

IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

## D11.4 EXTENDED ARCHITECTURE SPECIFICATION

---

|                          |                              |
|--------------------------|------------------------------|
| Author(s):               | Zoltán Balaton, Gábor Gombás |
| Document Filename:       | GridLab-11-D11.4-06-ExtArch  |
| Work package:            | WP11                         |
| Partner(s):              | GridWare, MU, SZTAKI, VU     |
| Lead Partner:            | SZTAKI                       |
| Config ID:               | GridLab-11-D11.4-06-v1.0     |
| Document classification: | IST                          |

---

**Abstract:** This document presents the extended architecture of the monitoring system. The document describes the extended features supported by the monitoring system, such as actuators, guaranteed data delivery, as well as security.

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>                     | <b>3</b> |
| <b>2</b> | <b>Actuators</b>                        | <b>3</b> |
| 2.1      | Control Definition . . . . .            | 4        |
| 2.2      | Control Instance . . . . .              | 4        |
| 2.3      | Built in metrics and controls . . . . . | 4        |
| <b>3</b> | <b>Guaranteed data delivery</b>         | <b>4</b> |
| 3.1      | Producer–consumer interaction . . . . . | 5        |
| 3.2      | Sensor–producer interaction . . . . .   | 5        |
| <b>4</b> | <b>Producer architecture</b>            | <b>6</b> |
| 4.1      | Main objects . . . . .                  | 6        |
| 4.1.1    | Connection . . . . .                    | 6        |
| 4.1.2    | Message . . . . .                       | 7        |
| 4.1.3    | Protocol Encoding . . . . .             | 7        |
| 4.1.4    | User . . . . .                          | 7        |
| 4.1.5    | Sensor . . . . .                        | 7        |
| 4.1.6    | Metric Instance . . . . .               | 7        |
| 4.1.7    | Metric ID . . . . .                     | 8        |
| 4.1.8    | Metric Value . . . . .                  | 8        |
| 4.1.9    | Actuator . . . . .                      | 8        |
| 4.1.10   | Control Instance . . . . .              | 8        |
| <b>5</b> | <b>Monitoring protocol</b>              | <b>8</b> |
| 5.1      | Protocol changes . . . . .              | 8        |
| 5.2      | Protocol encoding . . . . .             | 9        |
| 5.2.1    | Argument lists . . . . .                | 9        |
| 5.2.2    | Commands sent by the consumer . . . . . | 9        |
| 5.2.3    | Messages sent by the producer . . . . . | 10       |
| 5.2.4    | Command status responses . . . . .      | 10       |
| 5.2.5    | Producer capabilities . . . . .         | 11       |
| 5.2.6    | Authentication responses . . . . .      | 11       |
| 5.2.7    | Metric values . . . . .                 | 11       |
| 5.3      | Commands . . . . .                      | 12       |
| 5.3.1    | AUTH . . . . .                          | 12       |
| 5.3.2    | COLLECT . . . . .                       | 12       |
| 5.3.3    | STOP . . . . .                          | 12       |
| 5.3.4    | SUBSCRIBE . . . . .                     | 13       |
| 5.3.5    | BUFFER . . . . .                        | 13       |
| 5.3.6    | GET . . . . .                           | 13       |
| 5.3.7    | DEF_CHANNEL . . . . .                   | 14       |
| 5.3.8    | SET_CHANNEL . . . . .                   | 14       |
| 5.3.9    | DEL_CHANNEL . . . . .                   | 14       |
| 5.3.10   | COMMIT . . . . .                        | 14       |
| 5.3.11   | EXECUTE . . . . .                       | 15       |
| 5.3.12   | QUERY . . . . .                         | 15       |
| 5.3.13   | WRAP . . . . .                          | 15       |

|  |           |
|--|-----------|
| <b>6 Security</b>  | <b>16</b> |
| 6.1 Interaction with the Authorisation Service . . . . . | 17        |
| 6.2 Access Control . . . . .                             | 18        |
| 6.3 Resource Limits . . . . .                            | 19        |
| <b>References</b>  | <b>19</b> |

## 1 Introduction

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. It is the task of the monitoring system to provide facilities for this. In the Detailed Architecture Specification Report [2] a monitoring architecture based on the general Grid Monitoring Architecture [7] of the Global Grid Forum was presented that provides a flexible and extensible infrastructure for grid monitoring. In this document we present the extended features supported by the monitoring system, such as actuators, guaranteed data delivery and security.

## 2 Actuators

The GMA proposal of the Global Grid Forum [7] 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 in the monitoring system, in addition to the features described in the GMA document, *actuators* (similar to actuators in Autopilot [5] or manipulation services in OMIS [4]) are introduced.

Actuators are analogous to sensors in the GMA but instead of monitoring something they provide a way to control the monitored entity. As sensors are accessed by consumers via producers, actuators are made available for consumers via actuator controllers as shown in Figure 1. As the producer manages sensors (start, stop and control sensors and initiate measurements on a user's request) the actuator controller manages actuators.

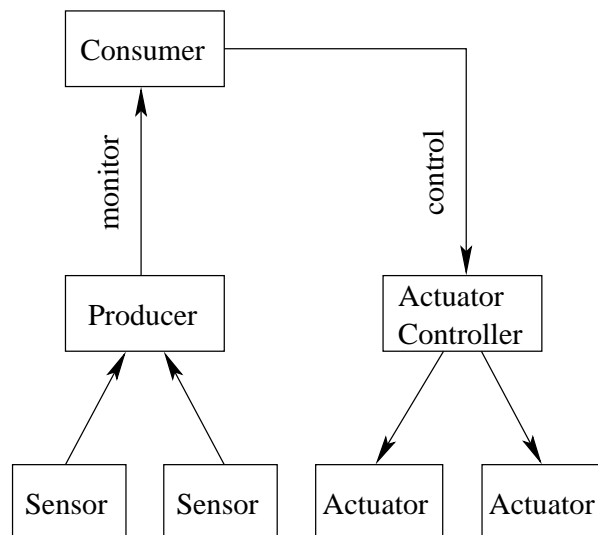


Figure 1: Monitoring System Architecture with Actuators

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.

## 2.1 Control Definition

A control is defined by the following properties:

- Control name
- Parameters
- Data type

The **Control name** is used to identify the control definition (e.g. job checkpoint request). It consists of dot separated words (e.g. `process.setpriority`).

The **Parameters** field in the control definition contains the formal definition of the control parameters. Just like the same metric is available at different places, controls can also act on different entities. For example, multiple jobs can run on a resource at the same time, each of which could be told to checkpoint. The control parameters can be used to distinguish between these different control instances.

The **Data type** is the definition of structure used for representing data returned by a control invocation. The types supported are the same as for metrics.

## 2.2 Control Instance

Substituting concrete values in the formal parameters of a control definition yields a control instance in the same way as substituting values in the parameters of a metric definition yields a metric instance. In contrast to metric instances that represent entities to be monitored, control instances represent entities to be influenced.

Unlike metric instances that can generate multiple metric values, control instances are always single-shot, i.e., every invocation of a control produces exactly one result. After execution control instances are destroyed automatically.

## 2.3 Built in metrics and controls

The use of metrics and controls makes the monitoring system very general and flexible. Producers are able to handle diverse sensors and actuators without detailed knowledge of their inner workings. At the same time, sensors and actuators are not restricted and free to exploit all the implementation possibilities. The flexibility provided by metrics and controls is also exploited within the monitoring system. There are some metrics built in the producer that provide information about the monitoring system. For example, producer capabilities and metric definitions are such metrics. Influencing the monitoring system is also possible via controls that act on the monitoring system itself. This lets consumers interact with the monitoring system and the monitored entity in the same way.

## 3 Guaranteed data delivery

Generally if either the producer or the consumer terminates, queued but not yet processed data will be lost. There are cases however (e.g. critical event notifications) when this behaviour is not acceptable. For these cases the monitoring system supports *guaranteed delivery* of measurements. Consumers can enable guaranteed delivery for a metric instance by executing a control. After guaranteed delivery is enabled, the producer awaits the acknowledgement of receiving each measurement from the consumer before deleting the metric value from its buffer. Sensors may also ask for guarantee notifications to ascertain that data they sent has safely arrived to the consumer.

Handling of guaranteed data delivery is expensive as it consumes resources of the producer. Therefore, the producer might impose resource limits of the number of transactions being active at any given time.

Guaranteed data delivery has two important roles:

- For the producer: avoid losing important data. Some information like job status change notifications must not be lost even if the communication between the producer and the consumers fails (because of a network failure, consumer crash, etc.). Such information are usually stored locally in a persistent database before being transferred to the consumers and are deleted after the consumers have processed them.
- For the consumer: for fail-safe operation the consumer may use persistent channels that are not destroyed when the network connection goes down or the consumer fails. However, when re-establishing a persistent channel the producer has to know which data did the consumer successfully process and what needs to be retransmitted.

In order to provide the above functionality, guaranteed data delivery involves two levels of operations: producer–consumer interaction and sensor–producer interaction.

### 3.1 Producer–consumer interaction

A consumer may request guaranteed delivery for a metric identifier by executing an appropriate control. This will result in all metric values belonging to that metric identifier being tagged with a transaction identifier. After receiving and processing the data the consumer should issue a COMMIT command (see section 5.3.10) with this transaction identifier to notify the monitoring system that the data has safely arrived and therefore can be removed from the buffer of the producer.

### 3.2 Sensor–producer interaction

Sometimes the sensor itself has to know when the data it sent has safely arrived to the consumer. To avoid losing important data they are often archived in a local persistent database before trying to send them to the producer. Such a database can be implemented by either the sensor responsible for the data or some external entity which the sensor represents. Thus, the sensor needs to know if the data were successfully processed and can be removed from the database. Another example is a sensor in the main monitor (MM) that is itself a consumer and gets metrics from sensors in an LM thus, it has to forward the acknowledgement to the lower level.

Sensors may ask for guarantee notifications for metric values to ascertain that they have been successfully sent to the consumer. If a consumer has requested guaranteed data delivery for a metric identifier this metric value belongs to, sending the metric value is considered successful if an appropriate COMMIT command has been received. If a consumer has not requested guaranteed data delivery for a metric identifier this metric value belongs to, sending the metric value is considered to be successful if the producer has sent it to the network connection (however, it can still be in an operating system buffer).

The sensor may ask for two kinds of notifications from the producer:

- Weak guarantee notification: the producer will send a success notification to the sensor when a metric value has been successfully sent on all channels that are associated with metric identifiers for which guaranteed delivery was requested. This ensures that all consumers that requested guaranteed delivery for any metric identifier belonging to the metric instance which generated the metric value have received it.
- Strong guarantee notification: the producer will send a success notification to the sensor when a metric value has been successfully sent on all channels that are associated with any metric identifier belonging to the metric instance which generated the metric value independent of the consumers requested guaranteed delivery or not. This ensures that the metric value is received by those consumers that requested guaranteed delivery and also has left the producer on all other channels.

## 4 Producer architecture

In the monitoring system the local monitor (LM) provides both producer and actuator controller functionality. Both sensors and actuators are loadable modules that are dynamically linked into the LM depending on configuration. This modularity makes the monitoring system flexible and makes it easy to add new sensors and actuators. The main monitor (MM) and the Monitoring Service (MS) uses the same modular architecture that is referred to as producer architecture in this section.

The producer architecture can be divided into two parts: some complex data types and the operations acting on them, and several surrounding functions like authentication, authorisation or implementation of the protocol commands. We will refer to the most complex data types as objects although they do not fully exploit the OO paradigm (for example, there is no inheritance between them).

### 4.1 Main objects

The main objects and their relations are shown in Figure 2.

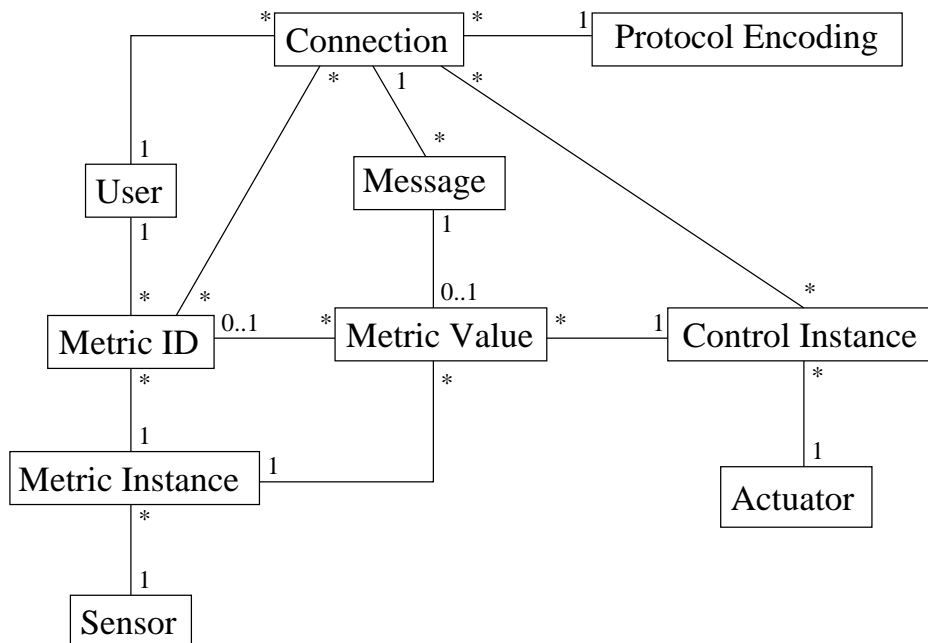


Figure 2: Main Objects in the Monitoring System

#### 4.1.1 Connection

The Connection is one of the central objects as it represents a channel between the producer and the consumer. In case of persistent channels, it is possible to have a Connection object without an actual network connection. For non-persistent channels if the corresponding network connection is lost the Connection object is destroyed.

The Connection object is responsible for doing all the I/O operations including any transport level transformations (compression, encryption, etc.) that were negotiated for the channel.

Every Connection holds the list of Metric IDs that are subscribed to it. When a new Metric Value is created, this information is used for determining whether the Metric Value will be encapsulated in a Subscribed or in a Deferred message (see below).

### 4.1.2 Message

Every Connection has a message queue for outgoing messages. There are 4 message priorities:

- Immediate: the message should be sent immediately before messages of other priorities. Immediate messages are used for command response codes and producer capabilities.
- Urgent: the message should be sent before messages with any other priorities except immediate messages. Urgent messages are used for authentication data and special metrics generated by the monitoring system itself.
- Subscribed: the message should be sent to the consumer when there are no messages with higher priority. This priority is used for metrics that are subscribed on a channel, or for control execution response messages.
- Deferred: the message should only be sent to the consumer if it has explicitly requested it. This priority is used for metrics that are buffered on a channel.

Messages within the same priority preserve the ordering they were added to the queue.

The GET command works by scanning the message queue and changing the priority of every messages matching the requested metric ID from "Deferred" to "Subscribed".

### 4.1.3 Protocol Encoding

The Protocol Encoding object is responsible for converting the data to be sent to the consumer from the internal in-memory representation to the representation that will be sent on the wire as well as converting commands received from the consumer from wire-format to internal representation.

The Protocol Encoding object to be used for a connection is decided when the network connection is established. This practically means that Protocol Encoding objects are directly bound to TCP port numbers. If in the future multiple Protocol Encoding objects are to be used simultaneously, they should be bound to different port numbers.

### 4.1.4 User

The User object represents an authenticated user. Its role is being a container for entities valid at the user level such as channels, Metric IDs, accounting records and resource limits.

### 4.1.5 Sensor

Sensors are encapsulating the implementation of one or more metrics. They are used as factories of Metric Instances.

### 4.1.6 Metric Instance

A Metric Instance represents a metric with a real value assigned to all of its formal parameters. Metric Instances are responsible for carrying out measurements according to their parameters and creating the appropriate Metric Values.

Metric Instances can be cached and shared between different users which reduces the resource costs for metrics that are commonly requested by several users simultaneously.

#### 4.1.7 Metric ID

Each Metric ID corresponds to exactly one COLLECT command which created it. Each Metric ID is bound to exactly one User. They contain all per-user information that is not relevant for the Metric Instance, such as filters.

Filters are used for implementing periodic measurements or the QUERY command. Multiple users may request different measurement periods but the Metric Instance has no knowledge about it. Instead, the Metric Instance is told to perform a measurement with a periodicity that is the largest common denominator of all the requested periods and a filter is used to determine which measurements should be queued for which Metric Instances.

The QUERY command can also be implemented using a filter which always matches but has a side-effect of destroying the Metric Instance after the first measurement. In future extensions of the monitoring system we want to experiment with providing the ability for the consumer to attach arbitrary filters to a Metric ID that may discard measurements based on threshold values, or only return data if it is different than the last measurement.

#### 4.1.8 Metric Value

Metric Values represent measured quantities. From the producer's aspect, response messages from the execution of controls also qualify as Metric Values.

#### 4.1.9 Actuator

Actuators are analogous to Sensors in that they encapsulate the implementation of one or more controls. They are used as factories of Control Instances.

#### 4.1.10 Control Instance

A Control Instance represents a control with a real value assigned to all of its formal parameters. Since invoking the same control twice means taking the same action twice, Control Instances are not cached and not shared like Metric Instances. Also, the level of Metric IDs is missing in the case of Control Instances.

## 5 Monitoring protocol

The protocol used for communication between producers and consumers was defined in [2]. This section describes changes to the protocol as well as the binary encoding that is used in the monitoring system prototype implementation. To provide a complete reference of all commands in one place this section also contains commands that are unchanged.

### 5.1 Protocol changes

The GET\_SPECIAL command that was used for internal communication between the consumer API and the producer and was foreseen to change is gone. The AUTH command is revised to allow for multi-step authentication. The following new commands have been defined to provide enhanced functionality: DEL\_CHANNEL, EXECUTE, COMMIT, QUERY and WRAP.

An additional type BOOLEAN representing logical true — false values is added to the list of supported data types.

To help interoperability with future protocol versions and discovering producer capabilities, when a channel is opened the producer now unconditionally sends a message (MON\_MID\_PRODUCER\_CAPS)

to the channel. This message contains the protocol version and a list of basic producer capabilities such as the list of supported authentication methods.

## 5.2 Protocol encoding

Communication between the consumer and the producer is encoded using a binary format similar to XDR [6]. As discussed in [2] the protocol encoding chosen should be efficient and the resource requirement of its interpretation should be low. This is important for ensuring that monitoring influences the system being monitored as little as possible and the monitoring system is able to handle large volumes of monitoring data efficiently.

The basic unit of the encoding has the length of 4 octets or 32 bits. This was chosen in favour of speed instead of compactness since many modern processors can process 32-bit data faster than smaller units. The basic types INT32, UINT32, INT64, UINT64 are encoded using network byte order (most significant bit first). The DOUBLE type is encoded according to the "Double Precision Format" from the IEEE 754 standard [3], using network byte order. The BOOLEAN type is encoded on 32 bits by setting all bits to 0 for *false* and setting all bits to 1 for *true*.

The STRING type is encoded by storing the length of the string as an UINT32 followed by the string itself without the terminating zero. If the length of the string is not divisible by 4, it is zero-padded to the next 4-octet boundary. The OPAQUE type is encoded like STRING.

RECORDs are encoded by encoding all the fields in the order they appear in the record definition.

Fixed length ARRAYs are encoded by encoding all array elements in order. Variable length ARRAYs are encoded by storing the number of elements as an UINT32 first, followed by the encoded array elements in order.

### 5.2.1 Argument lists

Argument lists are key-value lists that are used in multiple places. They are encoded by first storing the number of arguments in the list as an UINT32, then the list of argument names and their types as a string followed by the values of the arguments in order (Table 1).

Table 1: Encoding of argument lists

| Type   | Description              |
|--------|--------------------------|
| UINT32 | Number of arguments      |
| STRING | Argument names and types |
| N/A    | Argument values          |

### 5.2.2 Commands sent by the consumer

Every command sent by the consumer has the header format specified in Table 2. The encoding of the command-specific part will be described later.

Table 2: Common header for commands sent by the consumer

| Type   | Description                        |
|--------|------------------------------------|
| UINT32 | Command code                       |
| UINT32 | Command sequence number            |
| UINT32 | Length of the following data block |
| N/A    | Command-specific data              |

### 5.2.3 Messages sent by the producer

All the messages sent by the producer has the same header described in Table 3.

Table 3: Common header for messages sent by the producer

| Type   | Description                  |
|--------|------------------------------|
| UINT32 | Message ID                   |
| UINT32 | Length of the following data |
| N/A    | Message-specific data        |

Bits 0 – 23 of the Message ID contain the metric ID. Bits 24-31 are reserved for future use. Metric IDs fall into three categories:

- 0 – 127: Message has its own encoding. These messages currently: command status response (Table 4), producer capability list (Table 5), authentication response (Table 6).
- 128 – 255: Message is a special metric (Table 7)
- 256 –  $2^{24}-1$ : Message is a normal metric (Table 7)

### 5.2.4 Command status responses

The encoding of command status responses is shown on Table 4. The command result identifier is only sent if it is non-0.

Table 4: Encoding of command status responses

| Type   | Description                          |
|--------|--------------------------------------|
| UINT32 | MON_MID_CMD_RESPONSE (= 0)           |
| UINT32 | Length of the following data         |
| UINT32 | Command sequence number              |
| UINT32 | Command status code                  |
| UINT32 | Command result identifier [optional] |

The following status codes are currently defined:

- MON\_STATUS\_OK: The command was successful.
- MON\_STATUS\_UNKNOWN\_CMD: Unknown command.
- MON\_STATUS\_UNKNOWN\_METRIC: Unknown metric name or metric ID.
- MON\_STATUS\_UNKNOWN\_CHANNEL: Unknown channel ID.
- MON\_STATUS\_BADPARAM: Bad parameter value.
- MON\_STATUS\_PARAM\_TYPE: Parameter type mismatch.
- MON\_STATUS\_PARAM\_MULTIPLE: Non-multivalued parameter specified multiple times.
- MON\_STATUS\_PARAM\_UNKNOWN: Unknown parameter.
- MON\_STATUS\_PARAM\_MISSING: Required parameter is missing.
- MON\_STATUS\_AUTH\_NEEDED: Authentication needed.

- MON\_STATUS\_AUTH\_ERR: Authentication error.
- MON\_STATUS\_RES\_LIMIT: Resource limit exceeded.
- MON\_STATUS\_GENERIC\_ERR: Generic error in the producer.

### 5.2.5 Producer capabilities

Producer capabilities are sent to the consumer when a channel is opened or when they change (for example, installing an encryption layer over the channel might change the list of available authentication methods). The encoding is shown on Table 5. The version code contains the major version number in bits 16-31, and the minor version number in bits 0-15.

Protocols with the same major version are expected to be able to inter-operate.

Table 5: Encoding of producer capabilities

| Type          | Description                  |
|---------------|------------------------------|
| UINT32        | MON_MID_PRODUCER_CAPS (= 1)  |
| UINT32        | Length of the following data |
| UINT32        | Protocol version code        |
| Argument list | Producer capabilities        |

### 5.2.6 Authentication responses

Authentication response messages are sent by the producer when mutual or multi-step authentication is needed. Their encoding is shown on Table 6.

Table 6: Encoding of authentication responses

| Type   | Description                                |
|--------|--|
| UINT32 | MON_MID_AUTH_RESPONSE (= 2)                |
| UINT32 | Length of the following data               |
| N/A    | Authentication method specific opaque data |

### 5.2.7 Metric values

The encoding of metric values is shown on Table 7. The two high-order bits of the nanosecond part of the timestamp are reserved. In the future these bits might be used to extend the range of the timestamp.

Table 7: Encoding of metric values

| Type   | Description  |
|--------|--|
| UINT32 | Metric ID  |
| UINT32 | Length of the following data                           |
| UINT32 | Timestamp: seconds since 00:00:00 UTC, January 1, 1970 |
| UINT32 | Timestamp: nanoseconds                                 |
| N/A    | Measured data  |

### 5.3 Commands

#### 5.3.1 AUTH

##### Arguments

| Type   | Description           |
|--------|-----------------------|
| STRING | Authentication method |
| OPAQUE | Method-specific data  |

##### Description

The AUTH command authenticates the user to the monitoring system. Unless explicitly stated otherwise the consumer must not send any other commands to the producer before a successful authentication; trying to do so will result in an error. Issuing the AUTH command on an already authenticated channel will also result in an error.

The method-specific data contain the credentials needed to verify the identity of the user. For example, they can be a user name and a password for password authentication.

The AUTH command may return the status code `MON_STATUS_AUTH_NEEDED` to indicate that more authentication steps are needed. In this case exactly one authentication response message (see Table 6) will be sent that contains the producer's response to the consumer's challenge. After processing this message the AUTH command should be used again with leaving the authentication method string empty. This procedure should be repeated until the producer either accepts or rejects the authentication.

#### 5.3.2 COLLECT

##### Arguments

| Type          | Description      |
|---------------|------------------|
| STRING        | Metric name      |
| Argument list | Metric arguments |

##### Description

The COLLECT command instructs the monitoring system to create a metric instance with the given parameters. If this is successful, the response contains the metric identifier of the metric instance.

#### 5.3.3 STOP

##### Arguments

| Type   | Description |
|--------|-------------|
| UINT32 | Metric ID   |
| UINT32 | Channel ID  |

##### Description

The STOP command tells the monitoring system that no more metric values for this identifier should be sent to the specified channel. Metric values for this identifier that have been queued in the monitoring system for the specified channel will be lost.

If the metric identifier is still subscribed to other channels, both the GET and SUBSCRIBE commands can be used to receive further metric values for this metric id. If there are no other channels where

this metric identifier is subscribed, the STOP command will destroy the metric instance and all further references to this metric instance will result in an error.

### 5.3.4 SUBSCRIBE

#### Arguments

| Type   | Description |
|--------|-------------|
| UINT32 | Metric ID   |
| UINT32 | Channel ID  |

#### Description

The SUBSCRIBE command instructs the monitoring system that all metric values for the given metric identifier should be automatically sent to the specified channel. The same metric identifier can be subscribed to more than one channel by using the SUBSCRIBE command multiple times. If there were metric values queued for the given metric identifier on the current channel they will be copied to the destination channel.

### 5.3.5 BUFFER

#### Arguments

| Type   | Description |
|--------|-------------|
| UINT32 | Metric ID   |
| UINT32 | Channel ID  |

#### Description

The BUFFER command instructs the monitoring system that all metric values for the given metric identifier should be buffered on the specified channel. It is possible to use the SUBSCRIBE and BUFFER commands for the same metric identifier on different channels.

### 5.3.6 GET

#### Arguments

| Type   | Description |
|--------|-------------|
| UINT32 | Metric ID   |
| UINT32 | Channel ID  |

#### Description

The GET command performs the following tasks:

- For event-like metrics, if there are metric values queued for the given metric identifier on the given channel, the monitoring system will send them to the consumer.
- For continuously measurable metrics, a new measurement is requested from the appropriate sensor. When the measurement is ready, the measured metric value will be sent to the consumer on the given channel. The monitoring system makes no guarantee when (if ever) this metric value will be sent.

### 5.3.7 DEF\_CHANNEL

#### Arguments

| Type          | Description        |
|---------------|--------------------|
| STRING        | Destination URL    |
| Argument list | Channel parameters |

#### Description

The DEF\_CHANNEL command defines a producer initiated channel that the monitoring system should open to the specified external location.

### 5.3.8 SET\_CHANNEL

#### Arguments

| Type   | Description |
|--------|-------------|
| UINT32 | Channel ID  |

#### Description

The SET\_CHANNEL command changes the current channel to the one identified by the given channel identifier while keeping the network connection. If the destination channel had an open network connection it will be closed.

### 5.3.9 DEL\_CHANNEL

#### Arguments

| Type   | Description |
|--------|-------------|
| UINT32 | Channel ID  |

#### Description

The DEL\_CHANNEL command destroys a channel. Metric instances associated with the destination channel but not associated with any other channels are destroyed. All data queued on the channel are discarded and the network connection (if open) is closed. The value of 0 for the channel ID means the current channel.

### 5.3.10 COMMIT

#### Arguments

| Type   | Description    |
|--------|----------------|
| UINT64 | Transaction ID |

#### Description

The COMMIT command informs the producer that the consumer has successfully processed the metric value belonging to the given transaction identifier.

### 5.3.11 EXECUTE

#### Arguments

| Type          | Description       |
|---------------|-------------------|
| STRING        | Control name      |
| Argument list | Control arguments |

#### Description

The EXECUTE command executes the named control with the given arguments. The command response contains an identifier for the control's result. The result has the same format as a metric value and the identifier returned by the command response.

### 5.3.12 QUERY

#### Arguments

| Type          | Description      |
|---------------|------------------|
| STRING        | Metric name      |
| Argument list | Metric arguments |

#### Description

The QUERY command is an optimization for fast information retrieval. It is equivalent to a COLLECT-GET-STOP command sequence. Due to its nature it can only be used with continuously measurable metrics. It is guaranteed that at most one metric value will be sent by the producer as the result of the command. However it is not guaranteed when (if ever) that metric value will be sent since this depends on the sensor.

### 5.3.13 WRAP

#### Arguments

| Type   | Description          |
|--------|----------------------|
| STRING | I/O layer identifier |

#### Description

The WRAP command installs an I/O transformation layer on the current channel. This command can be used to turn on transparent compression or to activate data encryption.

Unlike the other commands WRAP must be used synchronously. That is, the consumer must wait for the completion of the command before issuing any other commands to the producer. If the command succeeds, any subsequent data sent or received on the channel should go through the transformation (compression, encryption, etc.) defined by this I/O layer.

The WRAP command unlike other commands, may be used on an unauthenticated channel if the producer advertised any available transformations in the producer capability list. Doing so might change the producer capabilities available to the consumer (e.g., if an encryption layer is installed, authentication methods using clear text secrets might become available). If this is the case, the command result identifier in the command status response will contain the MON\_MID\_PRODUCER\_CAPS value and the new set of producer capabilities will be sent to the consumer.

If the transformation layer requires any handshaking or other internal communication between the peers it should happen below the level of the monitoring protocol and is not described here.

## 6 Security

When a user wants to utilise a functionality provided by a service (such as monitoring) a trust relationship must be established between the user and the service. For this the user must check the authenticity of the service to decide if she really trusts it and the service must check the user’s identity and decide if she is authorised to perform the requested operation. The two steps of this process is called (mutual) authentication and authorisation.

In a grid environment where services and users belong to different administration domains appropriate facilities to establish trust relationships between different parties are very important for secure operation. Figure 6 depicts a typical scenario for monitoring that demonstrates the required security facilities.

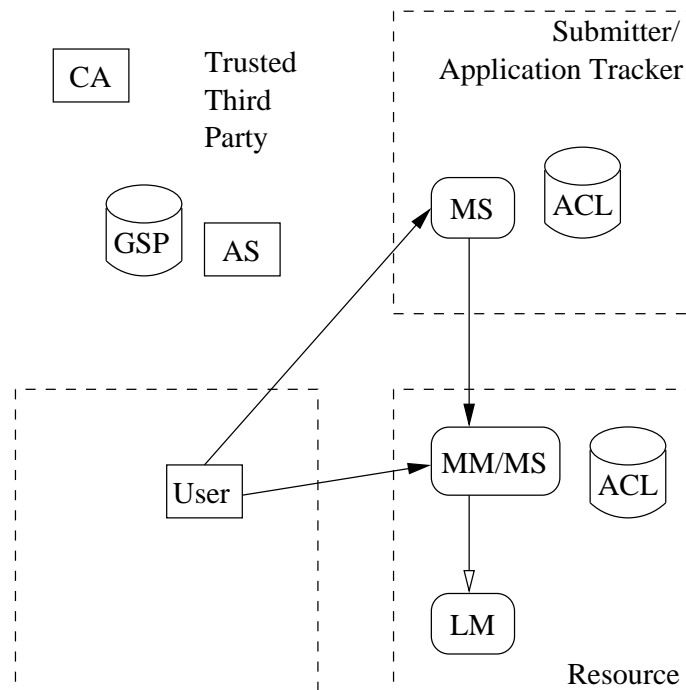


Figure 3: Authentication and Authorisation Scenario

The dashed boxes denote different administration domains enclosing the following grid entities:

- a grid resource that provides monitoring services via monitoring system components (LM, MM and MS),
- an intermediate grid service (the Submitter or Application Tracker) that provides monitoring information for running jobs via an MS component but is itself using the monitoring services provided by the grid resource,
- and a grid user who wants to access these services.

Authentication and authorisation is always required when crossing administration domain boundaries. This is denoted by arrows with filled heads in the figure. The services at the end of these arrows act as policy enforcement points applying local authorisation rules denoted by ACL. The authentication is

achieved via a third party in which both the user and the service trusts, certifying their identity. This trusted third party for authentication is called Certification Authority denoted by CA in the figure. For authorisation, apart from local policies there may be Grid Security Policies (labelled GSP in the figure) based on group (VO) membership and roles. These global policies are certified by another trusted third party called Authorisation Service denoted by AS.

When a user contacts a service directly (such as the MM/MS on a grid resource) simple mutual authentication and authorisation is sufficient. The service may contact another service inside the administration domain such as, the LM running on a host belonging to the resource, assuming the LM trusts the MM/MS to apply the appropriate local policies. This trust relationship between services is denoted by an arrow with an empty head in the figure.

However, when a service the user contacted initially such as, the MS of the Application Tracker contacts another service in another administration domain such as, the MM/MS on the grid resource, delegation is necessary to allow the MS to authenticate on behalf of the user. This is required to allow the MM/MS to authorise the user and apply the appropriate local policies as if the user had contacted it directly.

## 6.1 Interaction with the Authorisation Service

In the GridLab project the Security Work Package (WP6) provides the Authorisation Service that stores Grid Security Policies and can be used to obtain credentials corresponding to them. The monitoring system can use these credentials for authorisation as described below. Please note that following descriptions are significantly simplified as they do not cover for example all issues referring to mutual authentication and secure data transfer. The goal is to show the interaction between the monitoring system and the Authorisation Service.

Two types of interactions between the Monitoring Service and the Authorization Service is distinguished, connected with dividing the Monitoring Security Policy into static and dynamic parts. The static part, referred to as Implicit Rules, describes a small and static set of general rules (describing for example that the owner of a job has full privileges to monitor all data and processes belonging to it). The set of Implicit Rules can be downloaded during initialization of the Monitoring Service (and updated periodically). The dynamic part of security policy is handled by the User contacting the Grid Authorization Service in order to obtain a credential containing a part of his Security Policy referring to Monitoring. This credential is further used when a User requests performing of some operation by the Monitoring Service.

The following actions can be distinguished in this scenario:

1. Upon initialization, the Monitoring Service contacts the Grid Authorization Service and requests the set of Implicit Rules
2. The Grid Authorization Service provides the set of Implicit Rules for the specific Monitoring Service
3. A User sends a request to ask for performing an action by the Monitoring Service
4. The Monitoring Service makes an appropriate decision based upon the set of Implicit Rules obtained from the Grid Authorization Service and information available in the Monitoring Service configuration
5. A User may also send a request to the Grid Authorization Service to obtain a credential allowing to perform a specific action by the Monitoring Service
6. If the requested operation is allowed by the Grid Security Policy, the Grid Authorization Service issues the credential
7. The User sends the credential to the Monitoring Service requesting performing the action

8. The Monitoring Service verifies the validity of the credential and upon the results makes the appropriate decision

The general diagram of this scenario is presented in Figure 6.1.

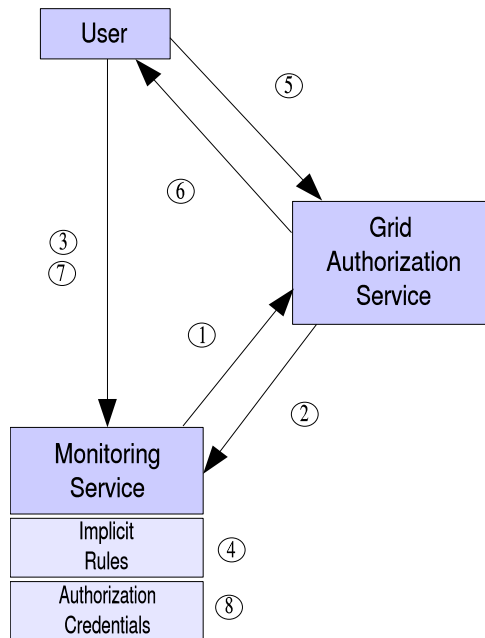


Figure 4: Interaction of the Monitoring System and the Authorisation Service

To provide better understanding of the scenario, consider the following example. Let’s assume that a User requests to monitor her process on some of the resources covered by the Monitoring Service. She requests performing a specific operation by the Monitoring Service. The Monitoring Service checks the set of Implicit Rules and ownership of the process and based on this information the decision is made. Now let’s assume that the User allowed her colleague (User2) to monitor her process and updated the Monitoring Security Policy referring to this process in the Authorization Service accordingly. User2 requests performing a specific operation by the Monitoring Service but he is rejected (he is not the owner of the process). Yet, he can obtain an appropriate credential from the Authorization Service. In this case, the decision will be made by Monitoring Service based on the credential issued by Authorization Service.

## 6.2 Access Control

The monitoring system uses Access Control Lists (ACLs) to control what services users are allowed to use. ACL entries can allow or deny access to certain functions for certain users. ACLs can be defined in the local configuration or can be provided by the Authentication Service (AS). When both types are present, ACLs defined in the local configuration should be applied after the ACLs received from the AS. The generic form of ACLs is:

<permission> <object> <requirements>

<Permission> can be either **allow** or **deny**. When testing for access permission, ACLs are evaluated in the order they defined. The result of the test is the <permission> value of the last ACL where <object> matches the requested operation and <requirements> matches the capabilities of the authenticated user entity. If there are no matching ACLs, permission is denied by default.

There are two categories for <object>:

- **metric** *metric name* [*metric parameters*]: The ACL refers to the metric *metric name*. If the optional *metric parameters* list is also given, the ACL matches only if the parameters supplied by the user match the parameters listed in the ACL.
- **control** *control name* [*control parameters*]: Likewise but for controls.

The <requirements> part is a boolean expression that can refer to the capabilities of the authenticated user entity and the parameters of the requested metric/control. At least the following expressions should be supported:

- **user** *user name*: true if the authenticated user identity is *user name*
- **group** *group name*: true if the authenticated user identity belongs to the named group
- **role** *role name*: true if the authenticated user has the named role
- **jobowner** *job id*: true if the authenticated user is the owner of the job. Usually *job id* is an expression referring to the parameters of the ACL's <object>.

### 6.3 Resource Limits

Resource limits has many similarities with ACLs. However contrary to ACLs, resource limits can only be set in the local configuration. The generic form of resource limit specifications is:

**limit** <object> [<requirements>] <value>

Resource limit <objects> can be either per-user or system wide. The <requirements> part has to be specified for per-user limits only and is interpreted as described in the previous section.

## References

- [1] Zoltán Balaton, Gábor Gombás: Requirements Analysis Report  
Technical Report, GridLab Project, March 2002.  
<http://www.gridlab.org/Project/Deliverables.html>
- [2] Zoltán Balaton, Gábor Gombás: Detailed Architecture Specification  
Technical Report, GridLab Project, June 2002.  
<http://www.gridlab.org/Project/Deliverables.html>
- [3] IEEE Computer Society: IEEE Standard for Binary Floating-Point Arithmetic  
ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, August 1985.
- [4] T. Ludwig, R. Wismüller: OMIS 2.0 – A Universal Interface for Monitoring Systems  
Proceedings of the 4th European PVM/MPI Users' Group Meeting, Crakow, Poland, November 1997.
- [5] R. Ribler, J. Vetter, H. Simitci, D. Reed: Autopilot: Adaptive Control of Distributed Applications  
Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, July 1998.
- [6] Raj Srinivasan: XDR: External Data Representation Standard  
IETF RFC 1832, August 1995.  
<http://www.ietf.org/rfc/rfc1832.txt>

- [7] Brian Tierney et al.: A Grid Monitoring Architecture  
Technical Report, Global Grid Forum, GMA-WG, March 2000.  
<http://www.gridforum.org/Documents/GFD/GFD-I.7.pdf>