EGEE

R-GMA System Specification

Document identifier:	**EGEE-JRA1-TEC-490223**
Date:	**September 4, 2006**
Activity:	**JRA1: Middleware Engineering and Integration (UK Cluster)**
Document status:	**DRAFT**
Document link:	**https://edms.cern.ch/document/490223/**

Abstract: This document presents the System Specification for the Information and Monitoring Services middleware component (R-GMA), in sufficient detail to support design verification, detailed design and test specification. It describes, in detail, *what* R-GMA will do.

The document is structured with a single chapter containing a technical overview for the proposed system, followed by a separate chapter for each of the R-GMA services, with a standard set of headings in each of these chapters. A chapter on security follows these, and the final chapter specifies the subset of SQL used in R-GMA.

Document Change Log

Issue	Date	Comment	Author
0.1	31/05/04	First Draft	JRA1-UK
2.0	28/07/04	Second Draft	JRA1-UK

Document Change Record

Issue	Item	Reason for Change

CONTENTS

1 INTRODUCTION

1.1 PURPOSE AND STRUCTURE OF THIS DOCUMENT

This document presents the System Specification for the Information and Monitoring Services middleware component (R-GMA), in sufficient detail to support the following activities:

- Design verification: There are few extant requirements documents for this middleware component, so as the first deliverable of system design activities, this document sets out explicitly *what R-GMA will do*, so that the project design authorities can verify that it will provide the required services to other middleware components and users of the EGEE grid infrastructure.
- System design: JRA1-UK will produce detailed designs to show exactly *how* the system specified here will be implemented.
- System test: This document is the primary input to test specification activities.

This introductory chapter contains the standard headings required for EGEE technical documents. Chapter 2 contains a technical overview for the proposed system. This is followed by a separate chapter for each of the R-GMA services, with a standard set of headings in each chapter. A chapter on security follows these, and the final chapter specifies the SQL language subset used by R-GMA.

1.2 APPLICATION AREA

This document affects the following three areas:

- JRA1-UK (system design)
- JRA1 Middlware Design Team and Project Technical Forum (design assurance)
- JRA1 Test Team (system/integration testing)

1.3 REFERENCES

This is the definitive specification document for R-GMA. The main inputs to the specification process were the DataGrid R-GMA documentation (especially the architecture document), the work already done on the new R-GMA design and an analysis of the R-GMA code as it was at the end of the DataGrid project. This document is self-contained, but the following documents, referred to in the text, can provide additional background information.

1.4 DOCUMENT EVOLUTION PROCEDURE

This document is collectively owned by JRA1-UK. This document will be updated periodically by JRA1-UK in the light of feedback received. It is held in CVS and new versions will be published on the JRA1-UK Web page and in EDMS.

1.5 TERMINOLOGY

General project definitions can be found in the Project Glossary. R-GMA and Web Services terminology is explained in the architecture overview (section 2).

2 BACKGROUND

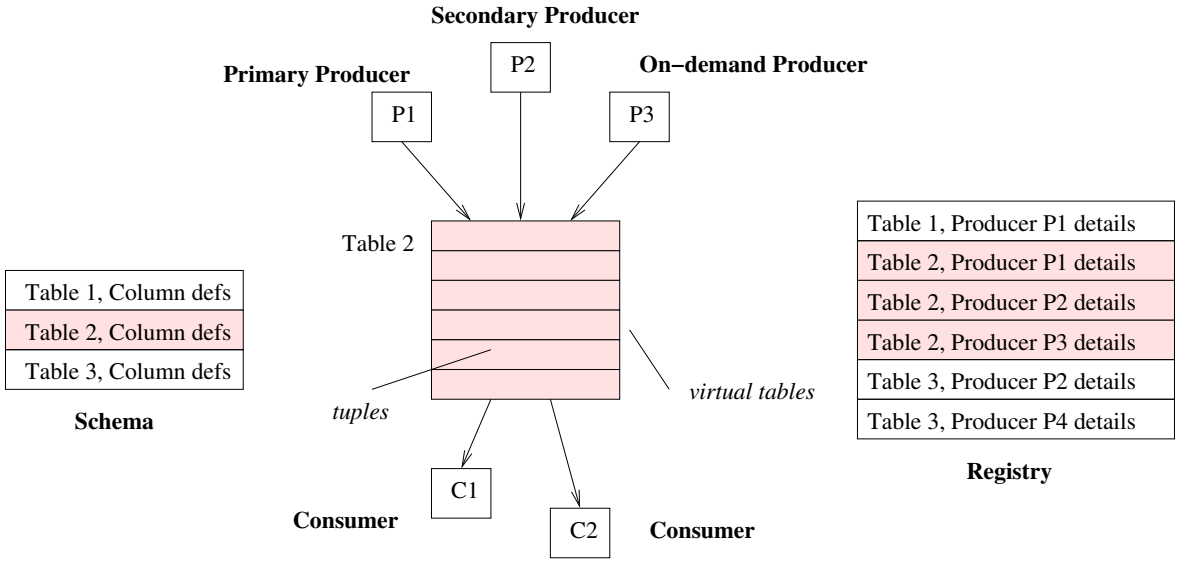
2.1 SCOPE OF THIS SPECIFICATION

The EGEE Grid Information and Monitoring Services component being specified here is a production version of the R-GMA Information and Monitoring System developed by Work Package 3 of the European DataGrid project - the name R-GMA is being retained. This document specifies the core R-GMA services and describes the standard tools which will be delivered with the system. It does not explain how other gLite (EGEE Middleware) components will interact with R-GMA.

2.2 R-GMA ARCHITECTURE

2.2.1 VIRTUAL DATABASE

R-GMA is an implementation of the Grid Monitoring Architecture (GMA) proposed by the Global Grid Forum (GGF), which models the information infrastructure of a Grid as a set of *Consumers* (who request information), *Producers* (who provide information) and a single *Registry* (which mediates the communication between producers and consumers). R-GMA imposes a standard query language (a subset of SQL) on this model - so producers publish *tuples* (database rows) with an SQL insert statement and consumers query them using SQL select statements. R-GMA also ensures that all tuples carry a *time-stamp*, so that monitoring systems (which require time-sequenced data) are inherently supported. A full description of the R-GMA architecture is contained in the EGEE Architecture document [2]: what follows, is a brief description of the components (services) provided by R-GMA, sufficient for reading this document.



R-GMA presents the information resources of a Virtual Organisation (VO) as a single *virtual database* containing a set of *virtual tables*. As the picture above shows, a single[1] schema contains the name and structure (column names, types and settings) of each virtual table in the system. A single registry contains a list, for each table, of producers who have offered to *publish* (provide data for) rows for the table. A consumer runs an SQL query against a table, and the registry selects the best producers to answer the query in a process called *mediation*. The consumer then contacts each producer directly, combines the information, and returns a set of tuples. The mediation process is hidden from the user. Note that there

[1] Although there is only one logical schema and registry pair per VO, identical replicas are made for scalability and robustness. This is discussed in detail later.

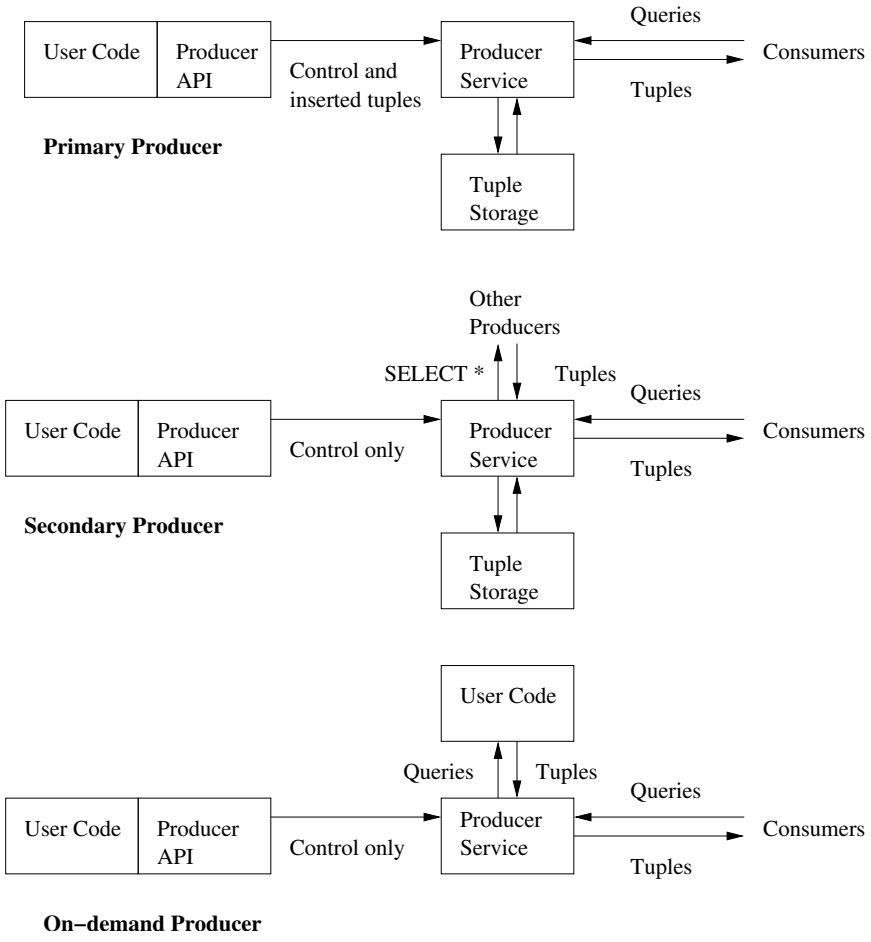
is no central repository holding the contents of the virtual table; it is in this sense, that the database is virtual.

2.2.2 PRIMARY KEYS AND TIME-STAMPS

Each virtual table has a *key* column (or group of columns) declared in the schema. Each tuple published by a primary producer also carries a *time-stamp*, added by the producer, which, together with the key columns, is similar to a primary key for the table. Tuples with the same key, but different values for the time-stamp, can also be thought of as different versions of the same tuple.

2.2.3 PRODUCERS

There are three classes of producer: *Primary*, *Secondary* and *On-demand*. Each is created by a user application and returns tuples in response to queries from other user applications. As the picture below shows, the main difference is in where the tuples come from.



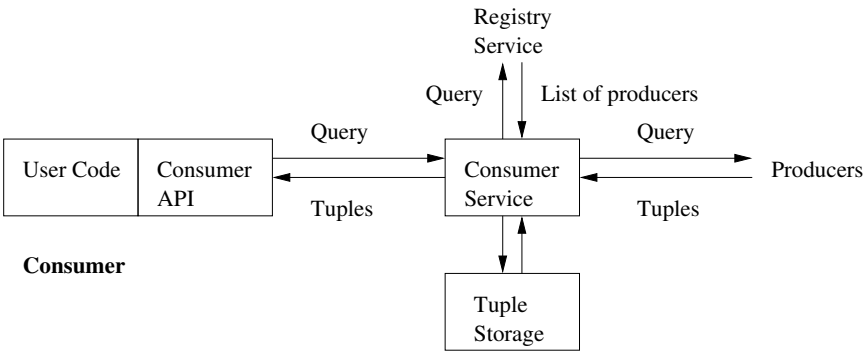
The *producer service* in these pictures is a process running on a server on behalf of the user code (this is expanded in the Web Services section which follows). In a Primary Producer, the user code periodically inserts tuples into storage maintained internally by the Primary Producer service. The producer service autonomously answers consumer queries from this storage. The Secondary producer service also answers queries from its internal storage, but it populates this storage itself by running its own query against the virtual table: the user code only sets the process running; the tuples come from other producers. In the

On-demand Producer, there is no internal storage; data is provided by the user code in direct response to a query forwarded on to it by the producer service.

The tuple-storage maintained by Primary and Secondary producers can either be in memory, in a file, or in a real database table. Producers that use non-database storage are optimized to answer simple queries quickly, but they must support complex queries too (e.g. by creating an in-memory database on the fly). In an On-demand producer, tuple-storage (if any) is the responsibility of the user code, but may also be in a real database.

2.2.4 CONSUMERS

In R-GMA, each consumer represents a single SQL SELECT query on the virtual database. The request is initiated by user code, but the *consumer service* carries out all of the work on its behalf. The query is first passed to the Registry to identify which producers, for each virtual table in the query, must be contacted to answer it. This process is called *mediation*, and the algorithm used in R-GMA is explained in section 6.2. The query is then passed by the consumer service to each relevant producer, to obtain the answer tuples directly.



There are four types of query: *continuous*, *latest*, *history* and *static*. The set of queries that a particular producer supports is recorded in the registry. All query types except static can take an optional time interval parameter.

A continuous query causes all new tuples that match the query, to be streamed into the consumer's tuple-storage, as soon as they are inserted into the virtual table by the producers. Streaming continues until the consumer requests it to stop. If a time interval is specified, the consumer will additionally receive any tuples which are already in the virtual table when the query starts, and which are no older than the time interval. There is no guarantee that tuples are time-ordered. All Primary and Secondary producers support continuous queries. On-demand producers do not.

Latest and history queries are *one-time* queries: they execute on the current contents of the virtual table, then terminate. In a history-query, all versions of any matching tuples are returned; in a latest-query, only those representing the "current state" (see below) are returned. In both cases, a time interval may be specified with the query, to limit the age of the tuples returned. Primary and Secondary Producers may optionally support one-time queries. On-demand producer do not.

Static queries are only supported by On-demand producers. They are one-off database-like queries and do not contain R-GMA time-stamps. The primary purpose of an On-demand producer is to allow large databases to be accessed through the R-GMA infrastructure, without the overhead of copying tuples into a producer service.

2.2.5 RETENTION PERIODS

To allow Primary and Secondary producers to periodically purge "old" tuples, and to give a precise meaning to the "current state" for a latest query, *retention periods* are used. On-demand producers do not use retention periods.

A *LatestRetentionPeriod* (time interval) is inserted into each tuple published by a Primary Producer, and remains there when a tuple is re-published by a Secondary Producer. In addition, Primary and Secondary producers declare (in the registry) a *HistoryRetentionPeriod* for each table to which they are publishing tuples.

Primary and Secondary producers therefore have two logical tuple-stores, one supporting latest-queries and the other supporting continuous and history queries. Producers undertake to retain the *most recent* version of any tuple which has not exceeded its LatestRetentionPeriod, and *all* versions of any tuple which have not exceeded the HistoryRetentionPeriod.

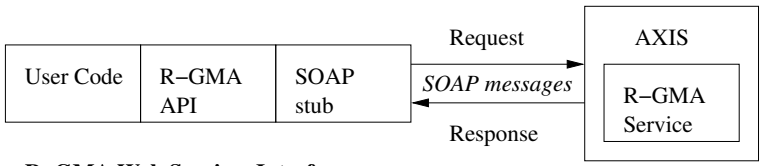
A latest-query returns only the most recent versions of tuples, and only those tuples which have not exceeded their LatestRetentionPeriod (this is the definition of current state). Conversely, a history-query returns whatever is available, but that is guaranteed to include at least all versions of tuples which have not exceeded the producer's HistoryRetentionPeriod for the table.

The exact details of how retention periods work for each producer type are specified in the corresponding chapters in this document.

2.3 WEB SERVICES ARCHITECTURE

2.3.1 OVERVIEW

R-GMA conforms to the Web Services Architecture [7]. It consists of six principal *services* (Primary Producer, Secondary Producer, On-demand Producer, Consumer, Registry and Schema), running on one or more servers. This document contains a chapter for each of these services. Each service has a well defined set of *operations* (such as a producer's *startStreaming* operation) which it can carry out, as specified in a machine-readable XML document conforming to the Web Services Description Language (WSDL [8]). There is one WSDL document for each service. All operations are requested by applications through an exchange of messages with the service. The sequence and format of messages required for each operation, including any parameters, is also specified in the WSDL.



R–GMA Web Services Interface

R-GMA uses "SOAP messaging over http/s" [6], in a request/response pattern, for all user-to-service and service-to-service communications apart, from streaming. As the picture shows, if the user chooses to use the API (described below), the SOAP interface is hidden from them. Apache AXIS software implements the SOAP interface on the server. Streaming tuples to continuous consumers uses a lower lever communication, for efficiency.

2.3.2 RESOURCE FRAMEWORK

Each instance of a producer or consumer in a running R-GMA system exists as a *resource* on a server. A resource contains the private data or threads associated with that particular instance (such as the tuple-storage and tuple-streamer in a Primary Producer - see section 3.1), and it is created by an R-GMA

service, when a user sends a "create" request. It resides on the server with the service, and is given an identifier which is passed back to the client. The client then includes the resource identifier with all subsequent requests relating to that instance (the API takes care of this). A resource is normally destroyed at the explicit request of the user, but in order to protect itself from an accumulation of redundant resources, an R-GMA service requires the user to specify a termination interval (in a restricted range) when it creates the resource. If the service doesn't hear from the user for any period exceeding the termination interval, the resource is destroyed. This is the concept of *soft-state registration*. It puts the onus on the user to keep the resource alive, by making periodic contact with the service, but any contact will do. The registry protects itself in the same way, against producers and consumers which register then disappear, so a periodic keep-registered message has also to be sent to the registry, by the consumer and producer services.

R-GMA has a Resource Framework (borrowing from the WSRF [4] work) to manage the life-cycle of resources running on a server. This includes sending keep-registered messages to the registry on behalf of producer and consumer resources. It is implemented as part of the R-GMA services, and is hidden from the user.

Registry and Schema instances also exist as resources created by a Registry or Schema service. This allows a single service to host registries or schemas for multiple virtual organisations. However they are not managed by the Resource Framework, the VO is the resource identifier, and they do not time out.

2.3.3 FACTORY SERVICES

For each of the six principal services, R-GMA has a corresponding *factory service* to create resources. This extra level of indirection allows R-GMA to support services with different attributes (*quality of service*): the user specifies the attributes required (e.g. reliable delivery of tuples) and the factory service creates a new resource with the URL of a suitable service. The user only needs to know the URL of their local factory service.

2.4 API

2.4.1 USER API

R-GMA provides APIs for Java, C++, C and Python languages, to make it easier for user applications to interact with the R-GMA services. The APIs are independent of each other and are designed to present an easy-to-use and apropriate interface to R-GMA for each supported language. Each operation of each service is represented by a method (or function) in the API, and that method simply packages up its parameters into a SOAP message and sends it to the Web Service for execution. Any return values or errors (exceptions) are passed back to the caller. The API transparently manages any authentication required by the server, and looks after the resource identifier. Direct interaction between a user application and the R-GMA Web Service (by-passing the API) is also supported.

All R-GMA services report errors by sending SOAP Faults. Each API presents these to the user in a manner appropriate to the language (e.g. as Java Exceptions). The absence of a fault indicates success. The results of SQL queries (tuples) are returned in Result Sets which are defined in section 10.4. Result Sets will contain a warning if the mediator is unable to guarantee that the result is complete.

2.4.2 SYSTEM/ADMINISTRATION APIS

System APIs exist for communication between R-GMA services, hiding the SOAP interface and managing authentication. An Administration API also exists to support browsing and administration of the

registry and schema. The System APIs are implemented only in Java and are not visible to the user. The Administration API is implemented in all supported languages.

2.5 Security Considerations

Security for R-GMA is discussed in detail, in chapter 9.

2.6 Performance Considerations

To be completed (the general requirement is, of course, for services which are robust, fast enough and highly scalable)

2.7 Fault Tolerance, Exception Handling and Recovery

2.7.1 Replication

The schema and registry databases represent the only single point of failure in an R-GMA system, and so multiple (running) replicas are maintained on different servers and are periodically synchronized (see 7.1.3 and 8.1.3). A producer or consumer must normally continue to use the replica with which it originally registered, but it is capable of transferring itself to another replica in its VO, if the original one fails. The other R-GMA services are self-contained and can be installed on as many servers in the VO as required.

2.7.2 Recovery following restart

R-GMA instances (Web Service Resources) are always destroyed when the corresponding service restarts. All calls to services in R-GMA (either from the user or another service) must therefore be prepared to discover that a resource no longer exists, or is no longer registered in the registry, and to handle the error gracefully. The use of time-outs (soft-state registration) on the registry and in the Resource Framework, ensures entries to non-existent resources are automatically removed within a reasonable length of time.

Those R-GMA services which use permanent storage (registry service, schema service and primary and secondary producers with user-specified tuple stores) do have some degree of resilience, because new resources can connect to existing storage, provided the storage itself is recoverable. In the case of the registry and schema, the database is brought up to date by sending a sychronization request to other running replicas. In the case of a producer's file or database tuple-store, existing tuples will be automatically available, and will remain in the store until they exceed their retention period, in the usual way.

2.7.3 Error reporting

Errors are reported by R-GMA services in a consistent way as *SOAP Faults* across the Web Services interface. There are three types of fault:

RemoteException Network or security error connecting to service (user can retry later)

UnknownResourceException Resource identifier is unknown (user needs to re-create resource)

RGMAException Other R-GMA error (user needs to investigate before retrying)

Each supported language API presents SOAP Faults to the user in a manner appropriate to the language. Each fault contains an error code, to specify the exact error which has occured, and there is a standard error message for each code.

Warnings are only issued in R-GMA in response to consumer queries, and only to indicate possible inadequacies in the results. They are part of the query's Result Set.

The table below shows how errors that are common to all services are handled by R-GMA. Each service also has a table of service-specific faults in its own chapter.

Fault	Response
API can't connect to service.	Service does nothing. API reports a RemoteException.
User not authenticated.	Service does nothing. API reports a RemoteException.
User authenticated but not permitted to connect.	User is not connected. Service reports a RemoteException.
User connected but not authorised to carry out requested operation.	Operation is not carried out. Service reports a RemoteException.
User connected but resource no longer exists.	Operation is not carried out. Service reports an UnknownResourceException.
User omits mandatory parameters to operation or provides invalid values for them.	Operation is not carried out. Service reports an RGMAException.
Operation fails for any reason not described elsewhere, but recovery is possible.	Operation is reversed or resource is made stable. Service reports an RGMAException.
Operation fails for any reason not described elsewhere, and recovery is not possible.	Service destroys resource (but not any permanent tuple-store) and reports an RGMAException.

In all cases, any external resources (especially database and network connections) are freed up wherever possible, and any threads created by the resource should be terminated as cleanly as possible.

2.8 HARDWARE/SOFTWARE CONSIDERATIONS

In line with EGEE SA1 requirements, R-GMA is designed to run on the following operating systems:

- Linux Red Hat Enterprise 3 (or binary compatible version)
- Microsoft Windows XP

There are no special hardware requirements.

2.9 STANDARD TOOLS

Two simple user-interfaces to R-GMA will be delivered as part of the standard installation, providing command-line and Web-browser access to an R-GMA installation.

2.9.1 R-GMA COMMAND LINE

The R-GMA command line tool provides a command-line administration facility for all tables in the virtual database. You can also write to and read from virtual tables by creating producers and consumers, and by issuing insert and select statements.

The R-GMA User Guide will include full documentation for this utility.

2.9.2 SCHEMA BROWSER

The R-GMA Schema Browser is a Java Servlet application which provides a Web browser-based query and administration facility for all tables in the virtual database (schema) of a particular VO.

The R-GMA User Guide will also include full documentation for this utility.

2.10 PACKAGING AND INSTALLATION

R-GMA will be packaged, installed and configured in accordance with EGEE SA1 requirements. It will be delivered in the following packages.

- Base (one package: all common configuration and documentation files)
- Services (one package: all six services)
- WSDL (six packages: each service's WSDL in a separate one)
- Java API: (two packages: interface and implementation)
- C++ API: (two packages: interface and implementation)
- C API: (two packages: interface and implementation)
- Python API (two packages: interface and implementation)
- Command Line and Browser (two packages: one for each tool)

All packages depend on the Base package, but the others can be installed independently (the Services package will contain a copy of any part of the Java API which it requires). Any other tools built on R-GMA (e.g. Service Status, Accounting, etc.) will be delivered in separate packages.

The WSDL is split off from the services because alternative implementations of the services may be provided by others (e.g. for logging and bookkeeping). The reason for splitting the APIs into two packages is to allow the implementation (e.g. SOAP stubs) to change without affecting the API interface.

An installation/configuration guide will be delivered with the system.

2.11 EXTERNAL DEPENDENCIES

2.11.1 INTERNET PORT NUMBERS

An R-GMA server needs three configurable ports with the following default assignments:

8080 - for http connections to services

8088 - for streaming to consumers

8443 - for https connections

Users creating On-demand producers will need to provide a port on the client system to handle queries and return data.

2.11.2 TIME SYNCHRONIZATION

Time intervals are used throughout the R-GMA system, and for consistency, they will be represented as integer milliseconds everywhere. Absolute time is held in only two places, in resource termination times and in tuple time-stamps.

The use of absolute termination times for resources in the Resource Framework is entirely internal to that system and therefore presents no time synchronization issues. The user specifies a termination *interval* when a resource is created and simply sends sign-of-life signals to keep it alive.

The UTC time-stamps on each tuple in R-GMA *do* have visibility beyond the system which creates them, so synchronization problems can occur when time-stamps are compared. This happens at three points in R-GMA:

- *where tuples are inserted into a latest-type producer*
- *when deleting tuples which have exceeded their minimum retention period*
- *where a user runs a query which involves the time-stamp fields*

A perfect solution is not needed. R-GMA time-stamps only have a resolution of one second[2], and system/network latencies will further reduce the resolution of any monitoring system using them. It is suggested therefore, that all computer systems within a VO which connect to an R-GMA installation, should synchronize their system clocks using something like NTP. R-GMA will contain a warning to this effect in the User Guide and in the installation software, but will not enforce it.

2.11.3 WEB SERVICES SOFTWARE

The R-GMA Web Service is currently implemented using a SOAP/https binding and uses Apache Axis software to handle that interface. Axis is a Java Web Application running in the Apache Tomcat *servlet container*, and is accessed by client programs via a URL, over http/https. R-GMA services (port types) are implemented as Java classes registered with Axis, and Axis simply hands off incoming messages to the appropriate R-GMA service, running in the same Tomcat installation. The R-GMA Schema Browser tool runs as a stand-alone Java servlet, also in Tomcat. A Java 2 SDK is a pre-requisite for installing Apache Tomcat.

2.11.4 RELATIONAL DATABASE MANAGEMENT SYSTEMS

R-GMA services use JDBC to interface to the Registry database, Schema database and the tuple-storage databases used by some producers.

It is intended to support any database for which a JDBC driver is available, but R-GMA is primarily tested with MySQL (disk) and HSQLDB (in-memory).

See section 10.1 for information about the subset of SQL used by R-GMA.

2.11.5 SECURITY MANAGEMENT SOFTWARE

R-GMA will use the standard EGEE security middleware (as yet undefined) to access user/service credentials for authentication and authorization purposes. Interfaces for each language used by R-GMA (services and API) will be required.

[2] This may change

2.11.6 LOGGING SOFTWARE

R-GMA uses the open-source Log4j software for configurable debug logging in R-GMA services and the Java API. It is intended to make use of the Apache Logging Services [1] for consistent logging across all APIs.

2.12 SERVICE SPECIFICATIONS

The chapters which follow contain detailed specifications for each of the six R-GMA services. Each chapter has a standard set of headings: The *description* section describes what the service does and what its principle components are.

The *interface* section lists all of the operations provided by the service, because these comprise the entire interface to the service. We provide a precise description of the functionality of each operation, but details of parameters and return values are relegated to the WSDL document and the R-GMA Interface Specification document, which will be delivered with the system.

The *error-handling* section describes how major service-specific failures (e.g. buffers filling up) are dealt with, because in some instances, the correct response is not simply to report an error and give up. Minor/common errors are covered by the general headings in section 2.7.3.

The *external objects* section lists all of the externally configured things (such as databases and configuration parameters) on which this service depends.

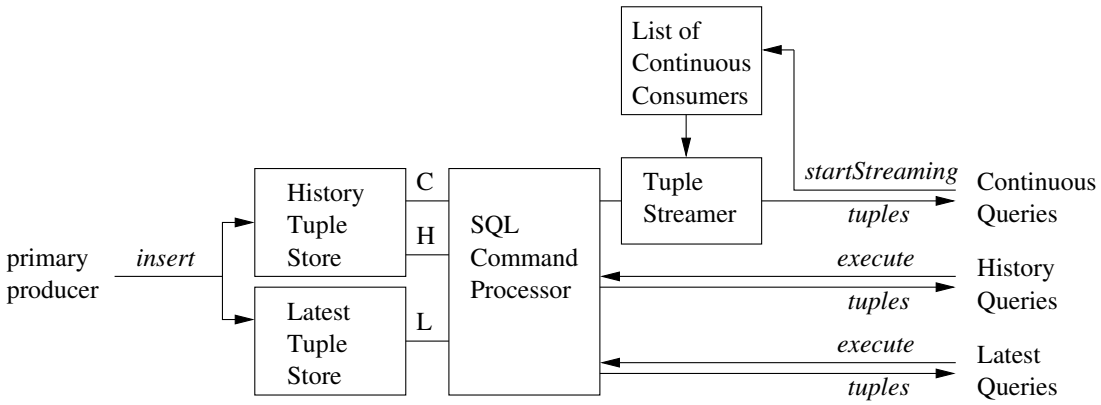
Finally, the schema and registry services have an extra section on *replication*, and the registry service has an extra one on *mediation*.

3 PRIMARY PRODUCER

3.1 DESCRIPTION

3.1.1 SERVICE COMPONENTS

The Primary Producer Service allows a user to publish tuples to one or more virtual tables. The picture below shows its principal components.



3.1.2 RESOURCE LIFECYCLE

A new Primary Producer Resource is created when a user calls the Primary Producer Factory Service's *createPrimaryProducer* operation. The resource holds the private data and threads for a single producer

instance. The user must declare the type of queries they wish the producer to support, and the type of tuple-storage they wish it to use. It is the job of the Producer Factory Service to identify a compatible Producer Service to create and store the resource.

Each Primary Producer resource has a Termination Interval that is a time interval within which the user must make contact with the producer service, in order to keep the resource alive and maintain its table entries in the registry. The Termination Interval is set by the user when the resource is created (the service imposes a maximum value) and can be subsequently changed by calling *setTerminationInterval*. If the producer's publication interval (the interval between calls to *insert*) is greater than the Termination Interval, then the user should call *showSignOfLife* perodically to keep it alive.

The resource is destroyed after the user sends a *close* or *destroy* request, or the resource exceeds its termination interval without any contact from the user.

3.1.3 REGISTRATION

Users must declare (*declareTable*) their intention to publish to a virtual table before they can do so. The table must already exist in the schema. The Primary Producer Service obtains the table structure from the schema and creates a corresponding table (with columns of the same types) in the resource's tuple storage. It then registers the producer in the registry as a publisher for that table. Once the producer is registered, the Producer Service will autonomously service SQL queries from consumers, from its internal tuple storage.

The user can declare more than one table in a single producer. They are treated independently by the Producer Service, except for processing "join" queries which involve more than one of the tables for which the producer is registered.

A producer always publishes complete tuples (i.e. all columns), but it might choose to publish only a subset of the tuples in the virtual table. The user indicates this by including a *predicate* in the form of an SQL WHERE clause, when the table is declared. A producer is only permitted to publish tuples that match its predicate; inserts which don't match will fail. The mediator requires that a producer's predicate consists only of a union of column equality constraints ("WHERE col1=val1 AND col2=val2…").

A producer may publish to more than one VO. The user supplies a list of VOs when the producer resource is created. The producer service registers tables in each VO's registry (and keeps them registered in all of the registries).

3.1.4 SUPPORTED QUERIES

All Primary Producers support continuous queries (so they can always be used by Secondary Producers). They may optionally support latest and/or history queries as well. Primary Producers do not support static queries. The type of queries a producer supports is the same for all tables for which it is registered, and is recorded against each table entry in the registry, so the mediator can check it.

3.1.5 PUBLISHING, STORING AND DELETING TUPLES

Primary Producers have up to two logical tuple-stores. Continuous queries are answered from a *history tuple-store*, as are history queries, if they are supported. If latest queries are supported, they are answered from a *latest tuple-store*. Tuple-stores may be physically in memory, file or database storage. Memory-based producers are optimized for fast streaming to continuous consumers, while database producers are optimized for answering complex one-time queries. A single producer uses only one type of storage for all types of queries that it supports. Users may specify their own databases (as a JDBC connection string) for use as tuple stores.

Tuples are inserted into the producer service's tuple storage by the original user through the *insert* operation, and are checked for type against the schema. The following tuple metadata is added to each tuple as extra columns, by the Primary Producer Service (still to be agreed):

RgmaDate	Date part of time-stamp, in the form "YYYY-MM-DD": only added if not already filled in by user.
RgmaTime	Time part of time-stamp, in the form "HH:MM:SS": only added if not already filled in by user.
RgmaLRP	Latest Retention Period (see below), in integer milliseconds.
RgmaHost	DNS name of host publishing the information.
RgmaUser	DN (distinguished name) of user publishing data (NULL if not authenticated).

The Producer Service puts the tuple into the history tuple store and, if present, the latest tuple-store, where it will replace any existing one with the same key, provided its time-stamp is not older.

The user must set a History Retention Period and a Latest Retention Period for each declared table. The History Retention Period is recorded in the registry. The Latest Retention Period is added by the service into each inserted tuple. All Primary Producers must insert the Latest Retention Period, even if they don't support latest queries: this ensures that any Secondary Producers to which they are streaming can choose to support latest queries. The two retention periods are independent.

A Primary Producer retains any published tuple in its history tuple-store until it has exceeded the History Retention Period for the table. It also retains the latest version of any published tuple in its latest tuple-store, if it supports latest queries, until the tuple has exceeded its Latest Retention Period. The producer service has site-configured maximum values for both retention periods, to protect the service itself.

Latest tuples will never be returned in any query once they have expired, so they may be deleted as soon the Latest Retention Period has been exceeded. Tuples in the history store, however, will usually be retained even when they have expired, if there is at least one continuous consumer still waiting for them. In this instance, the producer will block all future inserts when its tuple storage fills up, until a tuple has finally been delivered and deleted. A producer service may choose not to implement this and delete expired tuples regardless of any waiting consumers: the user selects a service with the appropriate behaviour via the *ignoreSlowConsumers* flag when the producer resource is created. When a Primary Producer resource is closed (via the *close* operation), the resource persists until the tuples have expired according to the rules just described. By contrast, the *delete* operation will cause a Primary Producer resource to terminate immediately.

3.1.6 MESSAGES FROM CONSUMER SERVICE

Consumer Services send a *startStreaming* message to the Primary Producer Service when they want a continuous consumer to start receiving newly inserted tuples from it. Tuples are streamed to consumers using an autonomous tuple-streamer (owned by the Primary Producer resource), as soon as they are inserted into the tuple-store. They are filtered on-the-fly according to the column expression and predicate of each consumer's query, before being pushed by the tuple-streamer into the consumer's own tuple-store. Only simple queries are supported (see section 10.1.1). There is also a *stopStreaming* message to turn streaming off for a particular consumer.

Consumer Services send an *execute* message to run a One-Time query against the current contents of a Primary Producer's tuple-store. Results are buffered for retrieval by the Consumer Service. Complex queries are supported, including those involving joins between the tables for which this producer is registered. Section 10.1.2 specifies the minimum subset of SQL which any Primary Producer, regardless of its tuple-storage type, must support. Beyond that, the complexity of one-time queries a Primary Producer will support is not restricted by R-GMA.

3.2 INTERFACE

The Primary Producer service provides the following operations:

createPrimaryProducer	(FACTORY SERVICE INTERFACE) Creates a new Primary Producer resource and registers it with the Resource Framework, with the specified termination interval; • query type (latest and/or history), storage type (memory, file or database) and handling for slow consumers are fixed at this point; • all Primary Producers support continuous queries; • the user must provide a list of one or more VO's to which this producer will publish; the producer will be registered in each VO's registry;
declareTable	(USER INTERFACE) Registers this producer in the registry as a publisher for a specified table, which must already exist in the schema; • must declare a predicate to restrict the tuples it will publish (this is an SQL WHERE clause which is limited (by the Registry mediator) to a union (AND) of equality constraints on columns in the table); may be empty • Latest Retention Period and History Retention Period fixed at this point; • Corresponding table is created in resource's tuple-store; if this already exists, its structure is checked and any existing tuples will be automatically available for consumer queries until they expire;
insert	(USER INTERFACE) Inserts one complete tuple of data into the producer's tuple storage; • tuples are provided in the form of SQL INSERT statements and must match the producer's predicate; • a time-stamp is added to the tuples if they do not already have one, or if set to NULL; the Latest Retention Period is also added; • new tuples are automatically streamed to any registered (continuous) consumers, evaluating the consumer's SQL SELECT statement on the fly;
insertList	(USER INTERFACE) Inserts multiple tuples of data in one operation (see *insert*).
startStreaming	(SYSTEM INTERFACE) Requests a producer to start streaming new tuples to the specified consumer, for a given SQL SELECT query and a given table; • consumer may also request existing contents of producer's tuple store, newer than a certain date/time, to be streamed first;

stopStreaming	(SYSTEM INTERFACE) Requests a producer to stop streaming new tuples to the specified consumer;
execute	(SYSTEM INTERFACE) Requests a producer to execute a consumer's SQL SELECT query on the contents of the producer's tuple store and return the answer immediately; • request will be either for latest or all (history) tuples;
showSignOfLife	(USER INTERFACE) Updates the contact time for the resource: if it exceeds its termination interval without any contact, the Resource Framework will unregister and close it; • the contact time is also updated following any operation except *ping*
getHistoryRetentionPeriod	(USER INTERFACE) Returns History Retention Period for specified table.
getLatestRetentionPeriod	(USER INTERFACE) Returns Latest Retention Period for specified table.
getEndpoint	(USER INTERFACE) Returns URL for specified Primary Producer resource.
getTerminationInterval	(USER INTERFACE) Returns Termination Interval for specified resource.
setTerminationInterval	(USER INTERFACE) Changes Termination Interval for specified resource.
ping	(USER INTERFACE) Checks whether the resource is still alive.
close	(USER INTERFACE) Schedules the Primary Producer for destruction. The Primary Producer is destroyed by the Resource Framework in accordance with the rules specified above.
destroy	(USER INTERFACE) Destroys the Primary Producer resource immediately; terminates any threads and frees up any operating system resources associated with it.

3.3 Error Handling

The default handling for all errors is to stop processing, make the resource stable if possible and notify the user (see section 2.7.3). The error conditions which require a non-default response are:

Fault	Response
Resource Framework cannot contact registry to keep a producer registered.	Log the error and try again later.
Resource Framework finds that the registry has already unregistered the producer.	Log the error and re-register the producer's tables.
Data error in tuple means that continuous consumer's SQL expression can't be calculated.	Log the error and skip the tuple.
Network I/O error trying to stream tuples to a continuous consumer.	Log the error, close the connection, try again later.
Network (streaming) connection to continuous consumer has closed prematurely.	Log the error and attempt to re-open the connection.
An error occurs during a *close* operation.	Log the error, but don't return a fault.

3.4 External Objects

3.4.1 Configuration Parameters

- Location of related EGEE software (security and SQL libraries) and external software (Tomcat, MySQL, HSQLDB and Log4j).
- URL of Producer Factory Service.

- Location of security certificates/keys (for user-to-service and service-to-service authorization).
- Location of access control list (hosts permitted/not permitted to connect to Primary Producer Service).
- Logging settings and location of log files (for reporting on status of Primary Producer Service).
- Default user name, password and JDBC connection string for database tuple storage in a Primary Producer Resource. Both administrative (for creating/dropping databases) and non-administrative (for other access) accounts are required.
- Maximum settings for Termination Interval for a Primary Producer Resource.
- Maximum settings for History and Latest Retention Periods.
- Maximum number of tuples that can be stored in the tuple-stores in a Primary Producer Resource.

4 SECONDARY PRODUCER

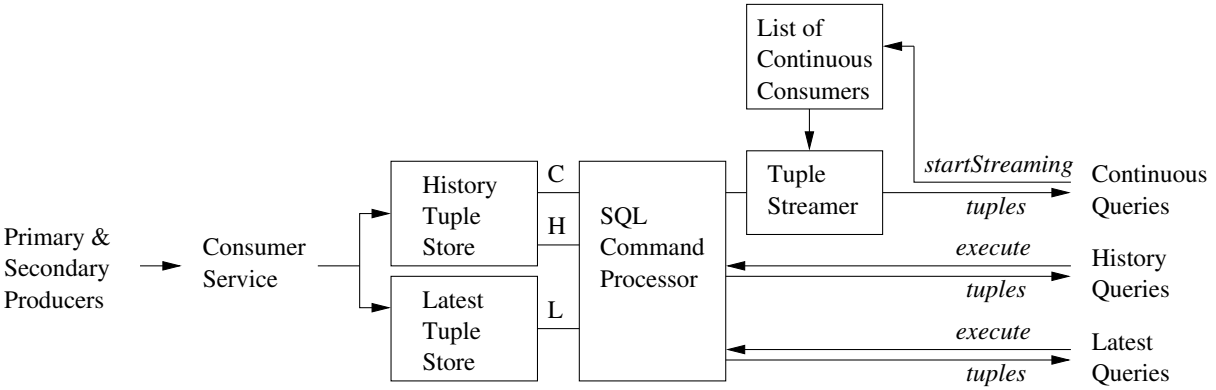
4.1 DESCRIPTION

4.1.1 SERVICE COMPONENTS

The Secondary Producer Service allows a user to *re-publish* rows, by running a query on a virtual table and publishing the results back to the same table. Its value is in merging together rows from multiple Primary Producers into a single place: if it has no predicate, that represents a copy of the entire virtual table. There are three reasons why this can be useful:

- if its tuple-storage is a real database, it can act as an archiver for the virtual table;
- if it re-publishes more than one table, it can answer queries involving joins which could not be distributed across multiple Primary Producers, because the tables are now all in one database;
- Secondary Producers can improve the scalability of the information system by reducing the load on the Primary Producers.

Secondary Producers consume from Primary and Secondary Producers. The picture below shows the principal components of a Secondary Producer.



4.1.2 RESOURCE LIFECYCLE

The resource lifecycle of a Secondary Producer is the same as a Primary Producer except that it has to manage (create, keep alive and close) the consumers it creates for each declared table.

4.1.3 REGISTRATION

As Primary Producer, except that a Secondary Producer is registered as a *re-publisher*, and there is no Latest Retention Period to set when declaring a table.

4.1.4 SUPPORTED QUERIES

As Primary Producer, except a Secondary Producer is not obliged to support continuous queries.

4.1.5 PUBLISHING, STORING AND DELETING TUPLES

As Primary Producer, except:

- There is no *insert* operation; tuples come from a built-in consumer running a "SELECT * FROM *predicate*" query. The mediator ensures that a Secondary Producer doesn't try to answer its own queries.
- A Secondary Producer respects the Latest Retention Period of any tuples it receives. It does not, however, wait for tuples to expire when it is closing. This is because the Secondary Producer itself must stop consuming immediately.

4.1.6 MESSAGES FROM CONSUMER SERVICE

As Primary Producer.

4.2 INTERFACE

The Secondary Producer service provides the following operations:

createSecondaryProducer	(FACTORY SERVICE INTERFACE) Creates a new Secondary Producer resource and registers it with the Resource Framework, with the specified termination interval; • query type (latest and/or history), storage type (memory, file or database) and handling for slow consumers are fixed at this point;
declareTable	(USER INTERFACE) Registers this producer in the registry as a re-publisher for a specified table, which must already exist in the schema; • must declare a predicate to restrict the tuples it will publish (this is an SQL WHERE clause which is limited (by the Registry mediator) to a union (AND) of equality constraints on columns in the table); may be empty • History Retention Period fixed at this point; • Corresponding table is created in resource's tuple-store; if this already exists, its structure is checked and any existing tuples will be automatically available for consumer queries until they expire; • Creates and starts a continuous consumer running a "SELECT * WHERE *predicate*" query on the specified table.
startStreaming	(SYSTEM INTERFACE) Requests a producer to start streaming new tuples to the specified consumer, for a given SQL SELECT query and a given table; • consumer may also request existing contents of producer's tuple store, newer than a certain date/time, to be streamed first;
stopStreaming	(SYSTEM INTERFACE) Requests a producer to stop streaming new tuples to the specified consumer;
execute	(SYSTEM INTERFACE) Requests a producer to execute a consumer's SQL SELECT query on the contents of the producer's tuple store and return the answer immediately; • request will be either for latest or all (history) tuples;
showSignOfLife	(USER INTERFACE) Updates the contact time for the resource: if it exceeds its termination interval without any contact, the Resource Framework will unregister and close it; • the contact time is also updated following any operation except *ping*
getHistoryRetentionPeriod	(USER INTERFACE) Returns History Retention Period for specified table.
getEndpoint	(USER INTERFACE) Returns URL for specified Secondary Producer resource.
getTerminationInterval	(USER INTERFACE) Returns Termination Interval for specified Secondary Producer resource.
setTerminationInterval	(USER INTERFACE) Changes Termination Interval for specified Secondary Producer resource and its consumer.
ping	(USER INTERFACE) Checks whether the resource is still alive.
close	(USER INTERFACE) Same as *destroy* for a Secondary Producer.
destroy	(USER INTERFACE) Destroys the Secondary Producer resource and its consumer immediately; terminates any threads and frees up any operating system resources associated with it.

4.3 ERROR HANDLING

The default handling for all errors is as in a Primary Producer, except for errors relating to its internal consumer, which are handled as follows:

Fault	Response
Secondary Producer Service can't connect to the Consumer Service, for its own consumer (or any error relating to *setTerminationInterval* on the consumer).	Log the error and try again later.
Secondary Producer Service finds its Consumer resource has been destroyed.	Log the error and re-create the Consumer.

4.4 External Objects

4.4.1 Configuration Parameters

As in Primary Producer except there is no setting relating to Latest Retention Period.

5 On-demand Producer

5.1 Description

5.1.1 Service Components

The On-demand Producer Service allows a user-application to service one-time queries on a virtual table in real time. There is no tuple storage and very little functionality in the service itself. Instead, it hands off all queries to a user-application that is expected to process the query and return the resulting tuples.

5.1.2 Resource Lifecycle

The resource life-cycle of an On-demand Producer is exactly the same as the other producer types, although there are no query-type and storage-type options to set when the resource is created. The On-demand Producer resource stores the connection details of the user-application which handles any queries directed to it.

5.1.3 Registration

On-demand Producers are declared (*declareStaticTable*) in the Registry like any other producer, so the mediator can find them. There are, however, no tuple retention periods to set.

5.1.4 Supported Queries

On-demand Producers only support static queries.

5.1.5 Publishing Tuples

On-demand Producers have no tuple-storage: tuples are only returned by the user-application in response to a specific query. The query is run synchronously by the service: it expects the user-application to be listening on the URL/port specified when the On-demand Producer was created. It writes the query to the port using the specified protocol (to be decided: probably http(s) or raw tcp depending on security implications), and expects an immediate reply. Returned tuples must be in the form of an XML ResultSet, defined as follows (where * indicates repetition):

```
<?xml version = '1.0' encoding = 'UTF-8' standalone = 'no'>
<XMLResultSet>
    <rowMetaData>
        <colMetaData>column-name</colMetaData>*
    </rowMetaData>
    <row>
        <col>column-value</col>*
    </row>*
    <RGMAWarning>
        <message>message</message>
    </RGMAWarning>
</XMLResultSet>
```

The service does not add R-GMA time-stamps or retention periods to the tuples. For this reason, Secondary Producers never consumer from On-demand Producers.

5.1.6 CONSUMER SERVICE MESSAGES

Consumer Services send an *execute* message to request an On-demand Producer to run a static query. Results are returned immediately. R-GMA places no restrictions on the subset of SQL (or data types) supported by an On-demand Producer.

5.2 INTERFACE

The On-demand Producer service provides the following operations:

createOnDemandProducer	(FACTORY SERVICE INTERFACE) Creates a new On-demand Producer resource and registers it with the Resource Framework, with the specified termination interval; • On-demand Producers only support static queries and have no tuple storage; • The user must specify contact details (URL/port/protocol) for a user-application that will answer any SQL queries directed to the producer.
declareStaticTable	(USER INTERFACE) Registers this producer in the registry as a publisher for a specified table, which must already exist in the schema; • must declare a predicate to restrict the tuples it will publish (this is an SQL WHERE clause which is limited (by the Registry mediator) to a union (AND) of equality constraints on columns in the table); may be empty;
execute	(SYSTEM INTERFACE) Requests a producer to forward a consumer's SQL SELECT query for processing by the producer-application and return the answer immediately;
showSignOfLife	(USER INTERFACE) Updates the contact time for the resource: if it exceeds its termination interval without any contact, the Resource Framework will unregister and close it; • the contact time is also updated following any operation except *ping*;
getEndpoint	(USER INTERFACE) Returns URL for specified On-demand Producer resource.
getTerminationInterval	(USER INTERFACE) Returns Termination Interval for specified resource.
setTerminationInterval	(USER INTERFACE) Changes Termination Interval for specified resource.
ping	(USER INTERFACE) Checks whether the resource is still alive.
close	(USER INTERFACE) Same as *destroy* in an On-demand Producer.
destroy	(USER INTERFACE) Destroys the On-demand Producer resource immediately; terminates any threads and frees up any operating system resources associated with it.

5.3 Error Handling

The default handling for all errors is to stop processing, make the resource stable if possible and notify the user (see section 2.7.3). The error conditions which require a non-default response are:

Fault	Response
Resource Framework cannot contact registry to keep a producer registered.	Log the error and try again later.
Resource Framework finds that the registry has already unregistered the producer.	Log the error and re-register the producer's tables.

5.4 External Objects

5.4.1 Configuration Parameters

As Primary Producer except there are no parameters related to tuple storage and tuple retention periods.

6 CONSUMER

6.1 DESCRIPTION

6.1.1 SERVICE COMPONENTS

The Consumer Service allows a user to run an SQL query on one or more virtual tables in the R-GMA schema. It runs the queries on the user's behalf, contacting all of the producers necessary to answer the query. The results are collected into an internal tuple-store, from which the user can subsequently retrieve them. A continuous consumer also has a *tuple-receiver* to allow tuples to be *pushed* asynchronously into its tuple-store, by the producers.

6.1.2 RESOURCE LIFECYCLE

A new Consumer Resource is created when a user calls the Consumer Factory Service's *createConsumer* operation. The resource holds the private data and threads for a single consumer instance. Each consumer represents a single query which is specified as an SQL SELECT statement by the user, when the consumer is created.

Each Consumer resource has a Termination Interval that is a time interval within which the user must make contact with the consumer service, in order to keep the resource alive and maintain its entry (if any) in the registry. The Termination Interval is set by the user when the resource is created (the service imposes a maximum value) and can be subsequently changed by calling *setTerminationInterval*. The user should call *showSignOfLife* perodically to keep the consumer alive.

The resource is destroyed after the user sends a *close* or *destroy* request, or the resource exceeds its termination interval without any contact from the user.

6.1.3 MEDIATION AND QUERY PLANNING

Most consumers allow the R-GMA Registry Service to identify appropriate producers to answer their query in a process called *mediation* (see 7.1.4). This is the whole purpose of the R-GMA registry, and allows R-GMA to ensure that a query is answered completely and correctly. However a consumer may choose to by-pass the mediation process by providing a list of producers itself. A mediated query is called a *global* query, and an un-mediated query is called a *direct* query. In either case, if there are alternative sets of producers capable of answering the query, the Consumer Service must choose the "best" one according to some criteria. This is called *query planning* and the criteria used are implementation-dependent. In the standard implementation, Secondary Producers are preferred, if available, and the one with the fastest response to a *ping* operation will be selected.

The user can request the consumer to run a global query against more than one VO, by providing a list when the consumer is created. In that case, the consumer contacts the registries for each VO during the mediation process. Duplicate producers are then removed from the resulting list (they will have the same URL and resource ID in each registry). *Some aspects of this process (e.g. how to ensure the resulting list of producers will give a correct and complete answer) are still not worked out.*

6.1.4 QUERY TYPES

There are four types of query: *continuous*, *latest*, *history* and *static*. The first three types (continuous, latest and history) can optionally take a *TimeInterval* parameter.

The mediator can determine the set of query types that a particular producer supports from its entry in the registry.

A continuous query causes all new tuples matching the query to be auomatically streamed to the consumer when they are inserted to the virtual table. If a TimeInterval parameter is specified, all existing tuples newer than (*now - TimeInterval*) will additionally be returned when the query is first started, although each producer only undertakes to store old tuples in its history store up to its History Retention Period for that table. The mediator will try to ensure the producer(s) it picks can cover the time interval, and warn if it can't.

A latest query returns the latest version of all tuples matching the query that have not exceeded their Latest Retention Period. In addition, if a TimeInterval is specified, only tuples that are newer than (*now - TimeInterval*) will be returned. Whether or not you specify a time interval, you wil never get tuples that have exceeded their Latest Retention Period.

A history query returns all available versions of tuples matching the query. A producer only undertakes to retain old versions of tuples up to its History Retention Period for the table. If a TimeInterval parameter is specified, only tuples that are newer than (*now - TimeInterval*) will be returned. The mediator will try to ensure the producer(s) it picks can cover the whole time interval, and warn if it can't.

A static query is handled by an On-demand Producer like a normal one-off database query. There are no time-stamps or retention periods associated with static queries.

All R-GMA queries must be valid SQL SELECT statements (See chapter 10). Continuous queries are restricted to *simple queries*, as defined in 10.1.1. Latest, history and and static queries can be more complex (see 10.1.2).

6.1.5 STREAMING

Continuous consumers require a mechanism to allow producers to push tuples asychronously into their tuple storage. This *tuple-receiver* consists of some kind of implementation-dependent server listening on a specific port (see 2.11.1). For efficiency reasons, streaming goes through a raw TCP socket, rather than the Web Services interface. See chapter 9 for the security implications of this. Tuples are written to the streaming port by producers (format to be decided, but probably as ResultSets), preceded by the resource ID, as a 32-bit (network byte ordered) integer, to the streaming port, without acknowledgement. The tuple-receiver is allowed to block (just by not reading from the streaming port) when its tuple-buffer is full - the size of the buffer is a site-configurable parameter. Producers are expected to retry later, but see section 3.1.5 for the handling of slow consumers.

6.1.6 REGISTRATION

Continuous consumers are registered along with their query details, when they are created. This allows the registry to notify continuous consumers (using *addProducer* and *removeProducer* when new producers should be added to (or removed from) the set of producers from which they are streaming. It is up to the consumer to send a *startStreaming* message directly to a new producer about which it has been notified.

6.1.7 STARTING AND STOPPING QUERIES

Users run queries by creating a new consumer and calling *start*. The query is run on the user's behalf by the consumer service, and the user can check its progress by calling *isExecuting*. When tuples are available in the consumer's tuple-store, the user can call *pop* or *popAll* to retrieve them; both return an empty set if the tuple-store is empty (the user can also call *count* to see how many tuples are in the tuple-store). If the user wants to abort a query, they can call *abort*, followed by *isExecuting* to check when it has stopped. The user can subsequently call *pop* to retrieve any tuples from the consumer's

tuple-store, but they may only get a partial result. A query can be safely re-started after a call to *abort*, but the consumer's tuple-store is cleared each time a query re-starts.

6.2 INTERFACE

The Consumer service provides the following operations:

createConsumer	(USER INTERFACE) Creates a new Consumer Resource and registers it with the Resource Framework, with the specified termination interval; • query string, query type and time interval (if appropriate) are fixed at this point; see 10.1 for limitations on the query; • Countinuous consumers are registered with the registry. • User also specifies which VO's they wish to query: this controls which registries are contacted.
start	(USER INTERFACE) Starts query: • contacts mediator to identify producers which can answer the query, if none are supplied by the user, then decides the best ones to contact; • sends "execute" (for latest/history queries) or "startStreaming" (for continuous queries) to the selected producers; • producer services push selected tuples into Consumer's internal tuple storage; • user must specify a time interval after which the query will be automatically aborted; • returns immediately;
isExecuting	(USER INTERFACE) Checks whether a query is still executing.
abort	(USER INTERFACE) Terminates query processing. Continuous consumers send a *stopStreaming* message to any producers from which they are streaming. Consumers can *pop* partial results, but tuple-store is cleared if query is re-started.
hasAborted	(USER INTERFACE) Checks if consumer has aborted (for any reason, including query time-out).
count	(USER INTERFACE) Returns the number of tuples available in the consumer's tuple-store (note that this may not be the complete result for the query).
pop	(USER INTERFACE) Retrieves tuples from the consumer's tuple-store; • always pops complete tuples; • user specify maximum number of tuples to be returned with each call; • quietly returns nothing if no tuples available (does not throw an exception);
popAll	(USER INTERFACE) As *pop*, but returns entire contents of tuple-store: user-application is assumed to be able to cope with the amount of data returned.
addProducer	(SYSTEM INTERFACE) Sent by registry to notify a running continuous consumer about a new producer from which it should stream tuples. Consumer is expected to send a *startStreaming* message directly to the producer.
removeProducer	(SYSTEM INTERFACE) Sent by registry to notify a running continuous consumer that a producer from which it is streaming tuples has been unregistered. Consumer should abandon any attempts to contact it (but should not send a *stopStreaming* message).
showSignOfLife	(USER INTERFACE) Updates the contact time for the resource: if it exceeds its termination interval without any contact, the Resource Framework will unregister and close it; • the contact time is also updated following any operation except *ping*
getEndpoint	(USER INTERFACE) Returns URL for specified consumer resource.
getTerminationInterval	(USER INTERFACE) Returns Termination Interval for specified consumer resource.
setTerminationInterval	(USER INTERFACE) Changes Termination Interval for specified consumer resource.
ping	(USER INTERFACE) Checks whether the consumer resource is still alive.
close	(USER INTERFACE) Schedules the consumer resource for destruction; sends a *stopStreaming* message to all relevant producers.
destroy	(USER INTERFACE) Destroys the Consumer resource immediately; terminates any threads and frees up any operating system resources associated with it.

6.3 Error Handling

The default handling for all errors is to stop processing, make the resource stable if possible and notify the user (see section 2.7.3). The error conditions which require a non-default response are:

Fault	Response
Resource Framework cannot contact registry to keep a consumer registered.	Log the error and try again later.
Resource Framework finds that the registry has already unregistered the consumer.	Log the error and re-register the consumer.
Consumer Service can't connect to Producer Service to send *startStreaming* or *stopStreaming* message.	Log the error and try again later.
Consumer Service's attempt to send *startStreaming* or *execute* message to a producer fails for any other reason.	Log the error, generate another query-plan (without this producer) and try again.
Consumer service (tuple-receiver) gets I/O error reading tuples from streaming port.	Log the error and try again later.
Data error in tuple read from streaming port.	Log the error and skip the tuple.
No data available for *pop* or *popAll*.	Return an empty set.
Abort called for query which is no longer executing.	Do nothing.
An error occurs during a *close* operation.	Log the error, but don't return a fault.

6.4 External Objects

6.4.1 Configuration Parameters

- Location of related EGEE software (security and SQL libraries) and external software (Tomcat, Log4j).
- URL of Consumer Factory Service.
- Location of security certificates/keys (for user-to-service and service-to-service authorization).
- Location of access control list (hosts permitted/not permitted to connect to Consumer Service).
- Logging settings and location of log files (for reporting on status of Consumer Service).
- Maximum setting for TerminationInterval for a Consumer Resource.
- Maximum numbers of tuple that can be stored in the tuple-store in a Consumer Resource.
- Default setting for user-specified timeout period for a running query.
- URL and port number for streaming.

7 Registry

7.1 Description

7.1.1 Service Components

The registry provides the resource discovery mechanism for R-GMA: it allows producers to announce their ability to publish rows to a virtual table, and it allows consumers (via the mediator) to find producers

which can answer their queries. The registry is essentially a database table which contains a list, for each virtual table in the schema, of publishers who are publishing tuples to that table. The details for each publisher include where to find it (URL/Resource ID), the query types it supports, whether or not it's a re-publisher, and its predicate (SQL WHERE clause specifying the subset of the table it publishes for).

The registry also contains a list of continuous consumers that want to be notified when a producer starts or stops publishing to a particular table.

Each producer and consumer entry in the registry carries a termination time after which the producer or consumer will be unregistered by the Registry Service, if there is no further contact. It is the responsibility of the Resource Framework to keep registered, all resources which it is managing.

7.1.2 RESOURCE LIFECYCLE

The Registry Service creates a separate Registry resource for each VO which wants to run a registry on its server. Registry resources are identified by VO. They are not registered with the Resource Framework, and do not time out. Registries are started up and shut down manually, through the Administration API.

7.1.3 REPLICATION

Although there is only one logical registry per VO, replicas can be made for resilience. Each replica is a Registry Resource. The Registry API has a list of locations of replicas and will use the closest working one, initially, and switch to an alternative if that one fails. Registry replicas act independently, but periodically synchronize themselves each other. This means that replicas can become inconsistent for a time, and R-GMA services must tolerate a registry informing them about producer and consumer resources which no longer exist, or failing to inform them about ones about which it does not yet know.

7.1.4 MEDIATION

The job of the Registry Mediator is to find a set of producers capable of answering a consumer's query, and meeting any additional constraints imposed by the consumer. In some instances, a query must be sent to several producers in order to obtain a complete answer. The mediator therefore returns a set of *plans*. Each plan consists of a group of producers which, as a set, can answer the query in full. The plan may also, in future, include some logic to deal with producers that can only partially answer the query (i.e. only some columns). The job of selecting the best plan to use is left to the consumer. The consumer invokes the mediator through the *getProducersForQuery* operation.

The producer-selection algorithm underlying the mediation process is explained in [5]. For the first release of R-GMA, it will works as shown below, where *simple* and *complex* queries are defined in chapter 10.

Continuous (simple)	Return Primary Producers which publish to the table in the query and which have a compatible predicate. Do not return Secondary Producers.
One-time (simple)	Return all producers (of any kind) which publish to the table in the query and which support the query type and have a compatible predicate. Omit any Secondary Producers which have predicates. Any Primary Producers go into one plan, any Secondary Producers go into separate plans for each of them.
One-time (complex)	Get all producers (of any kind) which publish to all of the tables in the query and which support the query type and have no predicate. Split up Primary and Secondary producers as above.

It is assumed, by the mediator, that Primary Producers in the same VO do not overlap, i.e. they do not publish the same keys (although this cannot be enforced). As a consequence, if there is no Secondary

Producer available for a particular query, *all* Primary Producers must be contacted, as a set, in order to obtain a complete reply.

In certain situations, the mediator may not be able to return a set of producers which are capable of providing a complete reply for a particular query. Two examples are where Primary Producers with matching predicates exist, but don't support the required (latest/history) query type, or where the user specifies a time interval (for old tuples) which no producer can cover. In these situations, the mediator returns a warning to the Consumer Service, and the Consumer Service is expected to forward it on to the user, with any tuples.

7.2 INTERFACE

The Registry service provides the following operations.

createRegistry	(ADMINISTRATION INTERFACE) Creates a new Registry Resource for the specified VO. • Connects the resource to the corresponding registry database.
addReplica	(SYSTEM INTERFACE) Requests a registry to synchronize itself with updates from the calling one.
registerProducerTable	(SYSTEM INTERFACE) Registers a particular producer as a publisher for a particular table. The publisher's details (URL, Resource ID, query type, republisher flag, termination interval and predicate) are recorded in the registry database. • Any relevant consumers registered to the table, are sent an *addProducer* message.
registerContinuousConsumer	(SYSTEM INTERFACE) Registers a continuous consumer's interest in a particular table. The consumer's details (URL, Resource ID, query string and termination interval) are recorded in the registry database.
getProducersForQuery	(SYSTEM INTERFACE) Returns (mediated) list of *all* producers able to answer a given query. Producers are split into plans, with each plans able to provide a complete and correct answer to the query, and meeting all constraints. The job of selecting the best plans is left to the caller.
unregisterProducer	(SYSTEM INTERFACE) Removes all producer's entries from the registry. • Sends a *removeProducer* message to all relevant consumers.
unregisterProducerTable	(SYSTEM INTERFACE) Removes a producer's entry for a specified table, from the registry. • Sends a *removeProducer* message to all relevant consumers.
unregisterConsumer	(SYSTEM INTERFACE) Removes consumer's entry from registry.
getPredicate	(SYSTEM INTERFACE) Returns predicate for given producer and given table (as SQL WHERE string).
getProducerConnections	(ADMINISTRATION INTERFACE) Returns list of all producers, with their details, for use by the Browser application.
updateContactTime	(SYSTEM INTERFACE) Informs registry that producer or consumer resource is still alive.
getEndpoint	(SYSTEM INTERFACE) Returns URL for specified registry resource.
ping	(SYSTEM INTERFACE) Checks whether the resource is still alive.
destroy	(SYSTEM INTERFACE) Destroys the registry resource (the registry database is closed, but not destroyed).

7.3 Error Handling

The default handling for all errors is to stop processing, make the resource stable if possible and notify the user (see section 2.7.3). The error conditions which require a non-default response are:

Fault	Response
Duplicate entry found for *addConsumer* or *addProducer*.	Log the error and replace the old entry with the new one.
Error trying to send an *addProducer* message to a consumer service.	Don't want to fail an *addProducerTable* because of this. If it's a connection failure, retry later. If the consumer resource no longer exists, remove it from the registry. In all cases, log the error, but carry on.
Error synchronizing with another registry replica.	Log the error and retry later.

7.4 External Objects

7.4.1 Configuration Parameters

- Location of related EGEE software (security and SQL libraries) and external software (Tomcat, MySQL and log4j).
- URL of Registry Factory Service.
- Location of security certificates/keys (for user-to-service and service-to-service authorization).
- Location of access control list (hosts permitted/not permitted to connect to Registry Service).
- Logging settings and location of log files (for reporting on status of Registry Service).
- List of URL's of replica schemas, for each VO.
- User name, password and JDBC connection string, for the Registry database, for each VO.
- Replication interval, for each VO.

7.4.2 Registry Database

The Registry Database contains two primary tables, the PRODUCERS table and the CONSUMERS table. All read and write access to this database comes through the Registry API.

The PRODUCERS table contains *for each table in the Schema*, the details of each Producer Resource which publishes rows to it (connection details, termination time, producer class (primary/secondary), producer type (latest/history) and predicate (subset of rows published by this producer)).

The CONSUMERS table contains *for each table in the Schema*, the details of each consumer which is running a continuous query against the table (connection details, termination time, consumer type, and predicate (subset of rows published by this producer)).

Exact details will depend on the system design and will be documented in a Database Specification document on completion of the design work.

8 SCHEMA

8.1 DESCRIPTION

8.1.1 SERVICE COMPONENTS

The schema contains the names and definitions of all of the virtual tables in a VO's virtual database. It also contains the VO's authorization rules for each table. It is implemented as a database.

The schema is managed through the Administration API. Special privileges are required to modify the schema. Tables can only be dropped when there are no producers publishing to them. Dropping tables therefore needs careful co-ordination with the Registry and is a more privileged operation than creating tables, because it must force any remaining producers to be unregistered.

8.1.2 RESOURCE LIFECYCLE

The Schema Service creates a separate Schema resource for each VO which wants to run a schema on its server. Schema resources are identified by VO. They are not registered with the Resource Framework and do not time out. Schemas are started up and shut down manually, through the Administration API.

8.1.3 REPLICATION

Although there is only one logical schema per VO, identical replicas can be made for resilience. Each replica is a Schema Resource. The Schema API has a list of locations of replicas and will use the closest working one. Replicas can service look-ups independently, but unlike the registry, modifications to the schema must carefully co-ordinated, as follows:

createTable	it must not be possible to create two tables with the same name and different column definitions, so schema replicas cannot create tables independently. Nevertheless, we don't want to hold up table creation un-necessarily, so it is acceptable for some replicas not to know about new tables straightaway. The replication design must take account of this (probably by using a master-slave architecture).
dropTable	must be co-ordinated with all registry replicas to ensure that no producers are registered for the table, and none re-register, until the operation is complete
table authorization	changes must be propagated to all schema replicas immediately.

8.1.4 CREATING TABLES

Users pass an SQL CREATE TABLE statement to the schema to create a new virtual table. The table definition must conform to the following rules.

table name	Anything permitted by SQL92 Entry Level standard
table attributes	table must have a primary key (as one or more columns)
column names	Anything permitted by SQL92 Entry Level standard
column types	see section 10.2.1
column attributes	primary key and time-stamp columns, and no others, must be NOT NULL
metadata	every tuple must have R-GMA tuple meta-data columns addded - see section 3.1.5

Producer Services may re-map R-GMA table names, column names and column types to alternatives supported by their underlying tuple-store database, provided no information is lost and the process is hidden from the user.

8.2 INTERFACE

The Schema service provides the following operations:

createSchema	(ADMINISTATION INTERFACE) Creates a new schema instance for a particular VO and registers it with the Resource Framework. Connects to existing schema database for that VO (using the supplied database connection parameters).
createTable	(AMINISTRATION INTERFACE) Creates a new virtual table in the schema. Supported column data types are specified in section 10.2.1. • Table authorization parameters specified at this point.
dropTable	(ADMINISTRATION INTERFACE) Removes a virtual table from the schema.
getAllTables	(SYSTEM INTERFACE) Returns a list of names of all virtual tables in the schema.
getTableDescription	(SYSTEM INTERFACE) Returns name, type and attributes of all columns in a specified table.
getEndpoint	(SYSTEM INTERFACE) Returns URL for specified schema resource.
ping	(SYSTEM INTERFACE) Checks whether the schema is still alive.
destroy	(SYSTEM INTERFACE) Destroys the schema resource (the schema database is closed, but not destroyed).

To be completed: additional operations are likely to be required for schema replication, and for table authorization.

8.3 ERROR HANDLING

The default handling for all errors is to stop processing, make the resource stable if possible and notify the user (see section 2.7.3). There are no error conditions which require a non-default response.

8.4 EXTERNAL OBJECTS

8.4.1 CONFIGURATION PARAMETERS

- Location of related EGEE software (security and SQL libraries) and external software (Tomcat, MySQL and log4j).
- URL of Schema Factory Service.
- Location of security certificates/keys (for user-to-service and service-to-service authorization).
- Location of access control list (hosts permitted/not permitted to connect to Schema Service).
- Logging settings and location of log files (for reporting on status of Schema Service).
- List of URLs of replica schemas, for each VO.
- User name, password and JDBC connection string, for the Schema database, for each VO.

- Replication interval, for each VO.

8.4.2 SCHEMA DATABASE

The Schema Database contains two primary tables, the PRODUCERTABLES table and the TABLECOLUMNS table. All read and write access to this database comes through the Schema API. The PRODUCERTABLES table contains a list of all of the virtual tables in the VO. The TABLECOLUMNS table contains *for each table*, the name, type and attributes of each column in the table. Exact details will depend on the system design and will be documented in a Database Specification document on completion of the design work.

9 SECURITY

This Chapter is provisional. Please do not (yet) base design work on it.

9.1 REQUIREMENTS

The security requirements for EGEE are still being defined by JRA3. The security issues discussed here are just an attempt to provide a framework for treating the security of R-GMA in a practical way. Much has been written before: see [5] for a discussion of the DataGrid project's work on security and [3] for a view-based authorization scheme.

Security is concerned with real-world people requesting or controlling access to real-world resources. We can identify four types of real-world users in R-GMA: consumer users (who request information), producer users (who provide information), site administrators (who run R-GMA services) and virtual organisations (who "own" the schema and registry). The real-world resources consist of the *operations* provided by the R-GMA services, and the *data* held in, and flowing between, the services.

At a high level, the security requirements for each type of R-GMA user are as follows.

9.1.1 CONSUMER USERS

Consumer Users simply want to be able to run queries on the information systems of any VO to which they belong. They should not be denied access to any resource (service, operation or data) to which they have been granted access by the owner, and they should be able to rely on the provenance of any results they receive. The security mechanisms of R-GMA should be as invisible as possible (in terms of procedure and performance degradation) to a user with proper authority.

9.1.2 PRODUCER USERS

Producer Users want to be able to publish data to the information systems of one or more VOs. They usually want to restrict read-access to only those users they personally authorize, and some will want to authorize down to the sub-table (rows and columns) level. They want to be sure that the integrity of their data, and sometimes the confidentiality, is preserved all the way from their application to the consumers application. They also want to ensure that no-one else can impersonate them and publish data in their name.

9.1.3 SITE ADMINISTRATORS

Site Administrators want to control who creates R-GMA services on their systems and which services are run (this is a local issue). They want to control who can connect to the services, and what operations they can request. They don't want unauthorized users to be able to compromise the quality of service they provide to legitimate users, and they don't want their systems to be used as a launch pad for attacks on their users' own systems. They will want to be able to know, down to the level of an individual, who is accessing their services and what they are doing. They will want to be able to recover their systems quickly following any attack.

9.1.4 VIRTUAL ORGANISATIONS

Virtual Organisations want to control who can add virtual tables to their schema and who can remove them. They also want to control who can publish to their virtual database (i.e. add entries to the registry). Producer Users, Consumer Users and Site Administrators may also trust Virtual Organisations to manage some aspects of authorization on their behalf, such as granting access to services and data to specific users, and possibly certifying services as trustworthy. In fact, this is one of the main reasons for forming a Virtual Organisation, but with the caveat that ultimate control of access to a resource must always rest with the owner of the resource.

9.2 SOLUTIONS

The solutions to most of these requirements involve *authentication*, *encryption* and *authorization*. These are discussed in the next sections. We don't provide an answer to denial-of-service here, as it is a project-wide issue. Nevertheless, R-GMA is designed to make it hard for legitimate users to accidentally disrupt the service provided to others, and this, together with authentication contols, may go some way to preventing this kind of attack.

9.2.1 AUTHENTICATION

Mutual authentication (guaranteeing who is at each end of an exchange of messages) is fundamental to R-GMA security. For the user, it guarantees that the service they have connected to is genuine. For the service, it is the basis of the authorization process described later. The DataGrid project came up with a mechanism (edg-java-security) for establishing mutual authentication based on digital certificates. EGEE is expected to come up with a similar solution for this project, and it must support all of the languages (Java, C, C++ and Python) used by R-GMA. Services and users will all need to have certificates acceptable to each other, so some means of creating and managing certificates must also be provided.

Authentication is carried out each time a connection is established between two R-GMA components (users or services). Since R-GMA services provide a small number of relatively complex operations, it may be an acceptable overhead to create a new connection (and thus authenticate) before each operation. Otherwise, it will be necessary to keep a secure connection open until it is no longer needed.

9.2.2 ENCRYPTION

Data in R-GMA exists on user systems, in services, and in the transfers between them. Details of what kinds of data require securing and where data is held in R-GMA, are given later in this chapter.

R-GMA provides the option of using an encrypted transport protocol (HTTPS) for all network transfers using the Web Services interfaces. The user can choose to use an unsecure protocol, but R-GMA cannot be considered to be secure if this is done in any part of an installation. Those R-GMA network transfers

that don't go through the Web Services interface (tuple-streaming and On-demand Producer queries) have special provisions to make them secure (see later in this chapter).

R-GMA does not encrypt data held within its services. If users consider their data to be too sensitive to trust R-GMA's internal security, then they may wish to encrypt the data themselves, before it leaves their system. R-GMA permits this, provided that the encrypted version of data can be stored in the standard data types supported by R-GMA. However there are some limitations on the querying of encrypted data in R-GMA (see section 10.2.1).

9.2.3 AUTHORIZATION

Authorization (controlling who can do what) is the final piece of the security puzzle. Any chain of authorization always starts with the owner of a resource, who either grants (or denies) access either directly to the end user, or to an intermediary who is trusted by both parties. In R-GMA, the services are the intermediaries, and there may be several R-GMA services involved in carrying out a single operation.

Authorization throughout R-GMA must be mutual. In one direction, this is easy to understand: authorization to connect to a service, authorization to request an operation and authorization to read data, etc. But users of services and readers of data must also trust the services they are using (and any intermediaries) to enforce the rules. This authorization can be implicit: if you don't trust a service, or any other services it uses, don't connect to it. But it it could be made explicit, e.g. by VOs providing certificates to trusted services, and this could be used to automatically build a chain of mutually authorized services between a resource provider and the end user. There is probably a strong link between this process and the proposed "quality of service" match-making enhancements to R-GMA.

Authorization is based on identity, and the basic unit of identity is a *distinguished name* (abbreviated to DN). Identity is authenticated by the possession of the private key corresponding to a particular digital certificate. It's likely that a VO will want to base its authorization schemes around groups of users (users in particular *groups* or having particular *roles*). VOMS certificates (short-lived proxy certificates) wrap a Grid certificate with additional VO credentials of Group, Role and Capabilities. R-GMA will need to adopt whatever mechanism is implemented across EGEE middleware.

Authorization to run R-GMA services is an internal issue for site administrators, and it's outside the scope of R-GMA to allow or deny this. Authorizing users to connect to services, and authorizing the creation of instances of producers, consumers, registries and schemas is also an issue for site administrators, but this is managed by the services, and is therefore an R-GMA problem. Authorizing users to modify the schema or registry is a VO issue, and authorizing users to access data itself (down to sub-table level) is a data-provider issue (although they may trust a VO to handle it for them). We must provide solutions for these too.

Since all of R-GMA's capabilities are presented as Web Services operations, we can solve the authorization problem by adding authorization steps to R-GMA at the following points:

- User connecting to a service.
- User requesting a service operation.
- User requesting access to specific data.

In addition, a user or service must decide whether or not to trust a service before connecting to it, and this may require an authorization mechanism.

Each of these steps is covered later in this chapter.

9.3 BOUNDARIES OF RESPONSIBILITY

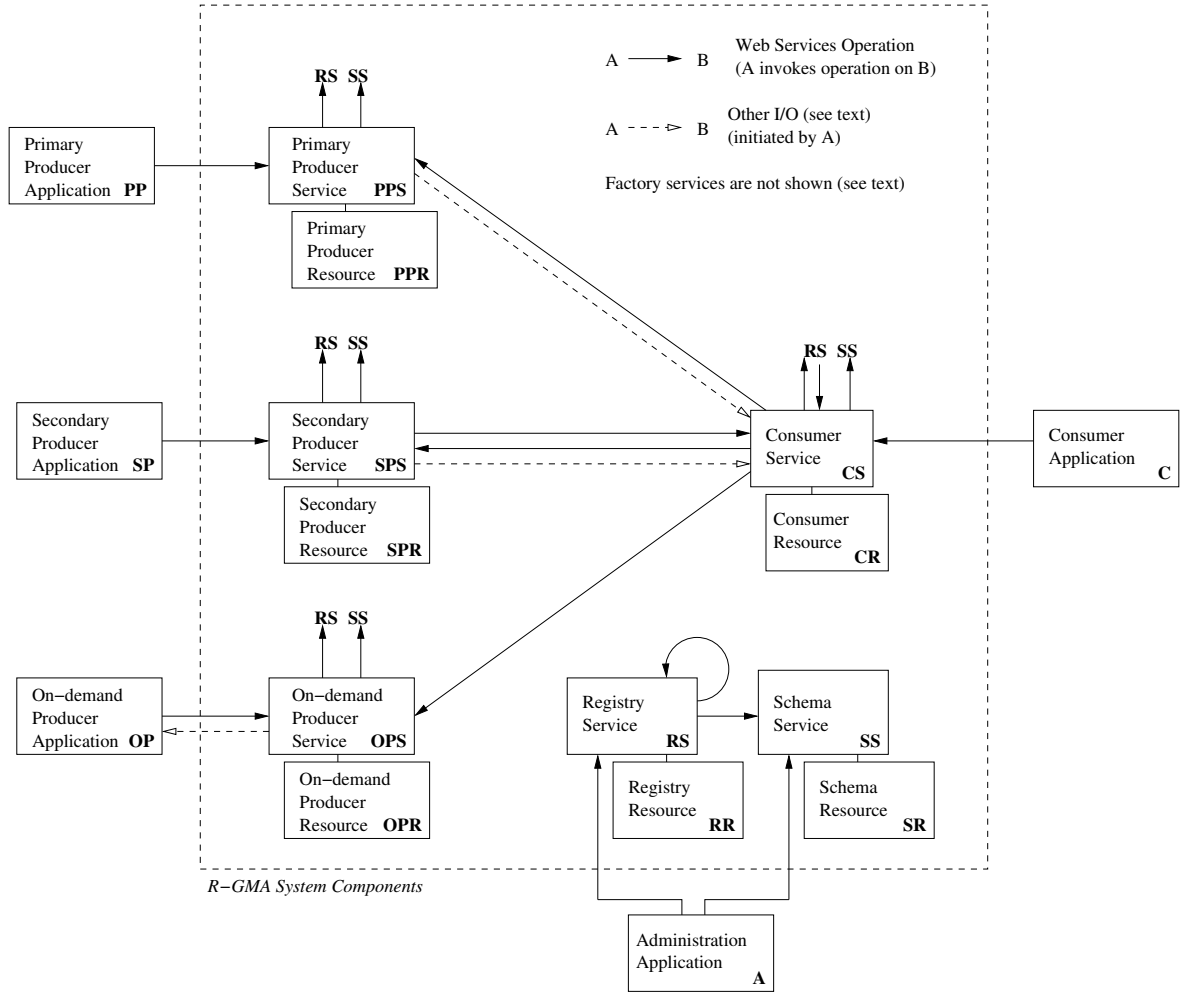
R-GMA's responsibility for data ends immediately at the user-end of the network transfer between its services and the user. R-GMA ensures that the network transfer itself is secure, and the user is genuine, by requiring the user code to authenticate itself, and by compelling it to use a secure transport protocol. In addition, if the user chooses to use the API, then the API will require the R-GMA service to authenticate itself to the user, proving it too is genuine. R-GMA *is* responsible for the security of data within its services and travelling between them.

The API has no role in protecting the services from attack. However if the API has been installed correctly, and the user has a valid certificate, the API will deal with the authentication process internally, making the security mechanisms fairly transparent to the user.

9.4 R-GMA RESOURCE OWNERSHIP

The table below shows the owner of each of the resources that make up R-GMA. The owner of each R-GMA resource is ultimately responsible for defining the security policy for it. The picture following the table shows all of the interactions between the resources.

Site that is hosting the service.	R-GMA Web Services (RS, SS, PPS, SPS, OPS, CS) and the operations they provide.
R-GMA user who is publishing information to R-GMA.	Producer Applications, Web Services Resources (PP, SP, OP, PPR, SPR, OPR) and the data in them.
R-GMA user who is querying information from R-GMA.	Consumer Applications and Web Services Resources (C, CR) and the data in them.
Virtual Organisation.	Administration Application, Registry and Schema Web Services Resources (A, RR, SR) and the data in them.



9.5 Securing Data

9.5.1 What data needs securing?

Data which may need to be secured in R-GMA consists of:

- Tuples (inserted by producers, stored by services, returned to consumers)
- Queries (query predicates can contain user data)
- Connection details (e.g. URLs, port numbers) of user-code and services
- User identification details (stored in the Schema for authorization)
- Database user names and passwords (for Registry/Schema/Tuple-stores)

9.5.2 Where is it held?

Data is held within R-GMA services, in the following places:

- The memory space of the AXIS Web Application running in the Tomcat servlet container.

- Database management systems (registry, schema and producers which use database tuple-stores). This includes data travelling over JDBC to the database, data in the database tables, and data in the database transaction logs.
- Data files of producers using file-based tuple-stores.
- Service log files (Tomcat logs, R-GMA logs)

Data is also passed across networks between R-GMA users and services. All of the network transfers are shown as arrows in the picture above.

9.5.3 How is it secured?

to be completed

The security of data within services: *How do we guard against attacks on R-GMA services that don't come through the Web Services interface (hacking into Tomcat, databases, files; security failures caused by malfunctioning code)*

The security of data in transit can be guaranteed by requiring mutual authentication of both parties involved and using an encrypted transport protocol. All R-GMA Web Services operations can be configured to require mutual authentication (for user-to-service and service-to-service connections) and to use HTTPS (with SSL encryption). There are two types of transfer in R-GMA that don't go through the Web Services interface. These are the streaming of tuples from the Primary and Secondary Producer Services to the Consumer Service, and the forwarding of a query from the On-demand Producer Service to a user-application. The former is a one-way communication, the latter is a request/response. *We still need to decide how these are going to be made secure.*

9.6 Authorizing Operations

to be completed

All R-GMA operations need some level of authorization. The table below lists all of the operations provided by R-GMA services, and the credentials required to access them. Each operation is also cross-referenced to the interfaces in the diagram (where A-B corresponds to an arrow from A to B).

Operation	*Interfaces*	*Required Authorization*
abort	C-CS	ConnectToService, etc (to be completed)
addProducer	RS-CS	
addReplica	RS-RS	
close	PP-PPS, SP-SPS, OP-OPS, C-CS	
count	C-CS	
createConsumer	SPS-CS	
createOnDemandProducer	OP-OPS	
createPrimaryProducer	PP-PPS	
createRegistry	A-RS	
createSchema	A-SS	
createSecondaryProducer	SP-SPS	
createTable	A-SS	
declareStaticTable	OP-OPS	
declareTable	PP-PPS, SP-SPS	
destroy	PP-PPS, SP-SPS, OP-OPS, C-CS, A-RS, A-SS	
dropTable	A-SS	
execute	CS-PPS, CS-SPS, CS-OPS	
getAllTables	A-SS	

getEndpoint	PP-PPS, SP-SPS, OP-OPS, C-CS, A-RS, A-SS	
getHistoryRP	PP-PPS, SP-SPS	
getLatestRP	PP-PPS	
getPredicate	PPS-RS	
getProducerConnections	A-RS	
getProducersForQuery	CS-RS	
getTableDescription	PPS-SS, SPS-SS, OPS-SS, CS-SS, RS-SS, A-SS	
getTerminationInterval	PS-PPS, SP-SPS, OP-OPS, C-CS	
hasAborted	C-CS	
insert	PP-PPS	
isExecuting	C-CS	
ping	PP-PPS, SP-SPS, OP-OPS, C-CS and all services to RS and SS	
pop	C-CS	
popAll	C-CS	
registerContinuousConsumer	CS-RS	
registerProducerTable	PPS-RS, SPS-RS, OPS-RS	
removeProducer	RS-CS	
setTerminationInterval	PP-PPS, SP-SPS, OP-OPS, C-CS	
showSignOfLife	PP-PPS, SP-SPS, OP-OPS, C-CS	
start	C-CS	
startStreaming	CS-PPS, CS-SPS	
stopStreaming	CS-PPS, CS-SPS	
unregisterConsumer	CS-RS	
unregisterProducer	PPS-RS, SPS-RS, OPS-RS	
unregisterProducerTable	PPS-RS, SPS-RS, OPS-RS	
updateContactTime	PPS-RS, SPS-RS, OPS-RS, CS-RS	

9.7 IMPLEMENTING AUTHORIZATION

to be completed

9.7.1 SITE FILTERING

Controlling connection-access to services...

9.7.2 SERVICE AUTHORIZATION (ACCESS TO OPERATIONS)

Delegation: chains of authorization; authorizing users to be users AND services to be services ("mutual authorization") - how to automate, e.g. by VO's certifying services; link with QoS mechanism

9.7.3 TABLE AUTHORIZATION

Schema-based table authorization... A prototype view-based system (which works very much like granting users access to tables through views in a normal RDBMS) is described in reference [3]...

10 SQL IN R-GMA

The query language supported by R-GMA is a subset of SQL. Data is also inserted to R-GMA using SQL statements, and SQL is used internally to manage Registry, Schema and Producer (tuple storage) databases. This chapter sets down precisely how SQL is used in R-GMA.

10.1 CONSUMER QUERIES

R-GMA parses all consumer queries, regardless of type, for mediation and security purposes. The subset of SQL understood by R-GMA is that defined by the SQL92 Entry Level standard. This ANSI standard is a pre-requisite for database drivers to claim full JDBC compliance. All R-GMA queries must conform to this standard, and they will be rejected by the parser if they do not.

The algorithm used by the mediator makes a distinction between *simple* and *complex* queries, as defined below. Continuous queries must always be simple. Latest and history queries may be complex, and any SQL command processor used by an R-GMA producer that supports latest or history queries, must support complex queries. Static queries must also conform to the standard, but the user-application that processes them in any particular On-demand Producer is not expected to implement the full standard (this, of course, implies some co-operation between the creator and users of an On-demand Producer).

10.1.1 SIMPLE QUERIES

A simple query is defined by exclusion. It is an SQL SELECT statement conforming to the SQL92 Entry Level standard, *without the following*:

- Joins: only one table is allowed
- DISTINCT, HAVING, GROUP BY, ORDER BY, UNION, INTERSECT, MINUS
- Aggregation operators such as AVG, MIN, MAX, COUNT, SUM
- Calculations, other than numerical ones involving only the operators: +, -, *, / and **
- WHERE clauses involving operators other than: =, >, <, >=, <=, and AND

10.1.2 COMPLEX QUERIES

A complex query is anything permitted by the SQL92 Entry Level standard. In the short term, R-GMA will not support the following features:

to be completed

10.2 CREATING TABLES

Virtual tables are created in R-GMA using the Schema Service's *createTable* operation, as an SQL CREATE TABLE statement conforming to the following specification (which meets the SQL92 Entry Level standard):

to be completed

10.2.1 DATA TYPES SUPPORTED

R-GMA supports the followng data types (as defined by the SQL92 Entry Level)

still to be agreed

INTEGER (signed integer, at least 32 bits)

BIGINT (signed integer, at least 64 bits)

REAL (floating point number, at least 32 bits)

DOUBLE (floating point number, at least 64 bits)

DATETIME (date as ISO8601 character string)

CHAR(*n*) (character string of fixed length *n*)

VARCHAR(*n*) (character string of variable length up to *n*)

Null values are represented by the (unquoted, case-insensitive) word NULL: empty strings are *not* considered to be NULL values. Character strings are delimited by single or double quotes. If quote characters are embedded in strings delimited by the same type of quote, then they must be duplicated (not escaped with a backslash). Encrypted values are permitted only if the encrypted version can be stored in one of the supported types. In queries, they can only be tested for equality of the encrypted versions (R-GMA services do not unencrypt user data).

10.2.2 DATA TYPE CONVERSIONS

Data flows into and out of R-GMA as strings (INSERT and SELECT statements), but numerical values are converted to the appropriate type (by the JDBC drivers) in order to carry out calculations and comparisons. The following table shows where type conversions may take place.

Input to Primary Producer	Data is received from user as an SQL INSERT statement held in a text string. Column types are read from the virtual table's entry in the schema. Values for columns with numerical types are stored in a column of the corresponding type in the tuple-store, so they can be used in column expressions. All other values are held as unquoted strings. Null values are stored as database NULLs.
Input to Secondary Producer	Data is copied unchanged from Primary Producer's tuple-store to Secondary Producer's tuple-store.
Input to On-demand Producer	Data is received from user code as an XMLResultSet, and is not changed.

Output from Primary and Secondary Producers	A new tuple is created according to the column expression in the user's select statement and returned in a ResultSet. Column values are converted according to their type, as read from the schema. Non-string values are converted back to a string representation (without quotes). Strings are enclosed in single quotes; embedded single quotes are preceded by a backslash. Null values are written as NULL (in upper case, and without quotes).
Output from On-demand Producer	The XMLResultSet is converted to a ResultSet, without interpretation, and returned to the user.

10.3 INSERTING DATA

Tuples are inserted to R-GMA by a Primary Producer in the form of an SQL INSERT statement, conforming to the following specification (which meets the SQL92 Entry Level standard):

to be completed

Tuples returned to the R-GMA On-demand Producer Service by its producer-application are in the form an XML ResultSet, as defined in section 5.1.5.

10.4 RETURNING DATA

Tuples are returned to users by the R-GMA Consumer Service in the form of *ResultSets.* The structure of these is defined in the R-GMA Interface Specification document.

10.5 DATABASE MANAGEMENT

The following database management operations are used internally by R-GMA to create and maintain the Registry and Schema databases, and the tuple-storage databases used by some Primary and Secondary producers. All interaction between R-GMA and the databases goes via JDBC. It is intended that R-GMA should be as independent as possible of the underlying database (although it will only be tested with MySQL and HSQLDB initially). This means that database-specific SQL is avoided where JDBC alternatives (e.g. for querying table metadata) are available. Care is also taken that the names given to databases, tables and columns are acceptable to all SQL92 compliant database management systems. The operations used are:

- Creating and dropping databases.
- Creating and dropping tables.
- Getting the names of all tables in a database.
- Getting the names, types and attributes of all columns in a table.
- Inserting and updating rows in a table.
- Selecting rows from a table (this includes left inner joins).

REFERENCES

[1] Apache logging services project. http://logging.apache.org/.

[2] JRA1 Design Team. Egee middleware architecture. Technical Report EDMS 476451, EGEE, 2004.

[3] JRA1 Design Team. Middleware prototype working document. Technical Report EDMS 458972, EGEE, 2004.

[4] Oasis web services resource framework tc. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[5] RAL. R-GMA security - principles, design, plans, status and caveats. Technical Report http://hepunx.rl.ac.uk/egee/jra1-uk/docs/rgmasecurity.pdf, DataGrid, 2004.

[6] SOAP Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

[7] Web services architecture working group note. http://www.w3.org/TR/ws-arch/.

[8] Web services description language. http://www.w3.org/TR/wsdl/.