

## Information Sources and Sinks in a Grid Environment

*Giovanni Aloisio<sup>1</sup>, Zoltán Balaton<sup>2</sup>, Peter Boon<sup>3</sup>, Massimo Cafaro<sup>1</sup>, Italo Epicoco<sup>1</sup>,  
Péter Kacsuk<sup>2</sup>, Thilo Kielmann<sup>3</sup>, Daniele Lezzi<sup>1</sup>*

<sup>1</sup>*Center for Advanced Computational Technologies ISUFI,  
University of Lecce and  
National Nanotechnology Lab/INFM&CNR,  
Lecce, Italy  
{giovanni.aloisio, massimo.cafaro, italo.epicoco}@unile.it*

<sup>2</sup>*MTA SZTAKI  
Computer and Automation Research Institute  
Hungarian Academy of Sciences, Hungary  
{balaton, kacsuk}@sztaki.hu*

<sup>3</sup>*Vrije Universiteit Amsterdam, The Netherlands  
{pboon, kielmann}@cs.vu.nl*



CoreGRID Technical Report  
Number TR-0022  
March 15, 2006

Institute on Systems, Tools and Environments (STE)

CoreGRID - Network of Excellence  
URL: <http://www.coregrid.net>

# Information Sources and Sinks in a Grid Environment

Giovanni Aloisio<sup>1</sup>, Zoltán Balaton<sup>2</sup>, Peter Boon<sup>3</sup>, Massimo Cafaro<sup>1</sup>, Italo Epicoco<sup>1</sup>,  
Péter Kacsuk<sup>2</sup>, Thilo Kielmann<sup>3</sup>, Daniele Lezzi<sup>1</sup>

<sup>1</sup>Center for Advanced Computational Technologies ISUFI,  
University of Lecce and  
National Nanotechnology Lab/INFM&CNR,  
Lecce, Italy  
{giovanni.aloisio, massimo.cafaro, italo.epicoco}@unile.it

<sup>2</sup>MTA SZTAKI  
Computer and Automation Research Institute  
Hungarian Academy of Sciences, Hungary  
{balaton, kacsuk}@sztaki.hu

<sup>3</sup>Vrije Universiteit Amsterdam, The Netherlands  
{pboon, kielmann}@cs.vu.nl

*CoreGRID TR-0022*

March 15, 2006

## Abstract

The grid computing paradigm is strongly based on an infrastructure built with the main goal to provide seamless, dependable, consistent, and pervasive access to distributed resources and services managed by cooperating virtual organisations. These grid components are often sources and/or sinks of information, which in turn can be static, quasi-static or dynamic in nature. In order to fully exploit grid infrastructures we must be able to maximise the use of this information to foster development of next generation grid-aware applications, taking into account that increasingly complex patterns range from advertising services and features to annotating and storing metadata to enable advanced workflows. In this context, an information cache mediator component plays a key role. We present such a component, designed to provide a uniform interface to access several kinds of different data originating from information services, monitoring services and application-level meta-data. A caching mechanism also allows delivering the information to applications and/or components that need it really fast.

## 1 Introduction

Flexible, secure and coordinated resource sharing among VOs requires the availability of an information rich environment to support resource discovery and decision making processes. Indeed, distributed computational resources, services and VOs can be thought of as sources and/or potential sinks of information. The data produced can be static or dynamic in nature, or even dynamic to some extent. Depending on the actual degree of dynamism, information is better handled by a Grid Information Service (static or quasi-static information) or by a Monitoring Service (highly dynamic information).

It is important to recall here the key role of information, and thus of Grid Information Services. High performance execution in grid environments relies on timely access to accurate and up-to-date information related to distributed

---

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

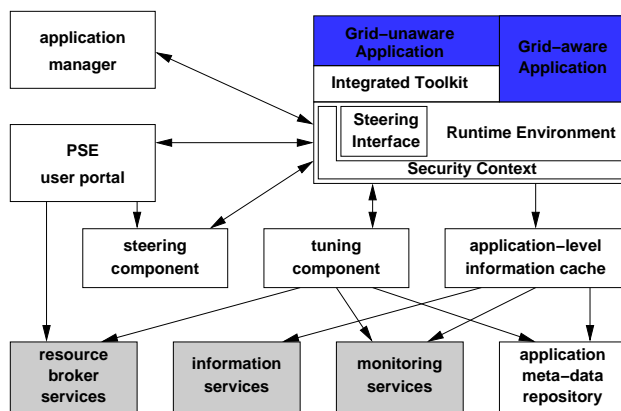


Figure 1: Grid component architecture [1]

resources and services: experience has shown that manual of default configuration hinders application performance. A so called grid-aware application can not even be designed and implemented if information about the execution environment is lacking. Indeed, an application can react to changes in its environment only if these changes are advertised. Self-adjusting, adaptive applications are natural consumers of information produced in grid environments.

However, making the relevant information available on-demand to applications is nontrivial. Care must be taken, since the information can be (i) diverse in scope, (ii) dynamic and (iii) distributed across one or more VOs. Information about structure and state of grid resources, services, networks etc. can be challenging to obtain. As an example, let us think about the problem of locating resources and/or services. Since grid environments are increasingly adopting architectures that are distributed and rather dynamic collections of resources and services, discovery of both resources and services becomes a difficult and complex task on large-scale grids. Thus, it is immediately evident the fundamental role of a Grid Information Service. It is, among many others, a key service that must be provided as part of a grid infrastructure/middleware.

The Globus Toolkit [17] version 2.x provides an LDAP based Information Service called Metadata Directory Service (MDS-2). This service has been deployed and used on many grid projects, both for production and development. However, the performances of the MDS Information Service were not satisfactory enough, and the Globus project provided users with a new version of this service, available in the Toolkit version 3.x and called Monitoring and Discovery Service (again, MDS-3) [18].

Version 3.x of the Globus Toolkit has been based on the Open Grid Service Infrastructure (OGSI) and the Open Grid Service Architecture [19] specifications. The MDS has been developed using Java and the Grid Services framework [21], but this service has not seen widespread adoption and deployment because it will change once again in the upcoming Globus Toolkit version 4. The new version, MDS-4, will be based on the emerging Web Service Resource Framework (WSRF) specification [20].

The management of a wide range of information besides being challenging, is burdening new grid-aware application. Moreover, the availability of information also depends on the underlying information and monitoring systems, therefore their performances are critical. Due to this issues, the CoreGrid partners of task 7.2 are actively developing a suite of components that mediate between applications and system software [1].

These Mediator Components will be derived from the ongoing developments of the partner institutions involved in this task. The research is focused on the definition and implementation of a novel grid component architecture; the aim is to foster integration and linking of different grid components with minimal effort, by providing a simple, well defined glue layer between all kind of components. The envisioned architecture also includes a tools framework, consisting of an integrated components Grid platform, a component support toolkit, and a generic problem-solving toolkit, as in Figure 1.

The proposal for a mediator component toolkit includes an application-level information cache mediator component. This component will be designed to provide a uniform interface to access several kinds of different data originating from information services, monitoring services and application-level meta-data. Moreover, a caching mechanism will allow delivering the information to applications and/or components that need it really fast. We are now jointly collaborating to develop an application-level information cache mediator component. Our activities also include the

integration of the iGrid information service [6] [3] and the Mercury monitoring service [4], to provide the envisioned grid component architecture with resource and service discovery capabilities.

The paper is organized as follows. Section 2 presents the iGrid Information Service and the Mercury Monitoring Service. These are the initial information sources/sinks that have been selected for integration. We describe the information cache mediator component in Section 3 and draw our conclusions in Section 4.

## 2 Information sources

### 2.1 iGrid Information System

The iGrid Information System, developed by with CACT/ISUFI University of Lecce Italy within GridLab Project, is a novel Grid Information Service. Previous work [2] has been done to extend the Globus Toolkit Monitoring and Discovery Service (MDS). However, due to lack of performances, a new approach based on the relational model has been investigated and implemented [6].

Among iGrid requirements there are performance, scalability, security, decentralized control, support for dynamic data and user supplied information. The iGrid Information Service has been specified and carefully designed to meet these requirements; indeed, the package provides the following features:

- Web service interface;
- Distributed architecture;
- Based on relational DBMS back-end;
- Fault tolerance;
- Support for Globus GSI;
- Support for TLS binding to DBMS;
- Support for GridLab Authorization service (GAS) and for Administrator defined ACL;
- Support for heterogeneous relational DBMS;
- Support for GridLab Mercury logging service;
- Builds on Linux, Mac Os X, tru64 and irix;
- Easy to extend by adding new information providers;
- Extreme performances.

As shown in Fig. 2, the iGrid distributed architecture is based on two kind of nodes, the *iServe* and the *iStore* GSI enabled Web services. The *iServe* collects information related to a specific computational resource, while the *iStore* gathers information coming from registered *iServes*. On each resource, a pool of information providers is installed, they are in charge of extracting system information from the resource. The current architecture allows *iStores* to register themselves to other *iStores*, thus creating distributed hierarchies and improving the fault tolerance of the entire system. The iGrid team is also investigating a possible implementation of a peer-to-peer overlay network based on one of the current state of the art distributed hash table algorithms in order to improve iGrid scalability; CAN [7], Chord [8], Kademia [9], Pastry [11], Tapestry [12], Viceroy [10] are just some of the peer-to-peer algorithms that are currently under investigation. The Information Service is based on a relational DBMS (PostgreSQL is currently used as back-end) and can handle both information extracted directly from a computational resource (through the activation of an appropriate information provider), and information directly supplied by users. The Web service interface is based on the gSOAP toolkit and on two packages developed within the ISUFI/CACT, the GSI plug-in for gSOAP and the GreC library. Currently, the iGrid Information System can handle information related to:

**System** belongs to this class information like operating system, release version, machine architecture etc;

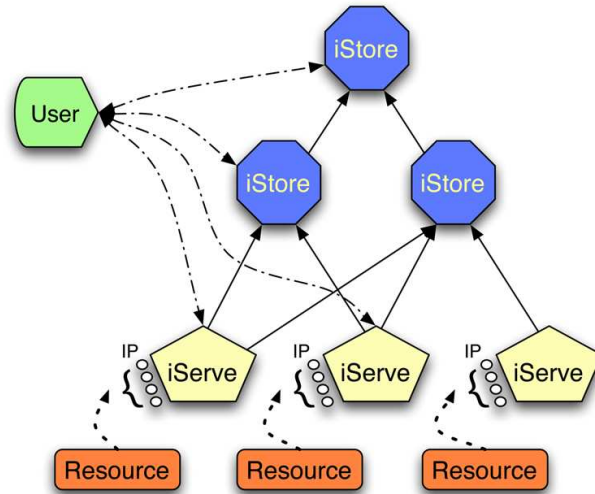


Figure 2: iGrid hierarchical architecture [2]

**CPU** for a CPU, static information like model, vendor, version, clock speed is extracted; but also dynamic information like idle time, nice time, user time, system time and load is provided;

**Memory** related to memory, static information like RAM amount, swap space size is available. Dynamic information is related to available memory and swap space;

**File Systems** static as well dynamic information is extracted; some examples include file system type, mount point, access rights, size and available space;

**Network Interfaces** information that belongs to this category includes: network interface name, network address and network mask;

**Local Resource Manager** the information belonging to this category can be further classified as belonging to three different subclasses: information about queues, about jobs and static information about Local resource Management System (LRMS). Some examples of information that can be extracted is: LRMS type and name; queue name and status, number of CPU assigned to the queue, maximum number of jobs that can be queued, number of jobs queued, etc; job name, identifier, owner, status, submission time etc. Currently information providers for OpenPBS and Globus Gatekeeper are available;

**Certification Authorities** information related to trusted Certification Authorities include certificate subject name, serial number, expiration date, issuer, public key algorithm etc.

Information can be also supplied by users by invoking the appropriate Web service register methods. The user supplied information include

**Firewall** name of the machine where the firewall is installed on, administrator name, the range of open ports allowed to pass through the firewall;

**Virtual Organization** information related to VO can be used to automatically discover which resources belong to a given VO. In this category we have VO name, resource type, a help desk phone number, help desk URL, job manager, etc;

**Service and Web Service** this information can be used for service or Web service discovery; information like service name, owner, description, WSDL location, keyword is available.

Of course, this set of information is not meant to be static, the iGrid schema will continue to evolve and will be extended to support additional information as required by the Grid community.

One of the most important requirements for grid computing scenarios is the ability to discover services and web/grid services dynamically. Services in this context refers to traditional unix servers. The iGrid system provides users and developers with the following functionalities: register, unregister, update and lookup. More than one instance for each service or web service can be registered. The following information is available for services: logical name, instance name, service description, default port, access URL, distinguished name of the service publisher, timestamps related to date of creation and date of expiration of the published information. For web services, relevant information includes logical name, web service description, WSDL location (URL), web service access URL, distinguished name of publisher and timestamps related to date of creation and date of expiration of the published information. Information related to firewalls is strictly related to service information. As a matter of fact, before registering a service, developers will query iGrid to retrieve the range of open ports available on a specified computational resource. This is required in order to choose an open port, allowing other people/services to connect to a registered service. The information available includes firewall hostname, open ports, time frame during which each port (or a range of ports) is open, the protocol (TCP/UDP) used to connect to these ports, the distinguished name of the firewall administrator, and timestamps related to date of creation and date of expiration of the published information. The iGrid implementation includes system information providers outputting XML, while user information is directly supplied by the user simply calling a Web service registration method.

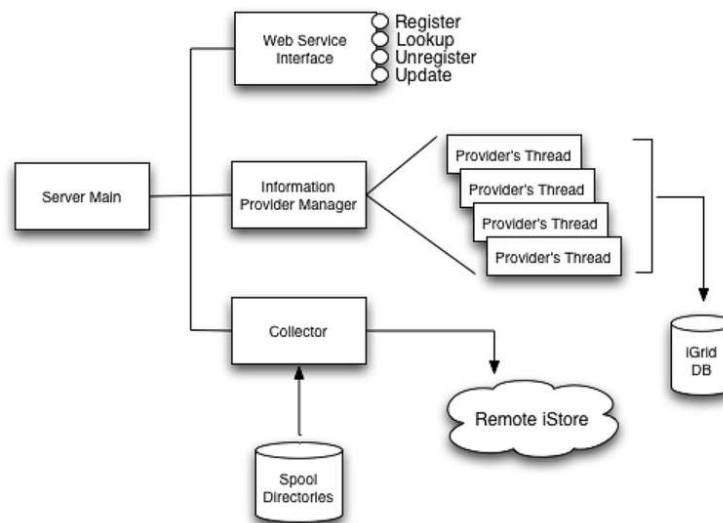


Figure 3: iGrid software architecture [2]

As shown in Fig. 3 iGrid software relies on three main threads:

- *Web Service Interface* this thread is in charge of serving all of the SOAP requests coming from SOAP clients;
- *Collector* this threads handles the communications needed to publish the iServe information into registered iStores. Periodically, the *Collector* will send to all of the registered iStores the information supplied by users or extracted by information providers. In case of failure of an iStore, iServes remove temporarily the faulty iStore from their registration list. Periodically, the iStore list is updated by adding previously removed iStores when iStores are available again. In this case, the local DBMS is dumped and immediately sent to newly added iStores;
- *Information Provider Manager* this threads will activate one thread for each information provider listed in the iGrid configuration file. Upon information provider termination the extracted information will be immediately stored into the local database and copied in an appropriate pool directory for further publication on registered iStores by the *Collector* thread.

iGrid uses a push model for data exchange: information extracted from resources is stored on the local DBMS, and sent periodically to registered iStores, while user supplied information is immediately stored on local DBMS and sent

to registered iStores. Thus, an iStore has always fresh, updated information, and does not need to ask iServes for information. Moreover, each information is tagged with a time to live that allows iGrid to safely remove stale information from the DBMS as needed. Indeed, on each user lookup, data clean-up is performed before returning to the client the requested information. When iGrid starts, the entire DBMS is cleaned up. Thus the user will never see stale information. The frequency of system information forwarding is based on the information itself, but we also allow defining a per information specific policy. Currently, system information forwarding is based on the rate of change of the information itself. As an example, information that does not change frequently or change slowly (e.g. the amount of RAM installed) does not require a narrow update interval. Interestingly, this is true even for the opposite extreme, i.e., for information changing rapidly (e.g., CPU load), since it is extremely unlikely that continuous forwarding of this kind of information can be valuable for users, due to information becoming quickly inaccurate. Finally, information whose rate of change is moderate is forwarded using narrow update intervals. We have found that the push model works much better than the corresponding pull model (adopted, for instance, by the Globus Toolkit MDS) in grid environments. This is due to the small network traffic volume generated from iServe to iStore servers: on average, no more than one kilobyte of data must be sent.

The system is fault tolerant for the following aspects: (i) the system is aware of iStore failures and as soon as iStores become available, iServes will send them updated, fresh dumped information; (ii) because of its distributed and hierarchical architecture it is possible to implement a primary/backup approach by mirroring an iStore configuring all of the iServes to publish their information simultaneously on two different iStores. More in detail several primary-backup protocols are currently being considered, namely, Alsberg and Day [13], the Tandem protocol [14], HA-NFS [15] and the non-blocking protocol [16]. The iGrid Web service uses the libxml2 library to parse XML documents and information providers extract information from resources using system calls and tools related to each specific platform supported.

### 2.1.1 Implementation Details

In this section we describe iGrid data access through GRelC libraries, the information model adopted and related information schema represented by means of ER diagrams.

**The GRelC libraries** The GRelC project supplies applications with a bag of services for data access and integration in a grid environment. In particular the two libraries heavily used in the iGrid project are:

- Standard Database Access Interface (libgrelcSDAI-v2.0);
- MultiQuery (libgrelcMultiQuery-v1.0).

The former provides a set of primitives to transparently get access and interact with different data sources. It offers a uniform access interface to several DBMSs exploiting a set of wrappers (that is, dynamic libraries providing dynamic binding to specific DBMSs) which can be easily plugged into our system. It is worth noting here that the SDAI library must take into account and solve the DBMS heterogeneity (different back-end errors, APIs and data types), hiding all of the differences by means of a uniform layer. To date, SDAI wrappers are developed for PostgreSQL, MySQL and unixODBC data sources, providing interaction both with relational and not relational databases. The main functionalities provided by this library are:

- open/close connection with database;
- submit query to database;
- open/commit/abort/rollback transaction;
- lock/unlock table;
- extract value from the recordset;

The MultiQuery library is built on top of the libgrelcSDAI and provides a set of APIs to develop an XML/SQL translator for the MultiQuery submission.

The MultiQuery is a GRelC proprietary kind of query/format used to update a data source (this is a single shot query) using a massive amount of data. The rationale behind this kind of query is that the user does not directly

transfer the update query (INSERT, UPDATE and DELETE), but just the data in (XML format) from which the XML/SQL translator is able to infer the query itself. The MultiQuery format contains logical links which will be resolved to physical links by the XML/SQL translator on the DBMS side. Due to the nature, the mechanism and the performance of this query, the Multiquery is strongly recommended to populate relational Information Services. Additional information related to a performance analysis in an European testbed of the MultiQuery can be found in [24]. To date, the two libraries, presented in this section, are extensively used in several grid projects such as GRelC, DGC [25], DEOSIS [26].

**Information Schema** One of the most important requirements for Grid computing scenarios is the ability to discover services and web services dynamically. The iGrid system provides Grid developers with the following functionalities: registration, unregistration, update and lookup. More than one instance for each Service or Web Service can be registered. The following information belongs to these categories:

- Service information.
  - GridLab-iGrid-Service-id: service identification number;
  - GridLab-iGrid-Service-name: service logical name;
  - GridLab-iGrid-Service-description: service description;
  - GridLab-iGrid-Service-keywords: set of key words of a service;
  - GridLab-iGrid-Service-defaultport: service default port.
- Service instance information.
  - GridLab-iGrid-Service-accessurl: service access URL;
  - GridLab-iGrid-Service-publisher: service publisher (X509v3 certificate distinguished name);
  - GridLab-iGrid-Service-creationdate: creation date of the service instance information;
  - GridLab-iGrid-Service-validitytime: validity time of the service instance information.
- Web Service information.
  - GridLab-iGrid-WebService-id: web service identification number;
  - GridLab-iGrid-WebService-name: web service logical name;
  - GridLab-iGrid-WebService-description: web service description;
  - GridLab-iGrid-WebService-keywords: set of key words of a web service;
  - GridLab-iGrid-WebService-wsdlloc: (multivalue) URL where the WSDL document for the Web Service can be found.
- Web Service instance information.
  - GridLab-iGrid-WebService-accessurl: web service access URL;
  - GridLab-iGrid-WebService-publisher: web service publisher (X509v3 certificate distinguished name);
  - GridLab-iGrid-WebService-creationdate: creation date of the web service instance information;
  - GridLab-iGrid-WebService-validitytime: validity time of the web service instance information.
- Firewall information.
  - GridLab-iGrid-Firewall-id: firewall identification number;
  - GridLab-iGrid-Firewall-hostname: firewall hostname;
  - GridLab-iGrid-Firewall-ports: (multivalue) attribute composed by the open ports (GridLab-iGrid-Firewall-port), the time frame during which each port (or a range of ports) is open (GridLab-iGrid-Firewall-firvaliditytime), the protocol (TCP/UDP) used to connect to firewall ports (GridLab-iGrid-Firewall-prot), the start time which each port (or a range of ports) is open (GridLab-iGrid-Firewall-fircreationdate) and a flag (GridLab-iGrid-Firewall-type\_por) to specify if the binding to the port(or range of ports) has allowed;

- GridLab-iGrid-Firewall-adminDN: distinguished name of the firewall administrator;
  - GridLab-iGrid-Firewall-publisher: firewall publisher (X509v3 certificate distinguished name);
  - GridLab-iGrid-Firewall-creationdate: creation date of the firewall information;
  - GridLab-iGrid-Firewall-validityTime: validity time of the firewall information.
- Host information.
    - GridLab-iGrid-Host-id: Host identification number;
    - GridLab-iGrid-Host-hostname: the FQDN of the Host;
    - GridLab-iGrid-Host-domainname: the domain name of the Host;
- Network information.
    - GridLab-iGrid-Net-name: Network name;
    - GridLab-iGrid-Net-address: Network address;
    - GridLab-iGrid-Net-mask: Network mask;
    - GridLab-iGrid-Net-creationdate: creation date of the Network information;
    - GridLab-iGrid-Net-validityTime: validity time of the Network information.
- Operative system information.
    - GridLab-iGrid-System-id: Operative System identification number;
    - GridLab-iGrid-System-name: Operative System name;
    - GridLab-iGrid-System-version: Operative System version;
    - GridLab-iGrid-System-release: Operative System release;
    - GridLab-iGrid-System-machine: host platform;
    - GridLab-iGrid-System-creationdate: creation date of the Operative System information;
    - GridLab-iGrid-System-validityTime: validity time of the Operative System information.
- Cpu static information.
    - GridLab-iGrid-Cpusta-id: Cpu identification number;
    - GridLab-iGrid-Cpusta-vendor: Cpu vendor;
    - GridLab-iGrid-Cpusta-version: Cpu version;
    - GridLab-iGrid-Cpusta-model: Cpu model;
    - GridLab-iGrid-Cpusta-cpumhz: Cpu Mhz;
    - GridLab-iGrid-Cpusta-cachesize: cache size;
    - GridLab-iGrid-Cpusta-features: Cpu features;
    - GridLab-iGrid-Cpusta-number: number of Cpu belonging to the host;
    - GridLab-iGrid-Cpusta-creationdate: creation date of the static Cpu information;
    - GridLab-iGrid-Cpusta-validityTime: validity time of the static Cpu information.

## 2.2 Mercury Monitoring System

In a complex system as the grid, monitoring is essential for understanding its operation, debugging, failure detection and for performance optimisation. To achieve this, data about the grid must be gathered and processed to reveal important information. Then, according to the results, the system may need to be controlled. The Mercury Grid Monitoring System provides a general and extensible grid monitoring infrastructure. Mercury Monitor is designed to satisfy specific requirements of grid performance monitoring [27]. It provides monitoring data represented as metrics via both pull and push model data access semantics and also supports steering by controls. It supports monitoring of grid entities such as resources and applications in a generic, extensible and scalable way. The architecture of Mercury Monitor extends the Grid Monitoring Architecture (GMA) [29] proposed by Global Grid Forum with actuators and controls. Mercury Monitor features a modular implementation with emphasis on simplicity, efficiency, portability and low intrusiveness on the monitored system.

The input of the monitoring system consists of measurements generated by sensors. Sensors are controlled by producers that can transfer measurements to consumers when requested, and are implemented as shared objects that are dynamically loaded into the producer at run-time depending on configuration and incoming requests for different measurements. It is also important to note that in Mercury measurements are performed only when requested by a consumer and data is only sent where it is needed. This ensures that intrusion on the monitored system is kept low. All measurable quantities are represented as metrics. Metrics are defined by a unique name such as *host.cpu.user* which identifies the metric definition, a list of formal parameters and a data type. By providing actual values for the formal parameters a metric instance can be created, representing a specific entity to be monitored. A measurement corresponding to a metric instance is called a metric value. Values contain a time-stamp and the measured data according to the data type of the metric definition. Sensor modules implement the measurement of one or more metrics. Mercury Monitor supports both event-like (i.e. an external event is needed to produce a metric value, e.g., a status change) and continuous metrics (i.e. a measurement is possible whenever a consumer requests it, e.g., the CPU temperature in a host). Continuous metrics can be made event-like by requesting automatic periodic measurements.

The GMA proposal of the Global Grid Forum only describes components required for monitoring. It is often necessary however, to also influence the monitored entity based on the analysis of measured data. For example, an application might need to be told to checkpoint and migrate if it does not perform as expected, a service may need to be restarted if it crashed or system parameters (such as process priorities or TCP buffer sizes) might need to be adjusted depending on current resource usage. To support this, actuators have been introduced in Mercury. Actuators are analogous to sensors in the GMA but instead of monitoring something they provide a way to influence the monitored entity. As sensors are accessed by consumers via producers, actuators are made available for consumers via actuator controllers. As the producer manages sensors (start, stop and control sensors and initiate measurements on a user's request) the actuator controller manages actuators. Similarly to metrics implemented by sensors, actuators implement controls that represent interactions with either the monitored entities or the monitoring system itself. The functional difference between metrics and controls is that metrics only provide data while controls do not provide data except for a status report but they influence the state or behaviour of the monitoring system or the monitored entity.

Besides providing information about grid resources, Mercury contains two elements to aid application and service monitoring and steering: an application sensor and an instrumentation library that communicates with this sensor. Together they allow to register application specific metrics and controls and to receive and serve requests for metrics and controls while the application is running. The application sensor is responsible for keeping track of processes of jobs as well as any private metrics or controls that the running applications provide. The application sensor forwards requests for application specific metrics or controls to the application process(es) they belong to. The request is then interpreted by the instrumentation library by performing the measurement or executing the requested control. The instrumentation library communicates with the application sensor using a UNIX domain socket, but it also has shared memory support to speed up transferring large volumes of data such as generated by fine-grained instrumentation and ensure that the intrusion on the monitored application is as low as possible. The application may also put certain variables into the shared memory area so they can be queried or modified without direct interaction with the application. This is useful for single-threaded applications or services that do not call the event handler of the instrumentation library for extended periods of time. Processing of application specific metric events is optimised inside the instrumentation library, i.e. they will not be sent to the application sensor if there are no consumers requesting them. This ensures that if an application is built with fine-grained instrumentation it can still run without noticeable performance degradation if the generated events are not requested. The instrumentation can be build into libraries (such as MPI implementations) to help application monitoring and the instrumentation library provided by Mercury also supports remote debugging and

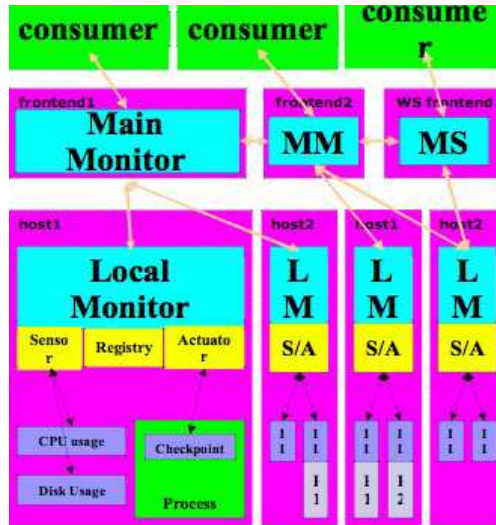


Figure 4: Mercury monitoring system architecture [28]

logging via appropriate sensor modules in the producer [5].

### 2.2.1 Host sensor modules

As an example of the metrics supported by Mercury sensor modules the information available about grid resources is listed below.

**Module cpuid:** processor type and cache information for ix86

- This module provides CPU type and cache information for ix86-compatible processors. Support for the cpuid instruction is required. Per-processor information on SMP systems requires the Linux cpuid kernel driver.
- List of supported metrics: host.cpu.l1dcache, host.cpu.l1icache, host.cpu.l2cache, host.cpu.l3cache, host.cpu.type

**Module darwin:** host sensor for Darwin

- This module provides information about machines using the Darwin (MacOS X) operating system.
- The following information are provided:
  - CPU number and type
  - Memory and swap size
  - Network interface statistics
  - Boot time
- List of supported metrics: host.cpu.number, host.cpu.frequency, host.cpu.l1dcache, host.cpu.l1icache, host.cpu.l2cache, host.cpu.l3cache, host.cpu.type, host.mem.size, host.net.recv.byte, host.net.recv.packet, host.net.recv.error, host.net.recv.drop, host.net.send.byte, host.net.send.packet, host.net.send.error, host.net.collision, host.net.total.byte, host.net.total.packet, host.net.total.error, host.os.boottime, host.swap.size

**Module gtop:** retrieve information through the libgtop2 library

- This module makes information provided by the libgtop2 monitoring library available in Mercury.
- The following information are provided (if the host operating system supports it):
  - Load average
  - CPU number, status and usage

- Memory size and usage
- Swap size and usage
- Network interface statistics
- List of supported metrics (not everything is available on every operating system): host.loadavg, host.cpu.number, host.cpu.online, host.cpu.clocktick, host.cpu.usage, host.cpu.user, host.cpu.nice, host.cpu.system, host.cpu.idle, host.mem.size, host.mem.free, host.swap.size, host.swap.free, host.swap.in, host.swap.out, host.net.recv.byte, host.net.recv.packet, host.net.recv.error, host.net.send.byte, host.net.send.packet, host.net.send.error, host.net.-collision, host.net.total.byte, host.net.total.packet, host.net.total.error

**Module irix:** host sensor for IRIX

- This module provides information about machines running IRIX 6.5.
- The following information are provided:
  - Load average
  - CPU number, status and usage
  - List of mounted filesystems
  - Network interface statistics
  - Operating system name and version
  - Swap size and usage
- List of supported metrics: host.loadavg, host.cpu.number, host.cpu.online, host.cpu.available, host.cpu.indexes, host.cpu.state, host.cpu.frequency, host.cpu.usage, host.cpu.user, host.cpu.nice, host.cpu.system, host.cpu.iowait, host.cpu.idle, host.cpu.irq, host.fs.mounted, host.mem.size, host.mem.free, host.mem.shared, host.mem.buffers, host.mem.cached, host.net.recv.byte, host.net.recv.packet, host.net.recv.error, host.net.recv.drop, host.net.send.-byte, host.net.send.packet, host.net.send.error, host.net.collision, host.net.total.byte, host.net.total.packet, host.-net.total.error, host.os.name, host.os.release, host.os.version, host.os.machine, host.swap.size, host.swap.free, host.swap.in, host.swap.out

**Module linux:** host sensor for Linux

- This module provides host information for Linux. The module takes most information from the /proc file system.
- The following information are provided:
  - Load average
  - Number of existing and running processes
  - CPU usage information (combined and per-cpu counters)
  - Known disks, partitions, and I/O utilization
  - List of mounted file systems
  - Memory and swap usage
  - Network statistics
- The module supports the following configuration parameters:
  - MinUpdateInterval (double). Minimum time between two updates of network statistics. When a query for a network metric arrives, all interface statistics are cached. If a new query arrives before the specified time elapses the cached information will be returned.
  - NetScanInterval (double). Interval between two automatic scans of network interface statistics. If a counter overflows more than once during this period due to high resource usage, the reported metrics become inaccurate.

- NetDataSource (string). Source of network statistics. The possible values are `procfs` to scan the `/proc` filesystem, or `netlink` to use the netlink protocol. The preferred source is `netlink` because it provides instant notification about changes but it may not be available on older kernels. If `NetDataSource` is set to `auto` (the default), the module will try to use `netlink` first, and fall back to `/proc` if `netlink` is not available.
  - IgnoreFS (string). File systems having this type will be ignored when reporting mounted file systems and file system statistics. This option may be specified multiple times.
- List of supported metrics: `host.loadavg`, `host.processes.all`, `host.processes.running`, `host.os.boottime`, `host.cpu.number`, `host.cpu.online`, `host.cpu.indexes`, `host.cpu.type`, `host.cpu.state`, `host.cpu.frequency`, `host.cpu.usage`, `host.cpu.user`, `host.cpu.nice`, `host.cpu.system`, `host.cpu.iowait`, `host.cpu.idle`, `host.cpu irq`, `host.cpu.softirq`, `host.disk.disks`, `host.disk.partitions`, `host.disk.iostat`, `host.fs.mounted`, `host.mem.size`, `host.mem.free`, `host.mem.buffers`, `host.mem.cached`, `host.swap.size`, `host.swap.free`, `host.swap.in`, `host.swap.out`, `host.vm.page.in`, `host.vm.page.out`, `host.vm.page.fault`, `host.vm.page.majorfault`, `host.net.recv.byte`, `host.net.recv.packet`, `host.net.recv.error`, `host.net.recv.drop`, `host.net.send.byte`, `host.net.send.packet`, `host.net.send.error`, `host.net.send.drop`, `host.net.collision`, `host.net.send.carrier`, `host.net.total.byte`, `host.net.total.packet`, `host.net.total.error`

**Module mach:** host sensor for operating systems based on the Mach microkernel

- This module provides information for Mach-based operating systems. The module has so far been tested on Tru64, Hitachi HI-UX and the GNU Hurd.
- The following information are provided (if the host operating system supports it):
  - CPU utilization
  - File system usage
  - Memory usage
  - Virtual memory statistics
- Access of some or all information may require root privileges.
- List of supported metrics (not everything is available on every operating system): `host.cpu.number`, `host.cpu.online`, `host.cpu.indexes`, `host.cpu.state`, `host.cpu.type`, `host.cpu.usage`, `host.cpu.user`, `host.cpu.nice`, `host.cpu.system`, `host.cpu.idle`, `host.cpu.iowait`, `host.mem.size`, `host.mem.free`, `host.vm.page.in`, `host.vm.page.out`, `host.vm.page.fault`

**Module osf:** host sensor for OSF/1-based systems

- This module provides information for machines running OSF/1-based operating systems (tested on Tru64 and HI-UX).
- The following information are provided (if the host operating system supports it):
  - Load average
  - Clock resolution
  - CPU usage
  - Disk utilization
  - Memory and swap size and usage
  - Virtual memory statistics
- List of supported metrics (not everything is available on every operating system): `host.loadavg`, `host.cpu.clocktick`, `host.cpu.idle`, `host.cpu.iowait`, `host.cpu.nice`, `host.cpu.system`, `host.cpu.usage`, `host.cpu.user`, `host.disk.disks`, `host.disk.iostat`, `host.mem.size`, `host.mem.free`, `host.os.boottime`, `host.swap.size`, `host.swap.free`, `host.vm.page.in`, `host.vm.page.out`, `host.vm.page.fault`

**Module solaris:** host sensor for Solaris

- This module provides information about machines running Solaris.
- The following information are provided (if the host operating system supports it):
  - Load average
  - CPU number, type, state and usage
  - Available disks and their utilization (requires root privileges)
  - List of mounted file systems
  - Memory and swap size and usage
  - Virtual memory statistics
  - Network interface information (IP addresses, hardware type, status etc.)
- The module supports the following configuration parameters:
  - MinUpdateInterval (double). Minimum time between two updates of network statistics. When a query for a network metric arrives, all interface statistics are cached. If a new query arrives before the specified time elapses the cached information will be returned.
- List of supported metrics: host.os.boottime, host.loadavg, host.cpu.number, host.cpu.available, host.cpu.online, host.cpu.frequency, host.cpu.indexes, host.cpu.type, host.cpu.state, host.cpu.state\_begin, host.cpu.user, host.-cpu.nice, host.cpu.system, host.cpu.iowait, host.cpu.idle, host.cpu.usage, host.disk.disks, host.disk.partitions, host.disk.iostat, host.fs.mounted, host.swap.size, host.swap.free, host.swap.in, host.swap.out, host.vm.page.-in, host.vm.page.out, host.vm.page.majorfault, host.mem.size, host.mem.free, host.net.recv.byte, host.net.recv.-packet, host.net.recv.error, host.net.send.byte, host.net.send.packet, host.net.send.error, host.net.send.carrier, host.-net.collision, host.net.total.byte, host.net.total.packet, host.net.total.error

**Module tru64:** host sensor for Tru64 UNIX

- This module provides information for machines running Tru64 UNIX.
- The following information are provided (if the host operating system supports it):
  - CPU number, type, state and usage
  - Memory size
  - Network interface statistics
- List of supported metrics: host.cpu.available, host.cpu.frequency, host.cpu.indexes, host.cpu.l2cache, host.cpu.-number, host.cpu.online, host.cpu.state, host.cpu.type, host.cpu.idle, host.cpu.iowait, host.cpu.nice, host.cpu.-system, host.cpu.usage, host.cpu.user, host.mem.size, host.net.recv.byte, host.net.recv.packet, host.net.recv.-error, host.net.send.byte, host.net.send.packet, host.net.send.error, host.net.send.carrier, host.net.collision, host.-net.total.byte, host.net.total.packet, host.net.total.error

**Module unix:** host sensor for various UNIX-like systems

- This module provides information that is more or less common to several different UNIX-like systems.
- The following information are provided (if the host operating system supports it):
  - Clock resolution
  - File system usage
  - Load average
  - Network interface information (IP addresses, hardware type, status etc.)
  - Operating system name and version
- List of supported metrics (not everything is available on every operating system): host.cpu.clocktick, host.fs.-space, host.fs.inodes, host.fs.mounted, host.loadavg, host.net.interfaces, host.net.mtu, host.net.ifflags, host.net.-ifindex, host.net.addr.ipv4, host.net.addr.ipv6, host.net.addr.hwaddr, host.net.hwtype, host.os.boottime, host.os.-machine, host.os.name, host.os.release, host.os.version, host.users

### 3 Information Cache Mediator Component

The envisioned application-level information cache component [1] is supposed to provide a unified interface to deliver all kinds of meta-data (e.g., from a GIS like iGrid, a monitoring system like Mercury, or from application-level meta data) to a grid application. The cache's purpose is twofold. First, it is supposed to provide a unifying component interface to all data (independent of its actual storage), including mechanisms for service and information discovery. Second, this application-level cache is supposed to deliver the information really fast, cutting down access times of current Web-service based implementations like Globus GIS (up to multiple seconds) to the order of a method invocation. For the latter purpose, this component may have to prefetch (poll) information from the various sources to provide them to the application on time.

Such an application-cache component is currently being developed as a collaboration among the authors. Its API as presented to the application is inspired by GridLab's GAT [30]. The GAT specifies an API for monitoring purposes. The basis of this API is general and extensible enough to fit the current monitoring and information systems and possibly future ones too. Like Mercury and GAT, the mediator defines measurable quantities as metrics. A metric is a container which holds the unique name of the metric and the properties needed to retrieve the information, like parameters. There are two types of metrics, continuous metrics, which are available at all times and need to be polled, and discrete metrics, which become available after a certain event and therefore are pushed. The result of a measurement is stored in a container called a metric value.

An application that requests information creates a metric which specifies the unique name of the metric, the possibly required parameters for the metric, and a recommended frequency which indicates how often the mediator should update the corresponding metric value in its cache. Note that the frequency could be omitted if the underlying monitoring or information system is able to push the requested information.

From the moment the mediator receives the first metric value, the application will be able to get it from the cache without any delays. However, it could also choose to get notified if a value gets updated, this way both pull and push mechanisms are available to the application. Another option for applications is to request a metric value bypassing the cache. In this case, the mediator component will merely serve as a uniform interface between the underlying monitoring and information systems and the application.

In order to serve information from different sources, the mediator component uses an extensible 'plug-in' system, where each plug-in (metric provider) forms the link between the mediator component and the underlying monitoring or information system. The metric providers have to deal with the actual retrieval of information and present it to the (rest of the) mediator component. The mediator component will process the information further, possibly by caching it.

A metric provider instance will represent one running information system on a host. So multiple metric provider instances retrieving information from the same type of information system can exist next to each other but retrieving information from different hosts. Some types of information may be retrieved from only one information system, while other types of information could be obtained from multiple ones. This latter type of information could be presented by the different systems in different ways, using different data types or even different measurement units. It is up to the metric providers to translate information presented by the underlying system into a format which the mediator component presents to the application. If information is requested by an application, the application can either choose a certain metric provider, or leave it up to the mediator to decide which metric provider will be used to retrieve the information. If multiple metric providers are able to retrieve the same type of information, it is a matter of policy which one is chosen (for example the source with the lowest response time or the most reliable source).

As depicted in Fig. 5 a request for a metric value comes in at the main object, the GIC object (1). The request will then be forwarded to the MetricProviderSelector object (2). This object implements the mechanisms to choose a metric provider according to a certain policy. Currently, two mechanisms are implemented. Firstly, the MetricProviderSelector object can choose the first metric provider that successfully retrieves the value. Secondly, an ordering of metric providers can be specified, the first provider in this list that successfully retrieves the value will be assigned to the metric. The basic idea of the metric provider selector, is the fact that it can only decide whether a metric provider can retrieve a value, by actually retrieving the value. Due to this approach, the MetricProviderSelector object will not only set the metric provider to use, but it will also return the retrieved metric value. In order to decide which metric provider is able to retrieve the value, the metric provider selector will request a list of metric providers that are able to retrieve the specified metric, ignoring the given parameters (3). This way, some of the the metric providers may be already ruled out. This list will be forwarded together with the request to the MetricUpdaterManager object (4).

Some metric providers may allow multiple threads to use when updating retrieving a metric, while others may

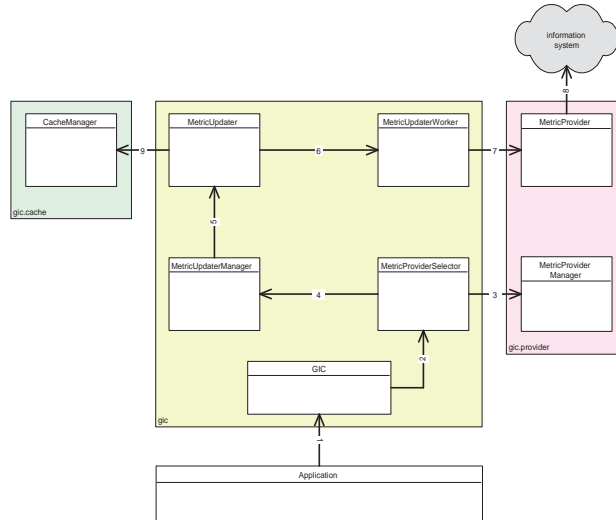


Figure 5: Metric flow and object interactions

allow only one. The metric updater manager takes care of controlling the number of simultaneous metric retrievals. This number of multiple threads can be set per metric provider. The metric provider manager maintains a pool of MetricUpdater objects, for each MetricProvider object in the system it holds one MetricUpdater object. Each metric updater holds a list of metrics it keeps updated according to the frequency specified in the metric. If this frequency is set to zero, it will retrieve the metric only once. Objects can install a listener to get notified of the results of the metric requests the MetricUpdater receives. The metric updater manager will choose the correct metric updater object to use for retrieving the metric (5).

The metric updaters will create a new thread running in a MetricUpdaterWorker (6) object which will actually make the request at the metric provider (7). The MetricUpdater object will make sure that not more MetricUpdaterWorker threads will be started simultaneously than the assigned metric provider allows. Within the mediator component, the MetricUpdaterWorker objects will be the only objects that make the actual calls for metric requests at the metric providers. The metric provider will then retrieve the value from some external source (8).

By default, each metric updater will notify the cache manager of each metric it processes (9), this way every metric is propagated to the cache manager. The cache manager will decide whether or not the value should be cached. This decision is based on the validity time which is assigned to each metric. This validity time of a metric can be specified by the application that retrieves the metric. If no validity time is assigned, a default will be used which is specified by the metric provider that retrieves the value. This way, the iGrid metric provider can make use of the validity time property stored in the iGrid database and the Mercury metric provider could, for example, indicate a validity time of 0 for real-time metrics as a default value.

### 3.1 Metric provider manager

The metric providers are loaded by the metric provider manager, this component locates the providers and registers the MetricProviderDefinitions to the mediator. If a metric provider instance is created, it needs to be registered at the metric provider manager. The manager will then call the `getSupportedMetricDefinitions()` method of the metric provider. For each metric definition, it will first check if it is already registered by another provider, and if so, whether it conforms to this existing definition. If more than one metric provider is able to retrieve the same metric, the metric providers must agree on the definition of this metric (parameters, return values). If registration is successful, the manager will add the definition to the database of metric definitions. This database is implemented using Berkeley DB.

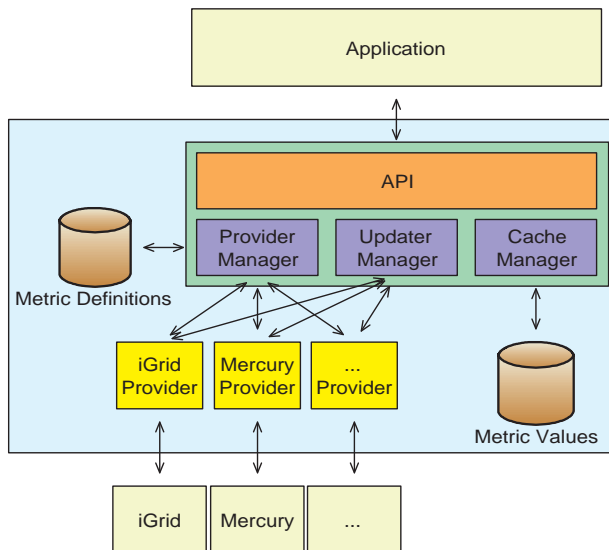


Figure 6: Information cache mediator component architecture

### 3.2 Metric updater manager

The metric updater manager takes care of forwarding requests to the metric providers. A request can be either to retrieve a value by using the `getMetricValue()` method, to show interest in a discrete metric by invoking `subscribeMetric` or a request to stop retrieving values of a discrete metric by invoking the `unsubscribeMetric` method. These requests will be handled by the metric updater manager. This object will take care of forwarding the request to the correct metric provider. Also, it controls the maximum number of concurrent threads of each metric provider, specified by the `getMaxThreadCount()` method. After a request has completed, successfully or not, the manager will notify other parts of the mediator by invoking a method defined in a listener interface. Apart from simply forwarding these request, the metric updater manager will also take care of continuously updating the metric values, if necessary. Assigned to each metric is a frequency which specifies the update interval the application specifies. According to this frequency, the metric will be requested from the assigned provider.

### 3.3 Metric provider selector

A metric can have a specific metric provider assigned which should retrieve the value. However, if none is assigned, the system will try to select the provider to use. The metric provider selector object will take care of choosing the provider. It implements the mechanisms to choose a metric provider according to a certain policy. The main idea behind the selector object is the fact that it can only decide whether a provider is able to retrieve a value by actually requesting it. So, it will request the metric from various providers and, on the basis of the replies, it will assign one of them to the metric. Currently, two mechanisms are implemented to decide which provider to use. The selector can choose either the first provider that successfully retrieves the value or an ordering of providers can be specified. In the latter case, the first provider in the list that was able to retrieve the value will be assigned to the metric.

### 3.4 Cache manager

As described earlier, notifications from the metric updater manager will be sent to other parts of the mediator whenever a new metric is retrieved. One of these parts is the cache manager which takes care of storing the metric values in cache, if necessary. The cache manager will decide whether or not the value should be cached. This decision is based on the validity time which is assigned to each metric. This validity time serves two purposes. First, when a metric is retrieved, and the cache manager gets notified to store the received value, it will check the validity time to see whether it should cache the value or not. A validity time of 0 indicates the metric should not be cached. Second, if the validity time is higher than 0, the value will be cached. The validity time is stored along with the metric value. When the value

is retrieved from the cache, the cache manager will check the validity time to see whether it should return the value or not. If the value is not valid according to the validity time, an exception will be thrown. The validity time of a metric can be specified by the application that retrieves the metric. If no validity time is assigned, a default will be used which is specified by the metric provider that retrieves the value. This way, the iGrid metric provider can make use of the validity time property stored in the iGrid database and the Mercury metric provider could, for example, indicate a validity time of 0 for real-time metrics as a default value.

### 3.5 API

Applications using the mediator component will usually only interact with the GIC object. This object contains all of the methods to retrieve the metric values. The GIC object will forward all of the requests to the correct objects in the system. It contains these methods:

**static GIC getInstance()**

Returns a reference to the GIC instance.

**List getMetricDefinitions()**

Returns a list of all of the available metric definitions.

**MetricDefinition getMetricDefinitionByName(String name)**

Returns a metric definition given the metric name.

**MetricValue getMetricValue(Metric metric)**

Retrieves a metric value from the cache, if available. If it is not stored in the cache, or it is invalidated, an exception will be thrown.

**MetricValue getMetricValueFromProvider(Metric metric)**

Retrieves a metric value directly from a metric provider, bypassing the cache. If the metric has no provider assigned, the system will choose one. If the metric has a frequency specified, the metric will be updated according to this frequency. An exception will be thrown if retrieval fails.

**void stopUpdating(Metric metric)**

Indicates that the metric described by the parameter metric does not need to be updated anymore.

**void addMetricListener(MetricListener listener, Metric metric)**

Adds a metric listener for retrieving updates about a certain metric. This can either be updates from a continuous metric which is updated according to its frequency, or updates from a discrete metric.

**void removeMetricListener(MetricListener listener, Metric metric)**

Removes a metric listener.

**void subscribeMetric(Metric metric)**

Subscribe a metric for updating. This should be used for discrete metrics.

**void unsubscribeMetric(Metric metric)**

Unsubscribe a previous subscribed metric to indicate the application is not interested in this discrete metric anymore.

### 3.6 Using the mediator component

Currently, the mediator component will create metric provider instances according to a configuration file. This way, the application does not need to create metric provider instances. Actually, the application does not even need to know what metric provider instances are running, nor which underlying systems are retrieving the values. It only needs to know about the metrics it wishes to retrieve. The following code shows this.

```

GIC gicInstance;
MetricDefinition metricDefinition;
Metric metric;
Map parameters;
MetricValue metricValue;

try {
    gicInstance = GIC.getInstance();
    metricDefinition = gicInstance.getMetricDefinitionByName("mem_ram_free");
    parameters = new HashMap();
    parameters.put("host", "n0.hpcc.sztaki.hu");
    metric = metricDefinition.createMetric(parameters, 30000);
    metricValue = gicInstance.getMetricValueFromProvider(metric);
    System.out.println("Free RAM: " + metricValue.getValue().get("mem_ram_free"));
}
catch (GICException e) {
    System.out.println("An error occurred.");
}

```

The code above will retrieve the metric definition for the metric with the unique name *mem\_ram\_free*. From this definition a metric is created with one parameter assigned, namely the parameter *host*. Also, the code indicates it wishes the value to be updated each 30 seconds. The actual request is made by invoking the `getMetricValueFromProvider()` method. The application does not specify any metric provider to use for retrieving this value. Therefore, the system will choose the metric provider according to the policy specified. If more than one provider is able to retrieve the value, it can either choose the first one that replies, or the one with the highest priority assigned. If the value is retrieved, the entry *mem\_ram\_free* from the map of values is printed out on the screen. Because the application indicated it likes the value to be updated in the cache in a 30 seconds interval, it can now use the `getMetricValue()` to retrieve the value from the cache.

```

try {
    metricValue = gicInstance.getMetricValue(metric);
    System.out.println("Free RAM: " + metricValue.getValue().get("mem_ram_free") +
        " created: " + new Date(metricValue.getCreationTime()));
}
catch (GICException e) {
    System.out.println("An error occurred."); }

```

Apart from pulling the value from the cache, the application can also install a listener to get notified of updates.

```

public void processMetricEvent(MetricValue metricValue) {
    System.out.println("Free RAM: " + metricValue.getValue().get("mem_ram_free") +
        " created: " + new Date(metricValue.getCreationTime())); }

try {
    gicInstance.addMetricListener(this, metric);
} catch (GICException e) {
    System.out.println("An error occurred."); }

```

In case of a discrete metric, the application can use the `subscribeMetric()` method to let the system retrieve events according to this metric. The application can process the values resulting from this request in the same ways as described above. It can pull the latest value from the cache, or it can install a listener to get notified immediately.

## 4 Conclusions and consideration

The information cache mediator component described in this report aims at two goals: the main one is establishing an immediate and faster way for applications to access a wide variety of information distributed among all of the grid

resources made available to the application itself; the second one is the creation of a uniform interfaces with respect to the heterogeneity of the grid information services currently deployed in existing grid infrastructures. By design the information cache mediator component can easily be used and embedded within *grid-aware* applications through the API provided. However, this mediator component can be also considered as belonging to the emerging Grid Component Model because of its design and features. In particular we can identify two parts of the mediator component: a *controller* and a *content*. The *content* is represented mainly by the Cache Manager module that will collect the information according to the metrics defined and specified by the application; the *controller* is represented mainly by the Metric Provider Manager module that is in charge of handling the information retrieval from the underlying and registered information services.

Moreover, the information cache mediator component exposes two types of *interfaces*: *server* and *client*. The mediator component will receive operation invocations whenever an application wants to access the needed metric values from the cache mediator component so that these specific interfaces can be considered as *server interfaces*; on the other hand the mediator component will emits operation invocations whenever it requires accessing underlying information providers to collect the requested information: these invocation interfaces can be classified as *client interfaces*. Finally, at the moment the information cache mediator component can collect information from iGrid and Mercury, but the designed architecture allows the integration of all types of information service because of its modularity and malleability based on plug-in mechanism.

## References

- [1] COREGRID VIRTUAL INSTITUTE ON PROBLEM SOLVING ENVIRONMENTS, TOOLS, AND GRID SYSTEMS *Proposal for mediator component toolkit*, CoreGRID deliverable D.ETS.02 (2005)
- [2] G. ALOISIO, M. CAFARO, I. EPICOCO, D. LEZZI, M. MIRTO, S. MOCVERO, *The Design and Implementation of the GridLab Information Service*, in Second International Workshop on Grid and Cooperative Computing (GCC 2003). Lecture Notes in Computer Science, Springer, 2003.
- [3] G. ALOISIO, M. CAFARO, I. EPICOCO, S. FIORE, D. LEZZI, M. MIRTO, S. MOCVERO, *Resource and service discovery in the igrd information service*, In Proceedings of International Conference on Computational Science and its Applications (ICCSA 2005). Volume 3482., Springer-Verlag (2005) 19
- [4] Z. BALATON, G. GOMBAS, *A flexible multi-level grid monitoring architecture*, In Proceedings of the First European Across Grids Conference. (2003)
- [5] G. GOMBAS, A. CS. MAROSI, Z. BALATON, *Grid application monitoring and debugging using the mercury monitoring system*, In Proceedings of the EGC 2005, volume 3470 of Lecture Notes in Computer Science. (2005)
- [6] G. ALOISIO, M. CAFARO, I. EPICOCO, S. FIORE, D. LEZZI, M. MIRTO, S. MOCVERO, *iGrid, a Novel Grid Information Service*, to appear in proceedings of the first European Grid Conference (EGC 2005).
- [7] S. RATNASAMY, P. FRANCIS, M. HANDLEY, R. KARP, S. SHENKER, *A scalable content-addressable network*, in Proceedings of ACM SIGCOMM, San Diego, CA (August 2001).
- [8] I. STOICA, ET AL, *Chord: A scalable peer-to-peer lookup service for Internet applications*, in Proceedings of ACM SIGCOMM, San Diego (August 2001); <http://www.pdos.lcs.mit.edu/chord>.
- [9] P. MAYMOUNKOV, D. MAZIERES, *Kademlia: A peer-to-peer information system based on the XOR metric*, in Proceedings of the 1st International Workshop on Peer-to-Peer Systems, Springer-Verlag version, Cambridge, MA (Mar. 2002); <http://kademia.scs.cs.nyu.edu/>
- [10] D. MALKHI, M. NAOR, D. RATAJCZAK, *Viceroy: A scalable and dynamic emulation of the butterfly*, in Proceedings of ACM Principles of Distributed Computing(PODC) Monterey, CA (July 2002)
- [11] A. ROWSTRON, P. DRUSCHEL, *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*, in Proceedings of the 18th IFIP/ACM Intl Conference on Distributed Systems Platforms (Nov. 2001); <http://www.cs.rice.edu/CS/Systems?Pastry>.

- [12] K. HILDRUM, J. KUBIATOWICZ, S. RAO, B. ZHAO, *Distributed Object Location in a Dynamic Network*, in Proceedings of 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA), August 2002.
- [13] P. A. ALSBERG, J. D. DAY, *A Principle for Resilient Sharing of Distributed Resources*, in Proceedings of the Second International Conference on Software Engineering, San Francisco, CA, 1976, 562-570.
- [14] J. F. BARTLETT, *A Nontop Kernel*, in Proceedings of the Eighth Symposium on Operating Systems Principles, in ACM Operating System Review, 1981.
- [15] A. BHIDE, E. N. ELNOZAHY, S. P. MORGAN, *A highly available network file server*, in Proceedings of the USENIX, 1991, 199-205.
- [16] N. BUDHIRAJA, K. MARZULLO, F. B. SCHNEIDER, S. TOUEG, *Optimal primary-backups protocols*, in Proceedings of the Sixth International Workshop on Distributed Algorithms, Haifa, Israel, 1992, 362-378.
- [17] Foster, I., Kesselman C.: GLOBUS: a Metacomputing Infrastructure Toolkit, Int. J. Supercomputing Applications, 1997, pp. 115–28
- [18] K. CZAJKOWSKI, S. FITZGERALD, I. FOSTER, C. KESSELMAN, *Grid Information Services for Distributed Resource Sharing*, in Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
- [19] I. FOSTER, C. KESSELMANN, J. NICK, S. TUECKE, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed System Integration*, Technical Report for the Globus project. <http://www.globus.org/research/papers/ogsa.pdf>
- [20] The WSRF specification. <http://www.globus.org/wsrf/specs/ws-wsrf.pdf>
- [21] I. FOSTER, C. KESSELMANN, J. NICK, S. TUECKE, *Grid Services for Distributed System Integration* in Computer, Vol. 35, 2002, No. 6, pp. 37–46
- [22] G. ALOISIO, M. CAFARO, I. EPICOCO, D. LEZZI, M. MIRTO, S. MOCAVERO, *The Design and Implementation of the GridLab Information Service*, in Proceedings of The Second International Workshop on Grid and Cooperative Computing (GCC 2003), 7-10 December 2003, Shanghai (China), Lecture Notes in Computer Science, Springer-Verlag, N. 3032, pp. 131-138, 2004
- [23] G. ALOISIO, M. CAFARO, S. FIORE, M. MIRTO, *The GRelC Project: Towards GRID-DBMS*, in Proceedings of Parallel and Distributed Computing and Networks (PDCN) - IASTED, February 17 to 19, 2004, Innsbruck, Austria
- [24] G. ALOISIO, M. CAFARO, S. FIORE, M. MIRTO, *Early Experiences with the GRelC Library*, Journal of Digital Information Management, Digital Information Research Foundation (DIRF) Press. Vol. 2, No. 2, June 2004, pp 54-60
- [25] G. ALOISIO, E. BLASI, M. CAFARO, I. EPICOCO, S. FIORE, D. LEZZI, M. MIRTO, *Dynamic Grid Catalog Information Service*, in Proceedings of the First European Across Grids Conference, February 13-14, 2003 Santiago de Compostela (Spain), Lecture Notes in Computer Science, Springer-Verlag, N. 2970, pp. 198-205, 2003.
- [26] G. ALOISIO, M. CAFARO, S. FIORE, G. QUARTA, *A Grid-Based Architecture for Earth Observation Data Access*, to be published in The 20th Annual ACM Symposium on Applied Computing, Santa Fe, New Mexico, March 13 -17, 2005
- [27] Z. NEMETH, G. GOMBAS, Z. BALATON, *Performance evaluation on grids: Directions, issues, and open problems*, in Proceedings of the Euromicro PDP 2004, A Coruna, Spain, IEEE Computer Society Press (2004)
- [28] G. ALOISIO, Z. BALATON, P. BOON, M. CAFARO, I. EPICOCO, G. GOMBÁS, P. KACSUK, T. KIELMANN, D. LEZZI, *Integrating Resource and Service Discovery in the CoreGrid Information Cache Mediator Component*, in Proceedings of the CoreGRID workshop "Integrated research in Grid Computing" S. Gorbach and M. Danelutto (Eds.), Nov 28-30, 2005 Pisa, pp 437-446

- [29] S. FISHER ET AL, *R-gma: A relational grid information and monitoring system*, in Proceedings of 2nd Cracow Grid Workshop, Cracow, Poland (2003)
- [30] G. ALLEN, K. DAVIS, T. GOODALE, A. HUTANU, H. KAISER, T. KIELMANN, A. MERZKY, R. VAN NIEUW-POORT, A. REINEFELD, F. SCHINTKE, T. SCHUTT, E. SEIDEL, B. ULLMER, *The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid*, in Proceedings of the IEEE 93(8) (2005) 534550