryServlet object, so that it then knows what the information is and where it is. Then the class can publish the information via an 'insert' method that issues a SQL 'INSERT' to the ProducerServlet.

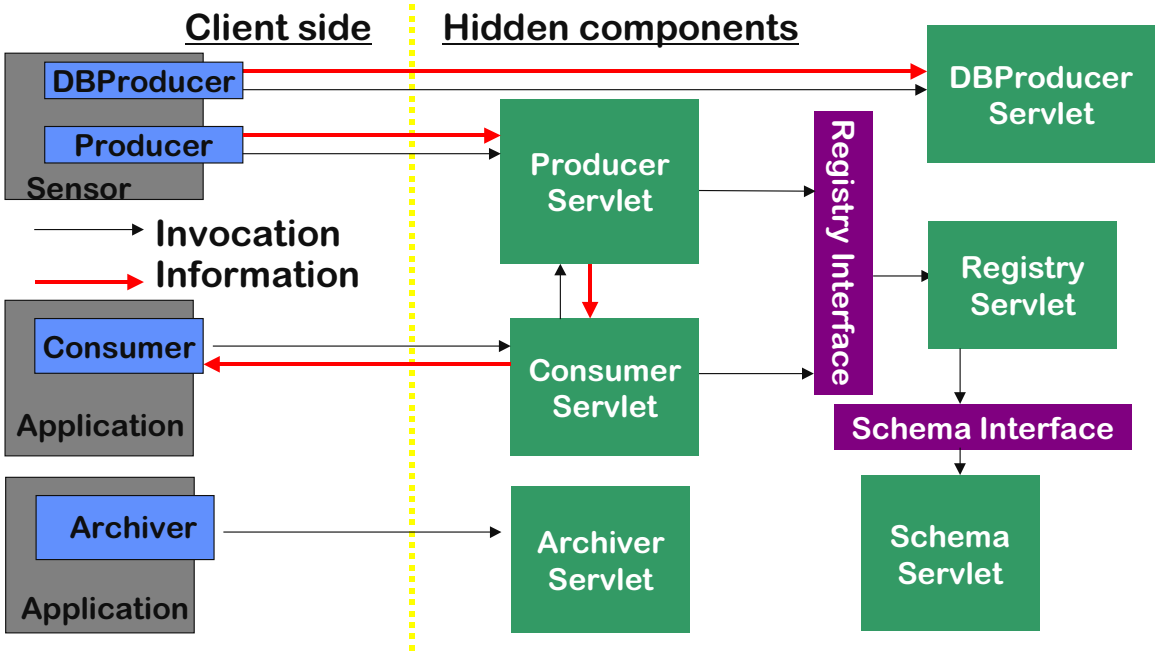

Figure 4.2.2 R-GMA Servlets

4.2.2. R-GMA PROTOCOLS

For R-GMA, SQL queries are issued using the HTTP/HTTPS protocol:

```
http://localhost:8080/ProducerServlet/constructProducer?tableName=cpuLoad&flags=1
```

Result-sets are returned via the XML protocol:

```
<GMA-Response … >
  <ResultSet>
    <rowMetaData>
      <colMetaData>loadavg</colMetaData>
      <colMetaData>timestamp</colMetaData>
    </rowMetaData>
    <row>
      <col>0.15<col>
      <col> … <col>
    <row>
    <row>
      <col>0.15<col>
      <col> … <col>
```

```
        <row>
      </ResultSet>
    </GMA-Response>
```

At the Producer servlet a ResponseWriter converts a result-set object to XML; at the Consumer servlet a XMLConverter restores them to Java objects.

4.2.3. R-GMA IMPLEMENTATION

In the implementation, each application that uses a servlet does so via an API. There are four kinds of servlets that interact directly with the client side: a Producer (a basic buffering or streaming producer), a DBProducer (that renders information persistent by storing it using a RDBMS such as MySQL), a Consumer and an Archiver. The Archiver is a special kind of servlet that can have multiple Consumer interfaces inserting selected information into a DBProducer, which can then itself be queried – this is expected to be intensively used within Task 3.3 for post-processing of monitored information. A fifth servlet, called a CanonicalProducer, is in development, and is also expected to be intensively used by Task 3.3 for access to independently-generated logfiles. Thus the implementation is as shown in Figure 4.2.3.

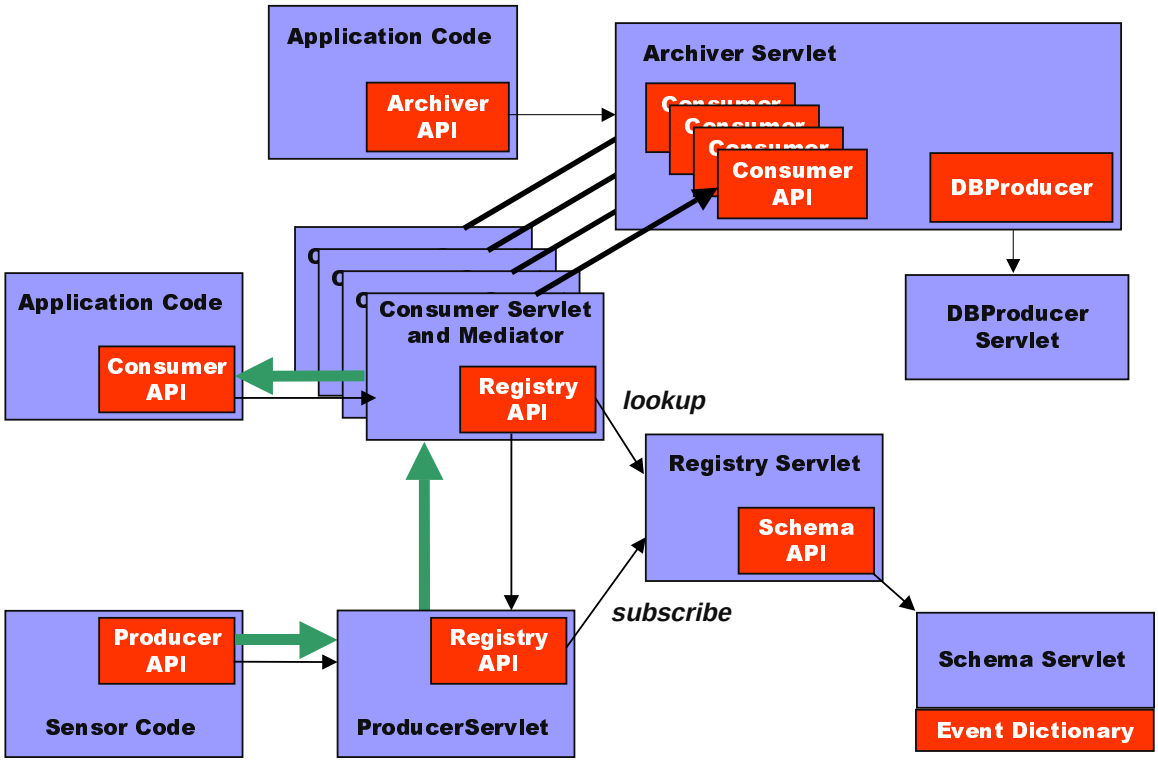

Figure 4.2.3 R-GMA Implementation

5. INDEX