IST-2001-32133
GridLab - A Grid Application Toolkit and Testbed

Final GAT API Specification: Object Based

Author(s):	Andre Merzky, Hartmut Kaiser
Document Filename:	Gridlab-1-GAT-0014.FinalAPISpecification
Work package:	Grid Application Toolkit
Partner(s):	VU
Lead Partner:	MPG
Config ID:	GridLab-1-FGS-0014-0.1
Document classification:	Public

Abstract: The Grid Application Toolkit (GAT) is a Grid Middleware abstraction layer for application programmers. It is able to communicate with the diverse services of the Grid environments via adaptors, which implement the GAT API capabilities. This document provides the final language-independent, object-oriented description of the GAT API.

Contents

1 Introduction

In the Grid community the demand for real applications using the emerging Grid infrastructure is steadily increasing. Over the last years much of the Grid Middleware advanced from pure experimental and research state to mature and widely supported software systems. The complexity of theses systems, both in terms of administration and application usability, remains high. One of these aspects, the usability for application programmers, is the focus of Grid Application Toolkit as developed in the EU GridLab project.

The purpose of the Grid Application Toolkit (GAT) is to provide a complete *application oriented* abstraction layer to the underlying Grid middleware. Application oriented here means, that the toolkit provides those functionality (capabilities) the application programmer needs. Such capabilities are often composed of several Grid operations, and may involve more than one Grid service, different protocols etc. Exactly that complexity should be hidden from the application programmer.

For example, the simple read access to a remote file may involve interaction with a replica location service, a data transport service, a GridFTP service, a resource management service, and a security service. It may involve communcation via LDAP/LDIF, GRAM, HTTP, and GSI, as protocols or protocol extensions.

All the application programmer should see are calls very much like:

```
fileCopy (source, destination);
```

Although that example is somewhat too simple, it surely illustrates the motivation for the present work. The API specified in this document is supposed to deliver a similar level of abstraction to the application programmer.

General GAT Design

One of the major strengths and major challanges of the Grid is its diverse nature in terms of technology used. The range of available Grid services is wide, and constantly growing. Although the Global Grid Forum (GGF) aims at a global standardization for these services, that efforts will take time; will not cover all Grid aspects; will not necessarily simplify usage of Grid middleware (on application level); and will not cover all Grid middleware systems (as research projects, propretiary systems etc.).

The GAT is designed to handle that diversity of Grid middleware. The API exposed to the application is, as far as possible, independent of the Grid middleware incarnation used.

In general terms, the capabilities provided by the GAT API implementation, a library, are handled by the GAT Engine. The GAT Engine picks from currently available adaptors, which implement that specific capability. The adaptors are lightwight modular software elements which interface to the specific Grid middleware to be used. The technical reasoning for this design is outlined in the GAT Technical Specification [2].

1.1 Scope of Document

As described, the GAT consists of a library — the **GAT Engine** — which implements the **GAT API**, and which all applications utilize; and a set of **adaptors** which provide the bindings between the operations called in the application and the underlying providers, e.g. Grid services. The GAT API can be split into four parts:

- **GAT Application API**

  This is the API which provides the "Grid" operations. It is called by applications which want to utilise the Grid infrastructure. This API part is very high level, and application oriented.

- **GAT Application Utility API**

  This API provides helper calls to support interaction between the application and the GAT Engine, and to support the exploitation of the GAT Application API.

- **GAT Adaptor Registration API**

  This is the mirror of GAT Application API and is the way in which adaptors provide those functions.

- **GAT Adaptor Utility API**

  These are the additional functions an adaptor needs to interact with the GAT Engine and possibly with other adaptors and utility libraries.

This document provides a language-independent, object-oriented description of the GAT Application API, and the GAT Application Utility API, based upon a set of descriptors. It is the second in a sequence of two language-independent API documents. The first is the GAT API SPECIFICATION [1]

As the GAT is an adaptation layer, it cannot enforce security in lower-level components — the adaptors and the capability providers they use. However, the GAT API will provide functions which allow the application to specify credential information to the underlying adaptors. Adaptors may use these information to authenticate to services and to delegate security, as determined by the underlying security model of these services.

Generally, all GAT calls should have asynchronous equivalents. However, this version of the API does not provide a general scheme for asynchronous GAT implementations, and touches that topic only on rare occasions (Monitoring) — such functionality may be added, at a general level, in later versions.

All GAT calls are reentrant (thread safe), unless noted otherwise.

1.2 Structure of Document

This document consists of three sections. Section 2 lists all the descriptors, with brief summaries of each, section 3 details the GAT Application API, and 4 details the GAT Application Utility

API. Section A describes classes used but not provided by the GAT. Section B provides a glossary about the terms used in this document.

Throughout the GAT API description in this document a number of examples are listed. That listing is provided in pseudocode, and is not supposed to represent real code.

The GAT Application API description is annotated with some sections explaining the separate API subsystems. These subsystems are loosely coupled and related API objects and calls, but do not define any particular substructure in the API — their nature is purely illustrative.

1.3 How to read this Document

1.4 Status of this Document

This is the final version of this document produced by the GridLab project.

1.5 RFC 2119 and this Document

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [4].

1.6 UML and this Document

Keywords in set in sans serif in this document are to be interpreted as described in the The Unified Modelling Language Reference Manual [5]. In addition, all figures in this document are written using the UML syntax and vocabulary [5].

2 API Descriptors List

2.1 GAT Application API Descriptors

GATRequest

An instance of this class represents a *request*, an action triggered by an external process which is requesting some information or some action to be performed.

GATRequestListener

This interface allows instances of classes which realize this interface to receive GATRequests.

GATRequestNotifier

When a GATRequest instance is created, an instance of a class realizing GATRequestNotifier is associated with it. When the request has been satisfied the application should invoke the Respond method on this instance. Only the originating adaptor should invoke the destructor on this instance.

GATMonitorable

Interface which can be realized by any model elements which are "monitorable." A model element is *monitorable* if it is capable of externalising its state.

GATMetric

An instance of this class represents a *metric*, a measurable quantity within a monitoring system.

GATMetricEvent

An instance of this class represents an metric event, an event indicating the measurement of a metric and the associated resultant data.

GATMetricListener

This interface allows instances of classes which realize this interface to receive GATMetricEvents from instances which are sources of GATMetricEvents.

GATJob

An instance of this class represents a monitorable process that may also be checkpointable.

GATSoftwareDescription

An instance of this class is a description of a piece of software (component) which is to be submitted to as a job.

GATSoftwareResourceDescription

An instance of this class is a description of software (component) which may be required for a piece of software to run.

GATHardwareResourceDescription

An instance of this class is a description of a node which may be required by a node or component.

GATResource

A "tagging" interface which is realized by any class which wishes to indicate it represents node or component; currently both a GATHardwareResource and a GATSoftwareResource realize this interface.

GATHardwareResource

An instance of this class is an abstract representation of a node which is monitorable.

GATSoftwareResource

An instance of this class is an abstract representation of a component which is monitorable.

GATResourceBroker

An instance of this class is used to broker resources.

GATReservation

An instance of this class is a reservation for a resource.

GATStreamable

GATStreamable is a interface which provides operations for connections to external entities such as files or other processes.

GATEndpoint

An GATEndpoint represents an endpoint of a GATPipe. An GATEndpoint can be created, and listened to, and connected to.

GATPipeListener

A class realizing this interface can process GATPipes produced by listening on GATEndpoints.

GATPipe

A GATPipe represents a connection to another process. It realizes GATStreamable and the communication methods are derived from that interface.

GATFileStream

A GATFileStream represents a seekable connection to an open file, the file may be either remote or local. It realizes the the GATStreamable interface and all communication operations are as documented there.

GATFile

An abstract representation of a physical file.

GATLogicalFile

An abstract representation of a set of identical physical files.

GATAdvertiseable

An interface which is realized by any class which wishes to be get advertised in a GATAdvertService.

GATAdvertService

The GATAdvertService allows GATAdvertisable instances to get published to and queried in an AdvertService. Such an AdvertService is any meta data directory with an hierarchical namespace attached.

2.2 GAT Application Utility API Descriptors

GATObject

Ancestor of all classes in the GAT API.

GATContext

An instance of this class is the primary GAT state object.

GATSecurityContext

A container for security information.

GAT<T>CredentialService

Classes binding to specific values of the parameter <T> provide methods to return specific security objects, given an instance of a GATSecurityContext. For example a GATGSICredentialService provides mechanisms to get GSI credentials, a GATSSLCredentialService provides access to an SSL security object.

GATSelf

This is a singleton class referng to the current application utilizing the GAT.

GATLocation

An instance of this class represents the location of an abstract or physical resource, as for example an URI.

GATPreferences

An instance of this class represents the user's preferences for selecting adaptors.

GATStatus

An instance of this class represents an error or an informational message which is returned by a GAT operation.

GATTime

An instance of this class represents a point in time.

GATTimePeriod

An instance of this class represents a length in time, with uncertain start and end point.

GATTable

An instance of the GATTable class maps keys to values.

2.3 GAT Adaptor Registration API Descriptors

The Adaptor Registration API Descriptors are language specific and are not covered in this document.

2.4 GAT Adaptor Utility API Descriptors

The Adaptor Utility API Descriptors are language specific and are not covered in this document.

3 GAT Application API Descriptors Descriptions

This section defines the public model elements. Note that private model elements are an implementation and language-specific detail and are thus not described here.

3.1 The GAT Advertisement Subsystem

The GATAdvertService provides an interface to a generic Grid meta data service. Such an services is a persistent external repository for any information which may be useful outside the application itself. Such information MAY include for example

- file names and locations published by the application,
- port and protocol information for contacting the application online,
- information about jobs started by the application.

For that purpose, the GATAdvertService allows one to annotate the appropriate GATObjects (which have to realize the GATAdvertisable interface) with arbitrary meta data, and to store that information in a meta data directory. That directory is also called Advertisement Directory, or Advert Directory (AD).

The meta data are considered to be a list of key value pairs, whereby keys and values are Strings. More powerful meta data schemes with structured and typed meta data elements seem useful and possible, but are not specified at the moment.
The meta data directory the information are stored in consists of a simple hierarchical namespace, where the nodes are tupels of an absolute path in that namespace, the GATObject published there, and the attached set of meta data.

Namespace

The namespace of the advert directory resambles a standard file system namespace. An `absolute path` is a String with a leading delimiter ('/'), plus a number of '/' delimited Strings (`path elements` or directories), plus a trailing undelimited String (`name`). A `relative path` does not have a leading delimiter, and refers to nodes relative to some other (absolute or relative) path. Relative Paths can contain `".."` as elements, which refer to the previous element of the referred absolute path.
Examples:

```
/www.zib.de/visual/data/test.h5  # absolute path
visual/data/test.h5              # relative path
../data/test.h5                  # relative path
test.h5                          # name of entry
```

The GATAdvertService has a current working directory, by default '/'. All relative paths and names are with respect to that directory initially. Allowed pathneames are as in the POSIX definition, but of arbitrary length.

Query Language

While querying the GATAdvertService, all entries in a directory are, recursively or non recursively, matched against a specified set of Query Meta Data (QMD). QMD are MD as specified above, with additional syntax for value elements which are interpreted during execution of the query. By the use of POSIX regular expressions [7], that syntax supports matching following cases (examples are in `this` font):

- presence/absence of a key
```
key="comment"  value="$^"          # key is not present
```
- empty values for a key
```
key="comment"  value="^$"          # key is present with empty value
```
- arbitrary values for a key
```
key="comment"  value=".*"          # key can have any or no value
key="comment"  value=".+"          # key can have any value
```
- specific values for a key
```
key="comment"  value="^text$"      # key can have a specific value
```
- regular expressions for values for a key
```
key="comment"  value=".*http://.*" # key matches a regex (contains http URL)
key="comment"  value="\d+"         # key matches a regex (is numeric)
```
- absolute paths of entries in namespace
```
key="GAT_PATH" value="/GAT/"       # entries in subtree /GAT/
```
- relative paths of entries in namespace
```
key="GAT_PATH" value="home/"       # entries in relative subtree home/
```
- names of entries in namespace
```
key="GAT_NAME" value="^test.*"     # entries with name matching 'test*'
```
- types of GATObjects stored
```
key="GAT_TYPE" value="GATEndpoint"  # entry object of type GATEndpoint
```

Key strings starting with `GAT_` are reserved for internal GAT specific AD entries. Entries with such keys cannot be used to store meta data. Currently, the user can use `GAT_TYPE`, `GAT_PATH` and `GAT_NAME`for queries. If paths and names are not specified, all entries relative to the current working directory of the Advert Service are searched.

Use Case Example

Ann runs a server application **A**, which creates a GATEndpoint `Endpoint_A`. Application **A** listens on that endpoint for incoming client connections in order to serve their requests. **A** advertises the GATEndpoint with the GATAdvertService.
Bill runs a client application, **B**, which wants to communicate with **A**. **B** queries the GATAdvertService for information about suitable GATEndpoints. Getting the path of such endpoints in the GATAdvertService hierarchy, **B** is able to reconstruct the GATEndpoint, and to connect to it.

```
Ann  creates a GATEndpoint.
Ann  creates a GATAdvertService.
Ann  advertises the GATEndpoint and meta data.
Ann  listens on the GATEndpoint and waits for
     incoming client connections
```

```
Bill creates a GATAdvertService.
Bill searches the GATAdvertService for suitable
     GATEndpoint entries
Bill selects one entry
Bill gets a GATEndpoint from that AD entry
Bill connects to that GATEndpoint (and hence to Ann)


----------------------------------------
--- Synchroneous case ------------------
----------------------------------------

Ann:

  GATAdvertService  A_AS = new GATAdvertService (...);
  GATEndpoint       A_EP = new GATEndpoint      (...);

  String PATH = new String  ("/Ann/GAT/Endpoints/EP1");
  GATable MD  = new GATable (...)
  MD.put ("name",        "myEndpoint")
  MD.put ("protocol",    "https")
  MD.put ("application", "A")
  MD.put ("owner",       "Ann_1")
  MD.put ("capability",  "test")

  AS.add (PATH, A_EP, MD);
  GATPipe A_Pipe  = A_EP.listen ();

  // read/write on pipe - communicate with B

Bill:

  GATAdvertService  B_AS = new GATAdvertService (...);
  B_AS.setPWD ("/Ann/GAT/");

  bool RECURSION = True;
  GATTable QUERY = new GATTable ()
  QUERY.put ("GAT_TYPE",    "GATEndpoint")
  QUERY.put ("protocol",    "https")
  QUERY.put ("application", ".+")
  QUERY.put ("owner",       "Ann_\d+")
  QUERY.put ("capability",  "test.*")

  list<String> PATHS  = B_AS.find (QUERY, RECURSION);
  GATEndpoint  B_EP   = B_AS.GetAdvertiseable (PATHS[0]);
  GATPipe      B_Pipe = B_EP.connect ();

  // read/write on pipe - communicate with A
```

3.2 interface GATAdvertisable

Description

An interface which is realized by any class which wishes to get advertized.

Classes realizing this interface need to provide a constructor which takes a serialization of the class (a String), and creates a class instance from the state information stored in that serialization. That is effectively the only way to de-serialice a GATAdvertisable object.

Apart from the constructor, the only operation for this interface is Serialize, which takes care of the encapsulation of the state of a GATAdvertiseable instance in a String. Though this method will likely not be used by the majority of application programmers directly, it will be used by adaptor writers who provide the functionality of a GATAdvertService.

Operations

Constructor This constructor takes the serialized state of an GATAdvertisable instance, and re-creates that instance.
Inputs:

GATContext — context — GATContext to be used for the recreated object instance.

String — serial — Contains the instance's serialization.

Serialize This operation obtains the serialized state encapsulated by this instance.
Outputs:

String — serial — Contains this instance's serialization.

3.3 class GATAdvertService specializes GATObject realizes GATMonitorable

Description

The GATAdvertService allows GATAdvertisable instances to get published to and queried in an advert directory. Such an advert directory is a meta data directory with an hierarchical namespace attached.

Use cases

In this use case the client wishes to advertise an GATAdvertisable instance using a GATAdvertService. The client first creates a String describing the path at which they wish to advertise the GATAdvertisable instance. Next the client creates an instance of the GATTable class and populates it with name/value pairs which are to be the meta-data associated with the GATAdvertisable instance. The client then creates an instance of the GATAdvertService class and calls the operation Add on this instance. This process is diagrammed in the following sequence diagram 1:

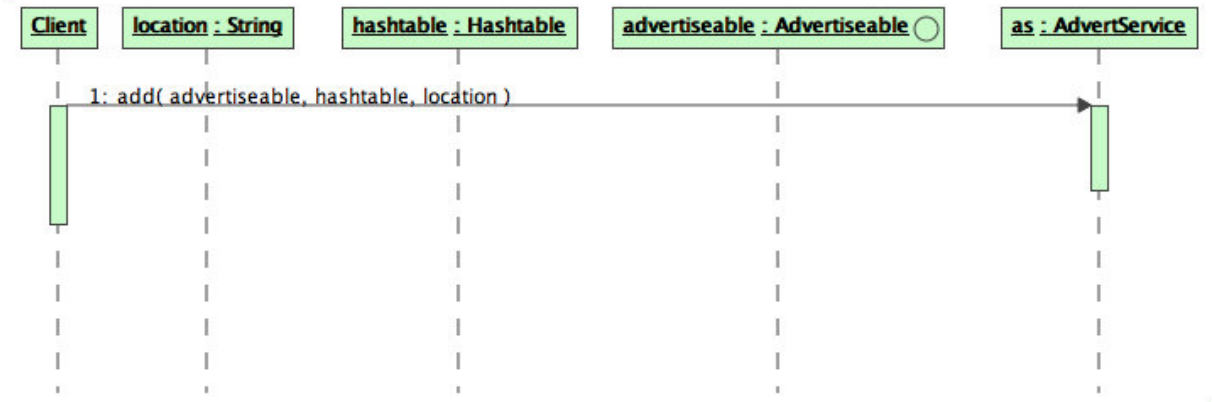

Figure 1: A sequence diagram for an instance of the class GATAdvertService.

In this use case the client wishes to find the path of GATAdvertisable instances in the GATAdvertService. The client first creates GATTable instance and populates it with name/value pairs which are to be the meta-data associated with the GATAdvertisable instance they wish to find. The client then creates an instance of the GATAdvertService class and calls the operation Find on this instance. This process is diagrammed in the following sequence diagram 2:

Operations

Constructor Creates an instance of the GATAdvertService corresponding to the passed GATContext.

Inputs:

GATContext — context — Used to broker resources.

GATPreferences — preferences — User preferences. (OPTIONAL)

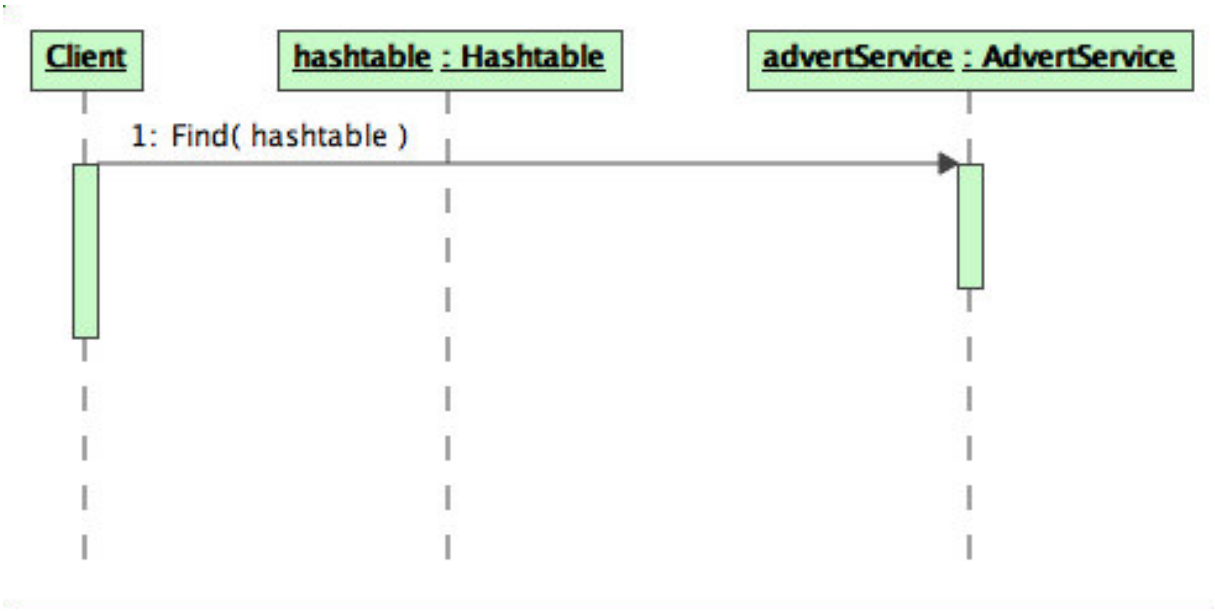

Figure 2: A sequence diagram for an instance of the class GATAdvertService.

Destructor Destroyes the instance of GATAdvertService.

Add Add an Advertizable instance and related meta data to the GATAdvertService, at path (absolute or relative to PWD). If an GATAdvertService entry exists at the specified path, that entry gets overwritten, and a warning is issued.
Inputs:

GATAdvertiseable — advert — instance to be entered into the GATAdvertService.

GATTable — metaData — Meta data to be associated with the passed GATAdvertiseable.

String — path — Path (relative to PWD) of the new entry.

Delete Deletes the specified entry from the GATAdvertService.
Inputs:

String — path — Path of the entry to be deleted.

GetMetaData Get the meta data for the specified AD entry. The returned meta data can be destroyed at any time.
Inputs:

String — path — Path of the entry of interest.

Outputs:

GATTable — metaData — Meta data of the entry of interest.

GetAdvertisable Get the Advertizable instance for the specified GATAdvertService entry. The returned instance can be destroyed at any time.
Inputs:

String — path — Path of the entry of interest.

Outputs:

GATAdvertiseable — advert — instance for the entry of interest.

Find Query the GATAdvertService for entries matching the specified set of meta data. The returned paths can be destroyed at any time.
Inputs:

GATTable — metaData — Meta data describing the entries to be searched for.

Outputs:

List of Strings — paths — Paths, each pointing to a matching entry.

SetPWD Specify the element of the GATAdvertService namespace to be used as reference for relative paths.
Inputs:

String — path — New absolute or relative reference path.

GetPWD Returns the current element of the GATAdvertService namespace used as reference for relative paths.
Outputs:

String — path — Absolute reference path.

3.4 The GAT File and Streaming Subsystem

An application exchanging data and information via data streams can use the GAT Streaming System. The system inherits from the known concepts of BSD sockets, but simplifies that interface to enable independent implementation.
The principal concept in GAT is to create an instance of GATEndpoint, and (as server) to listen on it for incoming connections, or (as client) connect to it. Both operations result in an instance of a GATPipe, which can be used for two sided communication. A third class of the system, GATFileStream, provides approximately the same interface for remote file access.
In order to motivate the design for the GATEndPoint and GATPipe objects, a simple use case is listed here.

GATPipe Use Case

Ann creates a communication endpoint (A) and advertises it. Ann can further perform a listen on the endpoint, and gets a GATPipe (pipe_A) on success.
Bill retrieves an advertisement from an GATAdvertService and uses it to construct a communication endpoint (GATEndpoint B), and tries to connect to A - he gets a GATPipe pipe_B. A connection is established between pipe_A and pipe_B.
The advertisment is still valid and can be used by another party to try to establish a connection with Ann.

```
Ann  creates a GATEndpoint.
Ann  uses the GATEndpoint to create a new GATPipe (A)
     by calling listen on endpoint (callback).
Ann  advertises the GATEndpoint.

Bill gets the GATEndpoint from a Advert Directory.
Bill creates a new GATPipe (B) using the GATEndpoint by calling
     connect on it.

Ann  and Bill can now write down their GATPipes
     (A and B respectively) or read data from them.

Ann  can continue to listen.

Cees does the same as Bill, and gets a third pipe.

----------------------------------------
--- Synchroneous case ------------------
----------------------------------------

Ann:

  GATEndpoint      A  = new GATEndpoint (...)
  GATAdvertService AS = new GATAdvertService (...)
  GATPipe     pipe_A1 = A.listen (...)
  AS.add     (A, ...)
  pipe_A1.read  (...)
```

```
  pipe_A1.write (...)
  delete (pipe_A1);

  GATPipe pipe_A2 = A.listen (...)
  pipe_A2.read  (...)
  pipe_A2.write (...)
  delete (pipe_A2);
  delete (A);

Bill:

  GATAdvertService AS = new GATAdvertService (...)
  list<String> paths  = AS.query (...)
  GATEndpoint  B = AS.getAdvertizable (paths[0])
  GATPipe pipe_B = B.connect ()
  pipe_B.write (...)
  pipe_B.read  (...)
  delete (pipe_B);
  delete (B);

Cees:

  GATAdvertService AS = new GATAdvertService (...)
  list<String> paths  = AS.query (...)
  GATEndpoint  C = AS.getAdvertisable (paths[0])
  GATPipe pipe_C = B.connect ()
  pipe_C.write (...)
  pipe_C.read  (...)
  delete (pipe_C);
  delete (C);

----------------------------------------
--- Asynchroneous case -----------------
----------------------------------------

Ann listens with GATPipeListener, and then uses the GAT event
servicing mechanism to serve incoming pipes.

Ann:

  GATEndpoint            A  = new    GATEndpoint      (...)
  GATPipeListener listen_A  = new my_GATPipeListener  (...)
  GATAdvertService       AS = new    GATAdvertService (...)
  AS.add       (A, ...)
  A.listen     (listen_A, ...)
  // do work   (callback handling)
  delete (A);
  delete (listen_A);
```

```
  sub GATPipeListener:ProcessPipe (GATPipe pipe_A)
  {
    pipe_A.read  (...)
    pipe_A.write (...)
    pipe_A.close (...)
  }

Bill:

  GATAdvertService AS = new GATAdvertService (...)
  list<String> paths  = AS.query (...)
  GATEndpoint  B = AS.getAdvertisable (paths[0])
  GATPipe pipe_B = B.connect ()
  pipe_B.write (...)
  pipe_B.read  (...)
  delete (pipe_B);
  delete (B);
```

Possible extensions

Depending on demand, it may be useful to extend the present Stream API. For example, the ability to perform striped or multiplexed reading/writing to collections of streams may be useful. Such things can be implemented on top of the present spec, and are to be defined separately.

3.5 interface GATStreamable

Description

GATStreamable is a interface which provides operations for connections to external entities such as files or other processes.

To send data down a GATStreamable it is necessary to construct a Buffer, and pack it with data. Similarly to receive data a buffer must be created in which the data will be stored.

The method of setting up a connection is the responsibility of the realizing class (e.g. GATFileStream or GATPipe.)

Operations

Read Reads from this GATStreamable into the given Buffer.

This is a blocking call and only returns if the buffer is full or if an error occurred. On error, the buffer MAY contain partially read data.

Inputs/Outputs:

Buffer — buffer — Buffer into which data are to be transferred

Outputs:

Integer — nbytes — Number of bytes successfully read into the buffer.

Write Writes data from the given Buffer through the GATStreamable. The buffer is not modified.

This is a blocking call and only returns when all the data in the buffer has been sent, or if an error occurs. On error, the number of bytes written is returned.

Inputs:

Buffer — buffer — Buffer from which data are to be transferred.

Outputs:

Integer — nbytes — Number of bytes successfully written.

Close Closes this GATStreamable instance.

3.6 class GATEndpoint specializes GATObject realizes GATMonitorable, GATAdvertiseable

Description

An GATEndpoint represents an end of a byte stream. Depending on how a GATEndpoint gets created, it can be listened to or connected to. In both cases, the endpoint returns a GATPipe. Hence, a GATEndpoint acts in fact as a GATPipe factory: multiple GATPipes can be created from it by repeatedly listening for incoming connections. The behaviour is similar to listening on a BSD socket.

GATEndpoints obtained from the GATAdvertService cannot be listened to.

GATPipes created from endpoints continue to live after the GATEndpoint instance is destroyed.

Operations

Constructor Constructs a unconnected instance of this class. The parameters given determine the endpoint attributes, such as host, port, protocol etc, and are interpreted by the adaptor.

Inputs:

GATContext — context — Used to broker resources.

GATTable — parameters — Endpoint construction parameters (OPTIONAL).

GATPreferences — preferences — User preferences for this instance. (OPTIONAL)

Destructor Destroys this object GATEndpoint. The destructor does not effect (e.g. destroy) any GATPipes created from this endpoint.

Connect When a GATEndpoint is obtained from an Advert Directory, it can be used to create a GATPipe connected to the advertising application, by calling connect on the GATEndpoint instance. This method takes no parameters. The GATEndpoint can be destroyed at any time without affecting the obtained GATPipe.
Output:

A GATPipe connected to the endpoint.

Listen The creator of an GATEndpoint can use the GATEndpoint to create GATPipes, which represent incoming connections. This is done by calling Listen on the GATEndpoint instance. The GATEndpoint can be destroyed at any time without affecting the obtained GATPipe.
Output:

A GATPipe connected to the endpoint.

Listen The creator of an GATEndpoint can use the GATEndpoint to create GATPipes, which represent incoming connections. This is done by calling Listen on the GATEndpoint instance. This call is asynchroneous, and returns immediately. The GATPipe creation is performed on the passed GATPipeListener object. The GATEndpoint can be destroyed at any time without affecting the obtained GATPipe.
Input:

GATPipeListener — listener — GATPipeListener object which handles incoming connections.

3.7 interface GATPipeListener

Description

This interface allows instances of classes which realize this interface to process GATPipes produced by listening on GATEndpoints.

Operations

ProcessPipe An instance of a class realizing this interface receives GATPipes produced by listening on GATEndpoints through calls to this operation.

Input:

GATPipe — pipe — The new GATPipe.

3.8 class GATPipe
specializes GATObject
realizes GATMonitorable, GATStreamable

Description

A GATPipe represents a connection to another process. It realizes GATStreamable and the communication methods are derived from that interface.

A GATPipe can get created from an GATEndpoint only, by calling its Listen or Connect methods.

Operations

No operations are defined.

Destructor Closes the connection, destroys this GATPipe object.

3.9 class GATFileStream specializes GATObject realizes GATMonitorable, GATStreamable

Description

A GATFileStream represents a seekable connection to open file, the file MAY be either remote or local. It realizes the the GATStreamable interface and all communication operations are as documented there. The GATFileStream semantics is similar to a standard Unix file-descriptor. It provides methods to query the current position in the file and to seek to new positions.
To write data to a GATFileStream it is necessary to construct a Buffer, and pack it with data. Similarly, in order to read data a buffer must be created in which the data will be stored. Since the file MAY be remote, writes and reads MAY either be blocking, or asynchronous. Asynchronous writes or reads must be completed by appropriate call. These operations are present as a result of realizing the GATStreamable interface and are as documented there.

Operations

Constructor This creates a GATFileStream attached to the physical file at the specified GATLocation. The file MAY be opened in several modes:

GATFileStream.GAT_Read — Open file for reading. The stream is positioned at the beginning of the file.

GATFileStream.GAT_Write — Truncate file to zero length or create file for writing. The stream is positioned at the beginning of the file.

GATFileStream.GAT_Readwrite — Open for reading and writing. The stream is positioned at the beginning of the file.

GATFileStream.GAT_Append — Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

Inputs:

GATContext — context — Used to broker resources.

GATLocation — location — Location of the file to open.

Integer — mode — Mode to open the file (read, write, readwrite, or append; class constants).

GATPreferences — preferences — User preferences.
(OPTIONAL)

Destructor Closed the stream, destroys this GATFileStream object.

Seek This changes the current position in the file. The position can either be calculated from the current position in the file or from the beginning or end.

Inputs:

Integer — off — Offset from **whence**.

Integer — whence — "Whence", one of beginning, current or end of file.

Class Constants

GAT_Read — Constant used to indicate read mode

GAT_Write — Constant used to indicate write mode

GAT_Readwrite — Constant used to indicate readwrite mode

GAT_Append — Constant used to indicate append mode

3.10 class GATFile specializes GATObject realizes GATMonitorable, GATAdvertiseable

Description

An abstract representation of a physical file.

An instance of this class presents an abstract, system-independent view of a physical file. User interfaces and operating systems use system-dependent pathname strings to identify physical files. GAT, however, uses an operating system independent pathname string to identify a physical file. A physical file in GAT is identified by a URI.

An instance of this GATFile class allows for various high-level operations to be performed on a physical file. For example, one can, with a single API call, copy a physical file from one location to a second location, move a physical file from one location to a second location, delete a physical file, and perform various other operations on a physical file. The utility of this high-level view of a physical file is multi-fold. The client of an instance of this class does not have to concern themselves with the details of reading every single byte of a physical file when all they wish to do is copy the physical file to a new location. Similarly, a client does not have to deal with all the various error states that can occur when moving a physical file (Have all the various bytes been read correctly? Have all the various bytes been saved correctly? Did the deletion of the original file proceed correctly?); the client simply has to call a single API call and the physical file is moved.

Use cases

In this use case the client wishes to move a physical file at a physical location, specified by the GATLocation instance location1, to a physical location, specified by the GATLocation instance location2. This is done by creating a GATFile instance which corresponds to the physical file at the GATLocation location1, then calling the Move operation on the so created GATFile instance. This is diagramed in the figure 3:

Operations

Constructor Constructs a GATFile instance which corresponds to the physical file identified by the passed GATLocation and whose access rights are determined by the passed GATContext.

Inputs:

GATLocation — location — Location which represents the URI corresponding to the physical file.

GATContext — context — Used to determine the access rights for this GATFile.

GATPreferences — preferences — User preferences for this GATFile. (OPTIONAL)

Equals Tests this GATFile for equality with the passed GATObject.

If the given GATObject is not a GATFile, then this operation immediately returns False.

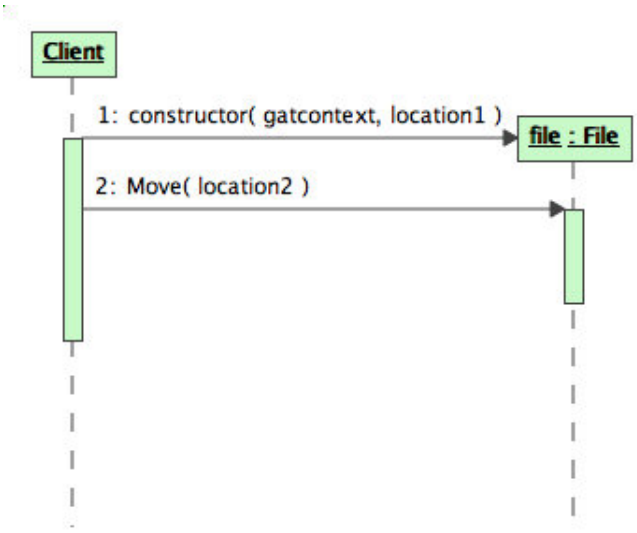

Figure 3: A sequence diagram for an instance of the class GATFile.

If the given GATObject is a GATFile, then it is deemed equal to this instance if a GATLocation object constructed from this GATFile's location and a GATLocation object constructed from the passed GATFile's GATLocation are equal as determined by the Equals operation of GATLocation.

Inputs:

GATObject — object — GATObject to test for equality.

Outputs:

Bool — equal — Boolean indicating equality.

Destructor Destroys this GATFile object.

Copy This operation copies the physical file represented by this GATFile instance, if such a physical file exists, to a physical file identified by the passed GATLocation. After successful completion of this operation this GATFile instance will not correspond to the new physical file.

Inputs:

GATLocation — targetLocation — Location to which to copy the physical file corresponding to this GATFile instance.

Integer — mode — overwrite mode for copy operation.

Move This operation moves the physical file represented by this GATFile instance to a physical file identified by the passed GATLocation. The mode specifies if the target file should be overwritten if it already exists. After successful completion of this operation this GATFile instance

will correspond to the new physical file. The mode is as in the copy operation.

Inputs:

GATLocation — targetLocation — Location to which to move the physical file corresponding to this GATFile instance.

Integer — mode — overwrite mode for copy operation.

Delete This operation deletes the physical file represented by this GATFile instance.

IsReadable Returns a boolean True if the physical file corresponding to this instance exists and if the physical file corresponding to this instance is readable .

Outputs:

Bool — readable — Boolean indicating readability.

IsWritable Returns a Boolean True if the physical file corresponding to this instance exists and if the physical file corresponding to this instance is writable.

Outputs:

Bool — writable — Boolean indicating writability.

GetLength Returns an Integer indicating the length in bytes of the physical file corresponding to this instance; then Integer 0 is returned if the corresponding physical file does not exist or if the corresponding physical file has length 0.

Outputs:

Integer — length — Length of the physical file corresponding to this instance, in bytes.

LastWriteTime Returns the time at which the corresponding physical file was last written; the Integer 0 is returned if the corresponding physical file does not exist.

Outputs:

GATTime — lastWriteTime — Time at which this file was last written.

3.11 class GATLogicalFile specializes GATObject realizes GATMonitorable, GATAdvertiseable

Description

An abstract representation of a set of identical physical files.

A GATLogicalFile is an representation of a set of identical physical files. This abstraction is useful for a number of reasons, e.g. if one wishes to replicate a physical file which is at one GATLocation to a second GATLocation. Normally, one takes all the data at the first GATLocation and replicates it to the second GATLocation even though the "network distance," between the first and second GATLocation may be great. A better solution to this problem is to have a set of identical physical files distributed at different locations in "network space." If one then wishes to replicate a physical file from one GATLocation to a second GATLocation, GAT can then first determine which physical file is closest in "network space" to the second GATLocation, chose that physical file as the source file, and copy it to the destination GATLocation. Similarly, the construct of a GATLogicalFile allows for migrating programs to, while at a given point in "network space," use the closest physical file in "network space' to its physical location.

A logical file has a logical file name. That name space is unique in the used namespace. The actual implementation of that namespace is done in a replica catalog, which maps that logical name to one or more physical names, as described above.

Since GAT may abstract from various implementations and incarnations of replica catalogs, the used name space is not necessarily uniq anymore. During the creation of the logical file, the catalog (and hence the name space) to be used can be specified, by adding a valid entry for the key "`catalog`".

The site configuration for GAT, and also the user specific configuration for GAT can specify what catalogs are available, by mapping a replica catalog service location to a key string.

Operations

Constructor This constructor creates a GATLogicalFile with passed name. As initial member, one location for a physical file MAY be given. The preferences MAY contain a key "`catalog`", whose value describes a replica catalog namespace to be used for the logical file name. If not given, a default catalog is used.

Inputs:

GATContext — context — Used to broker resources.

String — name — name in logical name space

Integer — mode — creation mode

GATLocation — location — Location of one initial physical file in this GATLogicalFile. (OPTIONAL)

GATPreferences — preferences — User preferences for this instance. (OPTIONAL)

The `mode` flag defines, if the logical file is created, and if it should be truncated.

Destructor Destroys this GATLogicalFile instance.

AddFile Adds the passed GATFile instance to the set of physical files represented by this GATLogicalFile instance.

Inputs:

GATFile — file — File to add to the set of physical files represented by this GATLogicalFile instance.

RemoveFile Removes the passed GATFile instance from the set of physical files represented by this GATLogicalFile instance.

Inputs:

GATFile — file — File to remove from the set of physical files represented by this GATLogicalFile instance.

Replicate Replicates the logical file represented by this instance to the physical file specified by the passed GATLocation.

Inputs:

GATLocation — location — Location of the new physical file.

GetFiles Returns a List of GATFile instances each of which is a GATFile corresponding to a physical file represented by this GATLogicalFile instance.

Inputs:

List of GATFiles — files — GATFile instances which correspond to physical files for this GATLogicalFile instance.

3.12 The GAT Event and Monitoring Subsystem

The GAT Event and Monitoring Subsystem covers both interaction with Grid monitoring services, and application defined events and actions.

An application may state that it can respond to some command (for example an externally triggered checkpoint), or provide additional metrics above those provided by a monitoring service. The application does this by providing a callback for each command, and for each piece of information it may provide. This is done by creating an object which realizes the GATRequestListener interface, and passing it to the AddRequestListener operation of the GATSelf object.

When an external process triggers a request, the GATEngine invokes the operation ProcessRequest on the GATRequestListener registered, through a call to AddRequestListener, to handle this request type. The GATEngine passes it an object which realizes the GATRequestNotifier interface. This GATRequestNotifier is used to return the results of the command or information request. The application MAY immediately perform the command or determine the information, then invoke the Respond operation on the passed GATRequestNotifier to respond, or it MAY invoke the Respond operation at a later time; e.g. in the case of checkpointing, the application MAY need to return to its normal control flow and only later notify the GATEngine that checkpointing has been completed.

Checkpointing

Currently the only command which MUST be supported by code implementing the GAT API is checkpointing.

As mentioned previously, when an external process triggers a command the GATEngine invokes ProcessRequest on the GATRequestListener registered to handle this command type. This ProcessRequest call passes the registered GATRequestListener a GATRequestNotifier. When this command is complete, the GAT application calls the Respond operation on this GATRequestNotifier. The GATTable passed to this Respond operation SHOULD contain the following keys

Checkpoint Files — a String with paths to any associated checkpoint files each of these paths MUST be formatted as if they were to be passed to the GATLocation constructor; furthermore, each path is quoted and comma-separated.

Restart Files — a String with paths to any other files which MAY be required in a restart case, e.g. new parameter files, etc. Again, these paths MUST be formatted as if they were to be passed to the GATLocation constructor, and, as above, they are quoted and comma separated.

Information Requests

Information requests are used to provide additional GATMetric's to a monitoring system. When registering such additional GATMetric's through a call to the AddRequestListener operation, a GATTable MUST be passed to the AddRequestListener operation. This GATTable is used to describe the new GATMetric which the application is allowing external processes to monitor. To describe this new GATMetric fully, this GATTable MUST have keys with values of the following types

Key	Value Type	Description
`Metric parameters`	`GATTable`	The metric parameters of the new metric.
`Metric measurement type`	`Integer`	The measurement type of the new metric.
`Metric data type`	`Type`	A language specific value type indicator
`Metric unit`	`String`	A String indicator of the measured value's unit.

Table 1: Information Requests: The key/values pairs defining a GATMetric.

More detail on the various values can be found in the GATMetric class documentation. In addition to the above information the call to the operation AddRequestListener contains a parameter which names the so constructed GATMetric. Note, the call to the operation AddRequestListener MAY fail if another GATMetric with the same name already exists — an error is issued in that case.

Upon an external process requesting information for a so registered GATMetric, the GATEngine calls ProcessRequest on the GATRequestListener registered to handle this information request. This ProcessRequest call passes the registered GATRequestListener a GATRequestNotifier. When the requested information has been gathered and is ready to be sent to the external process, the GAT application MUST call the Respond operation on this GATRequestNotifier. The application passes this Respond operation a GATTable. This GATTable is used to describe the GATMetricEvent the remote process will receive as a result of its information request. So, the GATTable passed to this Respond operation MUST contain the following keys with values of the following types

Key	Value Type	Description
`Metric value`	`Object`	A language specific structure giving the metric's value.
`Metric timestamp`	`GATTime`	A GATTime indicating information's harvest time.

Table 2: Information Requests: The key/values pairs defining a GATMetricEvent.

Use Cases

Checkpointing

The application indicates that it can respond to checkpointing requests; another GAT application checkpoints it by invoking the Checkpoint operation on the corresponding GATJob object.

The application creates an instance of a class realizing GATRequestListener, L.

The application passes L to GATSelf.AddRequestListener, stating that it is a "command" request listener, and that its name is "checkpoint". The GATTable can be empty.

At appropriate points in the flow of control of the application, it invokes GATContext.ServiceActions.

When an adaptor receives a checkpoint request, it creates a GATRequestNotifier, N and invokes the ProcessRequest operation on L with this N.

When L.ProcessRequest is invoked, the application notes that a checkpoint has been requested, and saves the GATRequestNotifier, N, for later use, then returns.

At a later point in processing, after the return of ServiceActions and checkpointing has been completed, the application invokes N.Respond.

When N.Respond is invoked, the adaptor informs the process which invoked the checkpoint of the data returned in the GATTable.

Application can provide number of iterations

The application indicates that it can respond to requests for the number of iterations since application startup; another GAT application requests this information by using the monitoring operations.

The application creates an instance of a class realizing GATRequestListener, L.

The application passes L to GATSelf.AddRequestListener, stating that it is an "information" request listener, and that its name is, say, "iteration_counter". The GATTable provides an empty GATTable for "Metric parameters", the constant Continuous for "Metric measurement type", "integer" for "Metric data type", and "iteration" for "Metric unit".

At appropriate points in the flow of control of the application, it invokes GATContext.ServiceActions.

When an adaptor receives a request for this information, it creates an instance of a class realizing GATRequestNotifier, N, and invokes the ProcessRequest operation on L with this N.

When L.ProcessRequest is invoked, the application determines its iteration and invokes N.Respond with the iteration stored under "Metric value", and the current time stored under "Metric timestamp".

When N.Respond is invoked, the adaptor informs the process which requested the information of the data returned in the GATTable.

3.13 interface GATRequestListener

Description

This interface allows instances of classes which realize this interface to receive GATRequestNotifier's.

Operations

ProcessRequest An instance of a class realizing this interface receives GATRequestNotifier's through calls to this operation when it is registered to receive such events.

Inputs:

GATRequestNotifier — requestnotifer — GATRequestNotifier used to signal the requests results operation call.

3.14 interface GATRequestNotifier

Description

When a particular type of request is sent to an application from a remote process, be it a command request or a information request, the ProcessRequest method is called on the GATRequestListener which has been registered to handle this particular type of request. When this ProcessRequest method is called, it is passed a GATRequestNotifier. This GATRequestNotifier is used by the ProcessRequest method to return to the remote process data pertinent to the remote process's request, be it GATMetricEvent data or command data.

Operations

Respond Passes any pertinent data back to the invoker.
Inputs:

GATTable — data — Return Data containing any response data.

3.15 interface GATMonitorable

Description

Interface which can be realized by any model elements which are "monitorable." A model element is *monitorable* if it is capable of externalising its state.

Use cases

In this use case the client wishes to register an instance of the class GATMetricListener to receive GATMetricEvents from an instance of the class GATMonitorable. In other words, the client wants the GATMetricListener instance to "monitor" the GATMonitorable instance. The client does so by obtaining an instance of the GATMetric class from the GetMetrics operation on the GATMonitorable interface, then providing appropriate values for the various parameters of this GATMetric instance. After doing so, the client passes the so modified GATMetric instance along with the desired GATMetricListener instance to the operation AddMetricListener. This is diagrammed in the figure 4

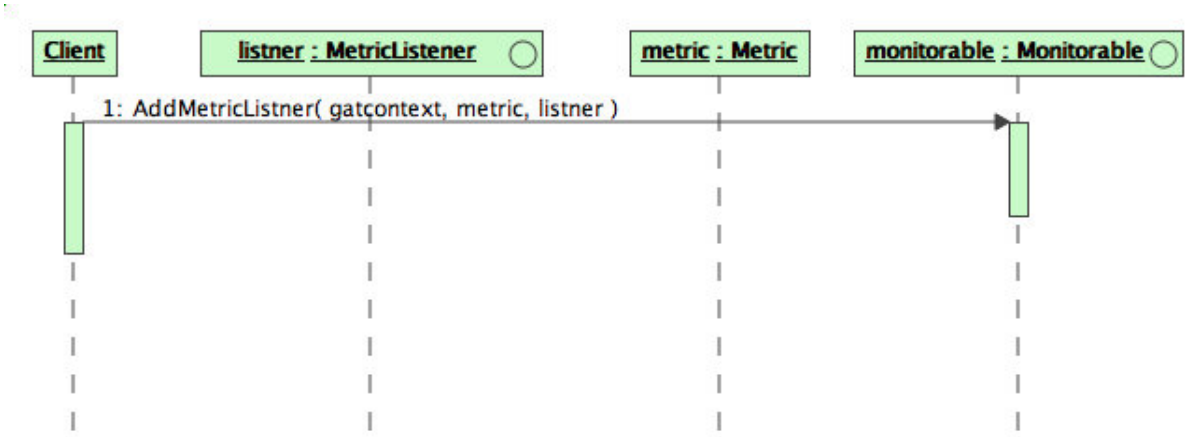

Figure 4: A sequence diagram for an instance of the interface GATMonitorable.

Operations

AddMetricListener Upon successful completion adds the passed instance of a GATMetricListener to the list of GATMetricListeners which are notified of GATMetricEvents by an instance of a class realizing this interface. Upon successful completion of this operation the passed GATMetricListener is notified of GATMetricEvents which correspond to GATMetric instance passed to this operation. The GATMetric instance passed to this operation must be obtained from the GetMetrics operation on this instance, and appropriate values for all the parameters of the passed GATMetric instance must be supplied or this operation will not complete successfully.

Inputs:

GATMetricListener — listener — GATMetricListener to notify of GATMetricEvents.

GATMetric — metric — GATMetric corresponding to the GATMetricEvents for which the passed GATMetricListener will be notified.

RemoveMetricListener Upon successful completion removes the passed GATMetricListener from the list of GATMetricListeners which are notified of GATMetricEvents corresponding to the passed GATMetric instance. The GATMetric instance passed to this operation must be obtained from the GetMetrics operation on this instance, and appropriate values for all the parameters of the passed GATMetric instance must be supplied or this operation will not complete successfully.

Inputs:

GATMetricListener — listener — GATMetricListener to notify of GATMetricEvents.

GATMetric — metric — GATMetric corresponding to the GATMetricEvents for which the passed GATMetricListener has been notified.

GetMetrics Upon successful completion returns a List of GATMetric instances. Each GATMetric instance in this List corresponds to a *metric* which can be monitored on this instance. To monitor the *metric* corresponding to a returned GATMetric instance one must supply appropriate values for all the parameters of the GATMetric instance and pass the modified GATMetric instance to the operation AddMetricListener with an appropriate GATMetricListener.

Outputs:

List of GATMetrics — metrics — Each GATMetric instance in this List corresponds to a *metric* which can be monitored on this instance.

GetMeasurement Upon successful completion, this function returns to the caller a GATMetricEvent corresponding to the monitorable quantity indicated by the passed GATMetric. Each GATMetric corresponds to a quantity which can be measured. Some GATMetric instances correspond to a quantities which can be measured in a continuous manner – such as the time one a remote computer – other GATMetric instances correspond to quantities which can be measured in a event-like manner – such as the key pressed by the user. If a GATMetric instance corresponds to continuous quantity, then this function allows one to measure such a quantity. One passes the GATMetric instance to this function and obtains in result a GATMetricEvent providing the corresponding measured quantity measured while this function was being called.

Inputs:

GATMetric — metric — GATMetric corresponding to the continuous quantity the caller wishes to measure. This *metric* must correspond to a continuous *metric*.

Outputs:

GATMetricEvent — metricEvent — A GATMetricEvent instance providing the measured quantity corresponding to the passed *metric*.

3.16 class GATMetric specializes GATObject

Description

An instance of this class represents a *metric*, a measurable quantity within a monitoring system. There are two types of metrics a monitoring system must deal with:

> **Local metrics** — Local metrics are metrics that are directly measured on a resource (e.g. on the local host or on a running application). These can be highly dependent on the physical parameters of the resource. Thus, local metrics originating from two different resources are not necessarily comparable (E.g. 1 hour CPU time on a 1 GHz Intel processor is different than 1 hour CPU time on an 800 MHz PPC processor although the numeric values are equal.) However resource administrators who know the configuration of the resource need local metrics for detailed monitoring of the status and operation of the resource.
>
> **Grid metrics** — Grid metrics are metrics that have predefined semantics thus, they are resource independent. Grid metrics are derived from one or more local metrics by applying a specific, well defined algorithm (such as unit conversion, aggregation or averaging). Because of this transformation grid metrics MAY have less precision or they could be less specific but are guaranteed to be comparable between different resources. Unlike local metrics which a local resource is free to change grid metrics must be agreed upon, standardised and introduced by community consensus.

Instances of the class GATMetric deal with both types of metrics.

A GATMetric can only be measured if there exists a sensor which can measure the quantity corresponding to the GATMetric. Such sensors are created by sensor developers and must embedded in a resource to allow the corresponding GATMetric or GATMetrics to be measured. (The creation of sensors is beyond the scope of this document and as such will not be covered.) Such sensors, once created, define the GATMetric or GATMetrics which they allow to be measured.

A metric definition contains the following information:

- Metric name
- Metric parameters
- Metric measurement type
- Metric data type
- Metric Unit

Metric name The Metric name is used to identify the metric definition (e.g. CPU usage). It consists of dot separated words, e.g. host.cpu.user. The last component of a metric name is the actual name of the metric, the preceding components are called scope. The scope can be used to group metrics as well as to differentiate between similar metrics defined at different levels (for example, CPU utilisation can be measured on a per-job or per-host level).

Metric parameters The Metric parameters field in the metric definition contains the formal definition of the metric parameters. Many metrics can be measured at different places simultaneously. For example, CPU utilisation can be measured on several hosts or grid resources. The metric parameters can be used to distinguish between these different metric instances.

Metric measurement type The Metric Measurement type can be *continuous* meaning data is always available or *event-like* meaning data only becomes available when some external event happens (e.g. a sensor embedded in an application can send events any time). Continuous metrics are only available using pull model delivery unless the user specifies a measurement frequency. (The reason for this is without a specified measurement frequency there is no event triggering the measurement of a continuous metric.) If the user asks for periodic measurements by specifying a measurement frequency the system will generate periodic events automatically. This avoids polling and allows the system to use the measurement frequency information to optimise measurements.

Metric data type The Metric data type contains the definition of the storage used for representing measurement data.

Metric Unit The Metric unit specifies the physical unit in which the metric is measured as a String. It is only valid for simple numeric types and Lists of these types. In the latter case it means the unit of all elements of the List.

To use the GATMetric class one must obtain an instance of the GATMetric class. One does so using GetMetrics operation on a GATMonitorable instance. The notion of GATMetric instance's parameters are necessary as the same metric could be measured at different places (e.g. on different hosts) at the same time. A GATMetric instances parameter values are used to differentiate between these measurements, e.g. instances of a GATMetric describing the available memory on two host are distinguished by the values of a parameter containing the host-name for example.

Operations

Equals Tests this GATMetric for equality with the passed GATObject.

If the given GATObject is not a GATMetric, then this operation immediately returns False.

For two GATMetric instances to be considered as equal they must have equal GATMetric names, a String, as determined my the Equals operation on String. In addition, they must have equal GATMetric parameters and values as determined by the Equals operation on GATTable.

Destructor Destroys this GATMetric object.

GetMetricName Gets the GATMetric name associated with this GATMetric.

Outputs:

String — metricName — Metric name.

GetMetricParameters Gets the GATMetric parameters associated with this GATMetric.

Outputs:

GATTable — metricParameters — Parameters of the GATMetric.

GetMetricParameterByName Gets the GATMetric parameter value associated with the passed GATMetric parameter name. The null Buffer is returned if there is no GATMetric parameter value with the passed name.

Inputs:

String — name — Parameter name for which to obtain the associated GATMetric parameter value.

Outputs:

Buffer — metricParameter — GATMetric parameter value, a Buffer, associated with the passed GATMetric parameter name.

Note: Currently, we expect the parameter value returned in `metricParameter` to be simple native typed, and hence castable to simple native types (like integers, floats and strings). With the future extended handling of typed data in Buffers, GAT will support more complex values for metrics.

GetMeasurementType Metrics can correspond to two types of measurable quantities "continuous" or "event-like." A *continuous* metric corresponds to a measurable quantity whose measurement may be made at any time. As an example, the measurement of the free disk space may be made at any time. An *event-like* metric corresponds to a measurable quantity which can only be measured at certain time and as a result when such a measurement is made the result is "pushed" to the measurement client, a GATMetricListener. As an example, the the measurement of the current key struck by a user may only be made when the user strikes a key. A event-like metric can not be used by the function GetMeasurement on the interface Monitorable but can be used by the AddMetricListener and RemoveMetricListener functions on this interface. Likewise a continuous metric can not be used by the AddMetricListener and RemoveMetricListener functions on the interface Monitorable but can be used by the function GetMeasurement function on this interface. This function returns one of the public class constants `GAT_Unknown`, `GAT_Continuous`, or `GAT_EventLike` indicating the type of this metric.

Outputs:

Integer — type — Type of this metric

Class Constants

The following integer constant are used to determine the type of an instance of this class:

- `GAT_Unknown`
- `GAT_Continuous`
- `GAT_EventLike`

3.17 class GATMetricEvent specializes GATObject realizes GATAction

Description

An instance of this class represents an metric event, an event indicating the measurement of a metric and the associated resultant data.

A metric event occurs whenever a monitored resource, sends out an event to the monitoring system. This can encompass almost any type of event from disk space running out to memory becoming available. The various events are defined by the various sensors. This topic is covered in more detail in the GATMetric documentation.

Operations

GetSource This operation returns an instance of the source of this GATMetricEvent.

Outputs:

GATObject — source — Source of this GATMetricEvent.

GetValue This operation returns the value corresponding to this GATMetricEvent.
Outputs:

Buffer — value — Contains the value of this GATMetricEvent.

Note: Currently, we expect the parameter value returned in `metricParameter` to be simple native typed, and hence castable to simple native types (like integers, floats and strings). With the future extended handling of typed data in Buffers, GAT will support more complex values for metrics.

GetMetric This operation returns an *instance* of the GATMetric to which this GATMetricEvent corresponds.

Outputs:

GATMetric — metric — Corresponds to this GATMetricEvent.

GetEventTime This operation returns the time when the event happened.

Outputs:

GATTime — eventTime — Time when the event happened.

3.18 interface GATMetricListener

Description

This interface allows instances of classes which realize this interface to receive GATMetricEvents from instances which are sources of GATMetricEvents.

Operations

ProcessMetricEvent An instance of a class realizing this interface receives GATMetricEvents through calls to this operation when it is registered to receive such events.

Inputs:

GATMetricEvent — event — Event which triggered this operation call.

3.19 The GAT Resource Subsystem

This part of the GAT API specification covers classes which are responsible for interaction with Resources and jobs on the Grid.

Use Cases for the GAT Resource Subsystem

Simple Job Submission (case A) A client wishes to schedule a job on any suitable resource. The client first creates a GATSoftwareDescription, and fills in all needed information describing the job and its software environment (e.g., executable, arguments, environment, required files, etc.). Then the client creates a GATHardwareResourceDescription, and fills in all needed information describing the hardware environment required by the job (e.g., memory, cpu speed, etc). From these descriptions the client then creates a GATJobDescription. That GATJobDescription is submitted to a GATResourceBroker instance, which returns a GATJob instance on successful scheduling.

```
User  creates a GATSoftwareDescription SD.
User  creates a GATHardwareResourceDescription HD.
User  creates a GATJobDescription using SD and HD.
User  creates a GATResourceBroker.
User  submits the GATJobDescription to the Resource Broker, and
     obtains a GATJob on success.

GATSoftwareDescription SD = new GATSoftwareDescription (...)
GATHardwareResourceDescription HRD = new GATHardwareResourceDescription (...)
GATJobDescription      JD = new GATJobDescription      (..., SD, HRD)
GATResourceBroker      RB = new GATResourceBroker      (...)

GATJob                 J  = RB.submit (JD)

delete (SD);   delete (HRD);
delete (JD);   delete (RB);
```

Simple Job Submission to a specific Resource (case B) This use case is similar to the previous one, but the client picks a specific compute resource, and submits the job to that resource. To do that, the client again creates a GATSoftwareDescription and a GATHardwareResourceDescription as above. The GATHardwareResourceDescription is passed to the GATResourceBroker instance, which returns a list of matching resources. The client picks one of these resources and creates a GATJobDescription from that resource and the GATSoftwareDescription. As before, the client submits the GATJobDescription instance to the GATResourceBroker, and on success gets a GATJob instance returned.

```
User  creates a GATHardwareResourceDescription HRD.
User  creates a GATResourceBroker.
User  queries the Resource Broker for Resources matching HRD.
User  picks one resource from the returned list of resources.
User  creates a GATSoftwareDescription SD.
User  creates a GATJobDescription using SD and the selected resource.
```

```
User  submits the GATJobDescription to the Resource Broker, and
     obtains a GATJob on success.

GATSoftwareDescription    SD = new GATSoftwareDescription (...)
GATHardwareResourceDescription    HRD = new GATHardwareResourceDescription (...)
GATResourceBroker         RB = new GATResourceBroker      (...)
list<GATHardwareResource> HL = RB.findResource            (HRD)
GATHardwareResource       HR = HL[0]

GATJobDescription         JD = new GATJobDescription      (..., SD, HR)

GATJob                    J  = RB.submit (JD)

delete (SD);   delete (HD);
delete (RB);   delete (HL);
delete (HR);   delete (JD);
```

3.20 class GATSoftwareDescription specializes GATObject realizes

Description

An instance of this class is a description of a piece of software (component) which is to be submitted as a job. It currently takes a table describing this piece of software's attributes to any underlying job submission system.

The GAT-API defines a minimum set of supported name/value pairs to be included in the GAT-Table used to construct a GATSoftwareDescription instance, as listed in table 5. This set MUST be supported by any implementation of the GAT-API. For any GATSoftwareDescription, the `location` MUST be specified.

RSL based description keys are accepted as an alternative, as listed in table 4.

Name	Type	description
`location`	`GATLocation`	software location.
`arguments`	`List<String>`	software arguments.
`environment`	`GATTable`	software environment, names/values are Strings.
`stdin`	`GATFile`	stdin from which the component reads.
`stdout`	`GATFile`	stdout to which the component writes.
`stderr`	`GATFile`	stderr to which the component writes.
`pre-staged files`	`List<GATFile>`	files which SHOULD be staged to the resource before the component is invoked.
`post-staged files`	`List<GATFile>`	files which SHOULD be staged from the resource after the component is finished.

Table 3: Software Description: the minimum set of supported name/values.

Operations

Constructor The class constructor takes a GATTable describing the software.

Inputs:

GATTable — attributes — Attributes describing this software component.

Destructor Destroys this instance.

Equals Tests this GATSoftwareDescription for equality with the passed GATObject. GATSoftwareDescription are equal if they have equivalent description tables.

Name	Type	description
rsl.directory	String	working directory.
rsl.executable	String	executable location.
rsl.arguments	list<String>	arguments to the program.
rsl.stdin	GATLocation	stdin from which the component reads.
rsl.stdout	GATLocation	stdout to which the component writes.
rsl.stderr	GATLocation	stderr to which the component writes.
rsl.count	Integer	number of executables to run.
rsl.hostCount	Integer	number of hosts to distribute on.
rsl.environment	GATTable	name/Value pairs in Strings.
rsl.maxTime	GATTime	maximal time.
rsl.maxWallTime	GATTime	maximal WALL time.
rsl.maxCPUTime	GATTimt	maximal CPU time.
rsl.jobType	String	single|multiple|mpi|condor|...
rsl.queue	String	target queue name.
rsl.project	String	project account to use.
rsl.dryRun	Boolean	if set, dont submit but return success.
rsl.minMemory	Integer	minimal required memory in MB.
rsl.maxMemory	Integer	maximal required memory in MB.
rsl.saveState	Boolean	keep job data persistent for restart.
rsl.restart=ID	String	restart job with given ID.

Table 4: Software Description: RSL type keys are also supported.

3.21 interface GATResourceDescription

Description

The GATResourceDescription interface forms the base for the GATSoftwareResourceDescriptions and GATHardwareResourceDescriptions classes; these are used to specify and find resources which may then be used, for example, to submit a GATJob to. It has an associated GATTable whose key/value pairs describe the resource.

A GATJob may have many requirements, both on software and hardware, which need to be satisfied to run, e.g. specific versions of the operating system, minimum amount of memory, presence of specific compilers or libraries on a system, etc. Each one of these requirements may itself possibly in turn depend on some other software or hardware requirement. Sometimes there may be possible alternatives, for example the GATJob may be able to use any one of a set of possible system libraries which might be installed. Hence, a complete resource description requires a list of possible specifications, and, ideally, some way of specifying allowable alternatives. In order to accommodate this, a GATResourceDescription has, as well as a table describing software of hardware resource requirements, a list of child GATResourceDescriptions, at least one of which must be satisfied in addition to the requirements of this GATResourceDescription. I.e. a GATResourceDescription is a tree, and it is matched if a path exists from the root of the tree to any leaf where the requirements of every node on that path are met.

Specifying Two Hardware Requirements which must both be met The user requires that the job be run on hardware which matches two different GATHardwareResourceDescrip-

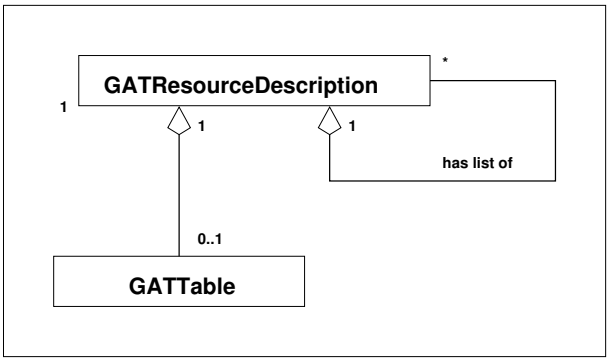

Figure 5: Classes realizing the GATResourceDescription interface contain descriptions of (software or hardware) resources (a GATTable), and optionally descriptions of other resources that resource depends on.

tions.

User creates a GATHardwareResourceDescription HRD1.

User creates a GATHardwareResourceDescription HRD2.

User invokes HRD1.AddResourceDescription(HRD2)

Note that this MUST be equivalent to

User creates a GATHardwareResourceDescription HRD1.

User creates a GATHardwareResourceDescription HRD2.

User invokes HRD2.AddResourceDescription(HRD1)

i.e. the order in which requirements appear on any single path through the tree MUST be irrelevant.

Specifying one Hardware Requirement and two alternative software requirements The user requires that the job be run on hardware matching a certain GATHardwareResourceDescription, but this hardware must have at least one of LAM or MPICH as MPI implementations.

User creates a GATHardwareResourceDescription HRD.

User creates a GATSofwareResourceDescription SRD1, specifying LAM.

User invokes HRD.AddResourceDescription(SRD1)

User creates a GATSofwareResourceDescription SRD2, specifying MPICH.

User invokes HRD.AddResourceDescription(SRD2)

Operations

Constructor The class constructor takes an optional GATTable describing the resource.

Inputs:

GATTable — description — Description for the resource. (OPTIONAL)

Destructor Destroys this instance.

Equals Tests this GATResourceDescription for equality with the passed GATObject. GATResourceDescriptions are equal if they have equivalent entries in the description table, and equivalent lists of dependent GATResourceDescriptions..

SetDescription This operation sets the resource description for this instance.

Inputs:

GATTable — description — Description for this resource's description instance.

GetDescription This operation returns the resource description for this instance.

Outputs:

GATTable — description — Description for the resource.

AddResourceAttribute Adds the name/value pair to the GATTable of name/value pairs which describe the resource.

Inputs:

String — name — Name to add to the name/value pairs describing the resource.

GATObject — value — Value to add to the name/value pairs describing the resource.

RemoveResourceAttribute Removes the name/value pair with the passed name from the GATTable of name/value pairs which describe the resource.

Inputs:

String — name — Name to of the name/value pair to remove from the name/value pairs which describe the resource.

AddResourceDescription This operation adds an instance realizing the GATResourceDescription interface (GATSoftwareResourceDescription or GATHardwareResourceDescription) to the list of dependent GATResourceDescriptions.

Inputs:

GATResourceDescription — description — Class instance realizing the GATResourceDescription interface to be added to the list.

RemoveResourceDescription This operation removes an instance realizing the GATResourceDescription interface (GATSoftwareResourceDescription or GATHardwareResourceDescription) from the list of dependent GATResourceDescriptions. An error is issued if the specified instance is not a member of the list of GATResourceDescriptions.

Inputs:

GATResourceDescription — description — Class instance realizing the GATResourceDescription interface to be removed from the list.

3.22 class GATSoftwareResourceDescription specializes GATObject realizes GATResourceDescription

Description

An instance of this class is a description of requisite software (a component) for some complete resource specification, for example pre-requisite software for a job to run, or the presence of a monitoring system to monitor a hardware resource or a running job. It accepts only specific values for keys on its GATTable containing the software resource description. These keys are specified in table 5.

Use Cases

To clarify the rather vague concept of what a GATSoftwareResourceDescription describes, the following list gives some examples:

- a system library,
- a compiler,
- a hosting environment,
- a helper application,
- a component,
- a plugin,
- a service
- an operating system,

E.g., a Fluent job requires the presence of Fluent, a java application requires a Java run-time environment.

In general any resource which corresponds to a component is described by a GATSoftwareResourceDescription. However, a hardware node is not described by a GATSoftwareResourceDescription.

The GAT-API defines a minimum set of supported name/value pairs to be included in the GATTable used to construct a GATSoftwareResourceDescription instance, as listed in table 5. This set MUST be supported by any implementation of the GAT-API.

Operations

Apart from the operations specified by the GATResourceDescription interface, no other operations are specified by this class.

Name	Type	Description
`os.name`	`String`	The os name as returned from `uname -s`.
`os.type`	`String`	The os type as returned from `uname -p`.
`os.version`	`String`	The os version as returned from `uname -v`.
`os.release`	`String`	The os release as returned from `uname -r`.
`os.name`	`String`	The os name as returned from `uname -s`.

Table 5: Software Resource Description: the minimum set of supported name/values.

3.23 class GATHardwareResourceDescription specializes GATObject realizes GATResourceDescription

Description

An instance of this class is a description of a hardware resource (node).

Use Cases

To clarify the rather vague concept of what a GATHardwareResourceDescription describes, the following list gives some examples:

- compute resources,
- network connections,
- a data storage element (e.g.hard drive),
- a graphics hardware resource,
- a supercomputer,
- a laptop,
- or similar pieces of hardware.

In general any resource which corresponds to a node is described by a GATHardwareResourceDescription. However, software components are not described by a GATHardwareResourceDescription.

The GAT-API defines a minimum set of supported name/value pairs to be included in the GATTable used to construct a GATHardwareResourceDescription instance, as listed in table 6. This set MUST be supported by any implementation of the GAT-API.

Operations

Apart from the operations specified by the GATResourceDescription interface, no other operations are specified by this class.

Name	Type	Description
`memory.size`	`Float`	The minimum memory in GB.
`memory.accesstime`	`Float`	The minimum memory access time in ns.
`memory.str`	`Float`	The minimum sustained transfer rate in GB/s.
`machine.type`	`String`	The machine type as returned from `uname -m`.
`machine.node`	`String`	The machine node name as returned from `uname -n`.
`cpu.type`	`String`	The generic cpu type as returned from `uname -p`.
`cpu.speed`	`Float`	The minimum cpu speed in GHz.
`disk.size`	`Float`	The minimum size of the hard drive in GB.
`disk.accesstime`	`Float`	The minimum disk access time in ms.
`disk.str`	`Float`	The minimum sustained transfer rate in MB/s.

Table 6: Hardware Resource Description: The minimum set of supported name/values.

3.24 interface GATResource

Description

GATResource is a base interface which is realized by any class which wishes to indicate it represents a node or component; currently both a GATHardwareResource and a GATSoftwareResource realize this interface. A GATReservation may be associated with this GATResource, and can be obtained by the operation GetReservation.

Operations

Destructor Destroys this GATResource instance, and cancels all pending reservations associated with it.

GetResourceDescription Gets the GATResourceDescription which describes this GATResource instance.

Outputs:

GATResourceDescription — rd — Describes this GATResource instance.

GetReservation Gets the GATReservation associated with this GATResource instance, if any — otherwise an error is issued.

Outputs:

GATReservation — reservation — An GATReservation instance associated with this GATResource.

3.25 class GATSoftwareResource specializes GATObject realizes GATMonitorable, GATResource, GATAdvertiseable

Description

An instance of this class presents an abstract, system-independent, view of a specific software component described by a GATSoftwareResourceDescription. It allows one to monitor the component, and to examine the various properties of the component to which this instance corresponds.

Operations

Constructor Constructs a GATSoftwareResource instance corresponding to the passed GATContext instances.

Inputs:

GATContext — context — Used to broker resources.

Destructor Destroys this GATSoftwareResource instance.

3.26 class GATHardwareResource specializes GATObject realizes GATMonitorable, GATResource, GATAdvertiseable

Description

An instance of this class presents an abstract, system-independent view of a hardware node described by a GATHardwareResourceDescription . It allows one to monitor the node, and to examine the various properties of the node to which this instance corresponds.

Operations

Constructor Constructs a GATHardwareResource instance corresponding to the passed GATContext instances.

Inputs:

GATContext — context — Used to broker resources.

Destructor Destroys this GATHardwareResource instance

3.27 class GATJobDescription specializes GATObject

Description

An instance of this class describes a job to be run. It consists of a description of the "executable" (a GATSoftwareDescription), and of a description of the resource requirements of the job. The latter can be given as either a GATResourceDescription, or as a specific GATResource; only one of these may be specified.

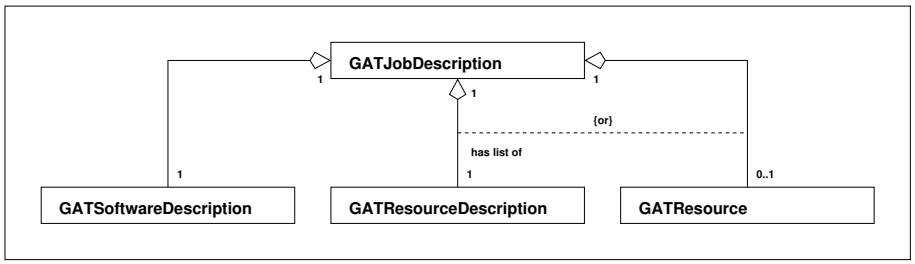

Figure 6: The class GATJobDescription contains information about the "executable" (GATSoftwareDescription), and of the resources the job be executed on (GATResourceDescription or GATResource).

Operations

Constructor The class GATJobDescription takes a GATContext as argument, and optionally accepts either one of the following set of arguments:

(GATSoftwareDescription, GATResourceDescription)

(GATSoftwareDescription, GATResource)

Inputs:

GATContext — context — GATContext used to broker resources.

GATSoftwareDescription — sd — Description of the job executable.

GATResourceDescription — rd — Description of the resources the job can be run on.

Constructor The second constructor, which accepts a GATResource instead of a GATResourceDescription.
Inputs:

GATContext — context — GATContext used to broker resources.

GATSoftwareDescription — sd — Description of the job executable.

GATResource — res — Resource the job SHOULD be run on.

Destructor Destroys this GATJobDescription instance.

3.28 class GATJob specializes GATObject realizes GATMonitorable, GATAdvertiseable

Description

An instance of this class represents a distinct unit which is submitted to a resource broker.

A GATJob instance gets created by a GATResourceBroker instance, on successful submission of a GATJobDescription.

Upon creation, the GATJob will not, in general, have been submitted to a specific resource, or queueing system. As time progresses the GATJob's will be submitted to a jobs scheduler somewhere, and eventually start to run. Thus a GATJob has various possible states:

- **GAT_Initial**
- **GAT_Scheduled**
- **GAT_SubmissionError**
- **GAT_Running**
- **GAT_Stopped**

The state may change automatically, or by the some operation performed on the GATJob. The states are defined as follows

GAT_Initial The GATJob instance has been submitted to an underlying resource management service but has not yet been submitted to a specific resource.

GAT_Scheduled The GATJob instance has been submitted by an underlying resource management service to a resource, and the GATJob is scheduled to run.

GAT_SubmissionError An error occurred when trying to submit this job, either to an underlying resource management service, or when such a service tried to submit to a queuing system or otherwise dispatch the job. A description of the error SHOULD be available from a GATStatus instance available by invoking the GetStatus operation.

GAT_Running The process represented by the GATJob instance is being executed.

(GAT_Stopped
The GATJob instance was running but is not currently running. That state can be reached by a successful call to the operation Stop, or due to the GATJob completing, or crashing.

Operations

Constructor The class GATJob does not have a publicly available constructor— instances get created by a GATResourceBroker instance. The initial GATContext used by the GATJob is that of the creating GATResourceBroker instance. The only other way to create a GATJob instance is by de-serialisation the class implements (it realizes the GATAdvertisable interface).

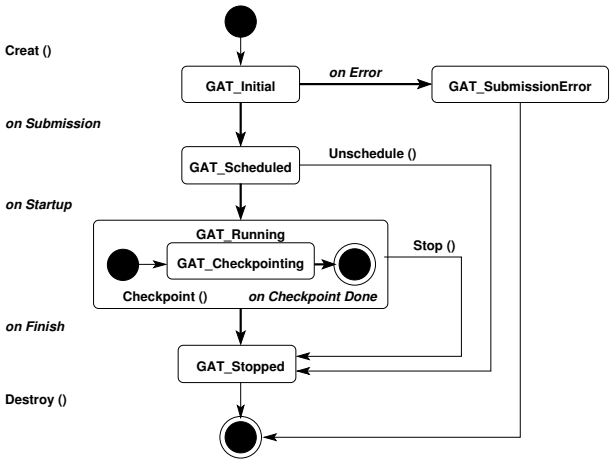

Figure 7: A state diagram for an instance of the class GATJob. The State can be actively changed by calling the operations unschedule and stop. The bold arrows are state changes induced by the resource management systems, and are out of direct user control.

Destructor Destroys this GATJob object.

UnSchedule Unschedule this GATJob. Upon successful completion this method guarantees that this GATJob is not scheduled to a job queue, its state is GAT_Finished. This operation can only be called on a GATJob in the GAT_Scheduled state, otherwise an error will is issued.

Checkpoint Trigger a checkpoint for the GATJob. The call can only succeed on processes which support application level checkpointing, or on resources which provide system level checkpointing. The call returns immediately after delivering the checkpointing request to the job or to the resource the job is running on; the actual checkpoint may happen some time after this return.
During checkpointing, the process state is saved to long term storage. That state information SHOULD be sufficient to restart the job, even on a different resource (migration) or with multiple copies (cloning). This is useful for processes which involve significant data manipulation and can not, for any number of reasons, finish all of this data manipulation in a single run; also, it is useful for processes which involve significant data manipulation and are, for any number of reasons, unstable. In both of these cases checkpointing allows the process to manipulate some data now, then, at a later date, continue running. This operation can only be called on a GATJob in the GAT_Running state, otherwise an error will be issued.

Clone The Clone operation creates a copy of the GATJob. The resulting GATJob has the same GATSoftwareDescription in its GATJobDescription, but the GATResourceDescriptions or GATResources of its GATJobDescription may be altered.
This operation upon success completes the following steps:

1. Constructs a new GATJobDescription instance with the GATSoftwareResourceDescription used to construct this GATJob instance.

2. Complete the new GATJobDescription instance with the optional GATHardwareResource passed to the Clone operation.
3. If no GATHardwareResource is given, use the GATHardwareResourceDescription or GATHardwareResource used to construct the current job.
4. Configures the new GATJobDescription instance so that when it begins running it will have the same state as the state saved in the last call to checkpoint on this GATJob instance.
5. Returns this new GATJob instance to the caller.

This operation can only be called on a GATJob instance on which the operation Checkpoint has been successfully called at least once, otherwise an error will be issued.
Inputs:

GATHardwareResource — hr — Hardware resource the clone SHALL run on. (Optional)

Outputs:

GATJob — clone — Cloned GATJob instance.

Migrate The Migrate operation provides similar functionality to the Clone operation. The only difference is that the calling GATJob instance is discontinued after the new job is spawned off successfully — its state is GAT_Stopped then.
Inputs:

GATHardwareResource — hr — Hardware resource the clone SHALL run on. (Optional)

Outputs:

GATJob — newJob — Migrated GATJob instance.

Stop Stops the GATJob. Upon a successful call to this operation, the processes associated with the GATJob are forcibly terminated. This operation can only be called on a GATJob in the GAT_Running state.

GetJobDescription This operation returns the GATJobDescription instance used to create that GATJob instance.
Outputs:

GATJobDescription — jobDescription — GATJobDescription for this GATJob instance.

GetState This operation returns the state of the represented process. This is one of the associated public class constants GAT_Initial, GAT_Scheduled, GAT_Running, or GAT_Stopped.

Outputs:

Integer — state — State of the represented process.

GetInfo This operation returns an instance of the class GATTable.

Outputs:

GATTable — info — Contains information about the associated GATJob.

This GATTable contains a set of key/value pairs the key, a String, being the name of the information and the value being the value of the associated named information. The minimum set of keys which the returned GATTable contains is as follows:

- **hostname**
- **scheduletime**
- **starttime**
- **stoptime**
- **checkpointable**

hostname The key hostname corresponds to a String value which is the name of the host on which the represented process is running, if GATJob is in the GAT_Running state, or will be running on, if GATJob is in the GAT_Scheduled state. If the associated GATJob is not in the GAT_Running or GAT_Scheduled state, then the value is empty.

scheduletime The key scheduletime corresponds to an Integer value which is the number of milliseconds after January 1, 1970, 00:00:00 GMT when the represented process was scheduled. This value is empty for a GATJob in the GAT_Initial state.

starttime The key starttime corresponds to an Integer value which is the number of milliseconds after January 1, 1970, 00:00:00 GMT when the represented process was started. This value is empty for a GATJob in the GAT_Scheduled or GAT_Initial states.

stoptime The key stoptime corresponds to an Integer value which is the number of milliseconds after January 1, 1970, 00:00:00 GMT when the represented process stopped, for a GATJob in the GAT_Stopped state, otherwise it is empty.

checkpointable The key checkpointable corresponds to a boolean value. This value indicates if the represented process is able to be checkpointed.

Other key/value pairs will be in future added to the list of key/value pairs returned in this GATTable as the need develops.

GetJobID This operation returns the job id, a globally unique identifier for the represented process corresponding to this instance. This operation SHOULD be called on a GATJob instance only when the instance is in a GAT_Running or GAT_Scheduled state, otherwise an error will be issued.

Outputs:

String — jobID — Job ID.

GetStatus When the job is in the GAT_SubmissionError state, this operation SHOULD provide a GATStatus instance which gives further information as to the cause of the error.
Outputs:

GATStatus — status — An object providing further information about the submission error.

Class Constants

The following integer constant are used to determine the state of an instance of this class:

- `GAT_Initial`
- `GAT_Scheduled`
- `GAT_SubmissionError`
- `GAT_Running`
- `GAT_Stopped`

3.29 class GATResourceBroker specializes GATObject

Description

An instance of this class is used to broker Resources. A resource can either be a node or a component.

Resources are found by constructing a GATResourceDescription instance specifying the desired properties of the resource or resources, e.g. amount of memory, amount of disk-space, installed software, etc.

Resources may either be found and reserved in one step by calling the ReserveResource operation with a GATResourceDescription, and, optionally, information describing the desired starting time and duration of the reservation, or resources may first be found by invoking the FindResources operation, which returns a list of GATResources satisfying the GATResourcedescription, and then the ReserveResource operation may be invoked with a specific resource (and optional time information); in either case a GATReservation instance is obtained.

A GATResource instance may then be extracted from the GATReservation, and this GATResource used to construct a GATJobDescription.

GATResources found by the FindResources operation may also be monitored.

Use cases

In this use case the client wishes to obtain a reservation for a hardware resource for a particular time period.

```
User  creates a GATTimePeriod TP.
User  creates a GATHardwareResourceDescription HRD.
User  creates a GATResourceBroker.
User  submits the GATHardwareResourceDescription and the
     GATTimePeriod to the Resource Broker, and
     obtains a GATReservation on success.

GATTimePeriod                  TP  = new GATTimePeriod                  (...)
GATHardwareResourceDescription HRD = new GATHardwareResourceDescription (...)
GATResourceBroker              RB  = new GATResourceBroker              (...)

GATReservation                 R   = RB.ReserveResource (HRD, TP)

delete (TP);   delete (HRD);
delete (RB);   delete (R);
```

Operations

Constructor This operation constructs a GATResourceBroker instance using the passed GATContext and GATPreferences.

Inputs:

GATContext — context — Used to broker resources.

GATPreferences — preferences — User preferences.
(OPTIONAL)

GATString — vo — Used to indicate the virtual organization
to which this resource broker is to be bound. resources.

Destructor Destroys this GATResourceBroker object.

ReserveResource This operation attempts to reserve the specified resource at the specified time, for the specified time period. Upon success this operation returns a GATReservation, which has an associated GATResource attribute representing the successfully reserved resource. Upon failing to reserve the specified resource this operation issues an error.

Inputs:

GATResourceDescription — rd — Description of the resource to reserve.

GATTime — time — Time for reservation to start.
(OPTIONAL)

GATTimePeriod — duration — Time period for which to reserve the resource.
(OPTIONAL)

Outputs:

GATReservation — reservation — Reservation upon success.

ReserveResource This operation attempts to reserve the specified resource at the specified time, for the specified time period. Upon success this operation returns a GATReservation, which has an associated GATResource attribute representing the successfully reserved resource. Upon failing to reserve the specified resource this operation issues an error.

Inputs:

GATResource — res — GATResource instance corresponding to the resource to reserve.

GATTime — time — Time for reservation to start.
(OPTIONAL)

GATTimePeriod — duration — Time period for which to reserve the resource.
(OPTIONAL)

Outputs:

GATReservation — reservation — Reservation upon success.

FindResources This operation attempts to find one or more matching resource (s). Upon success this operation returns a List of GATResource instances. Upon failing to find any specified resource an empty list is returned.

Inputs:

GATResourceDescription — rd — Description of the resource (s) to find.

Outputs:

List of GATResources — resources — Resources found.

SubmitJob This operation takes a GATJobDescription, and submits the specified job to some underlying resource management or allocation system. On success, a GATJob instance is returned, which represents the job. Upon failing to submit the job, an error is issued.

Inputs:

GATJobDescription — jobDescription — Description of the job to schedule.

Outputs:

GATJob — job — Representation of the job.

3.30 class GATReservation specializes GATObject

Description

An instance of this class is a reservation for a GATResource.

Operations

Cancel This operation upon successful completion cancels the reservation corresponding to this GATReservation instance.

GetResource This operation returns the GATResource corresponding to this GATReservation instance. That instance can in turn call the operation GetReservation to obtain this GATReservation instance.

Outputs:

GATResource — resource — Resource upon success, undefined otherwise.

4 GAT Application Utility API

This section defines the publicly accessible operations and class constants associated with each descriptor. Note that private data is an implementation and language-specific detail and is thus not part of the API.

4.1 class GATObject

Description

Ancestor of all classes in the GAT API.

Operations

Constructor This no arguments constructor creates an instance of an GATObject.

Destructor Destroys this GATObject.

Equals The Equals operation implements an *equivalence relation* indicating whether some other object is "equal to" this one. It is defined as a relation with the following properties:

- It is *reflexive*, for any reference value x, x.equals(x) should return True.
- It is *symmetric*, for any reference values x and y, x.equals(y) should return True if and only if y.equals(x) returns True.
- It is *transitive*, for any reference values x, y, and z, if x.equals(y) returns True and y.equals(z) returns True, then x.equals(z) should return True.

In addition, for any non-null reference value x, x.equals(null) should return False.

The Equals operation for this class returns True if and only two reference values x and y refer to the same instance in memory, in other words if x == y is True.

4.2 class GATContext specializes GATObject

Description

An instance of this class is the primary GAT state object.

Operations

Constructor This constructor creates an instance of a GATContext.

Destructor Destroys this GATContext object.

AddPreferences The given GATPreferences are used as default preferences if for GATObjects created with this GATContext.
Inputs:

GATPreferences — preferences — Default GATPreferences for GATObjects created with this GATContext.

RemovePreferences Remove the GATPreferences used as default preferences if for GATObjects created with this GATContext.

GetPreferences Return the GATPreferences that are used as default preferences if for GATObjects created with this GATContext.
Outputs:

GATPreferences — preferences — Default GATPreferences for GATObjects created with this GATContext.

Clone Clone is used to clone a specified context, copying all state and security information. The new GATContext is completely independent from the original one, which may be destroyed with no effect on the new one.

Outputs:

GATContext — context — New GATContext.

ServiceActions The ServiceActions call is used to allow the GAT Engine to service asynchronous actions, such as GATRequests and GATMetricEvents. In a single-threaded application it is likely that a timeout would be supplied, in a multi-threaded application one thread may be used for the GAT by using this call and no timeout.

The use of the ServiceActions operation MAY be language specific. For instance, some languages are naturally threaded, and this functionality may be provided by native means. As the timeout SHOULD be honored by all adaptors, the API user SHOULD treat that as a request that the implementation will attempt to honour.
Inputs:

GATTimePeriod — timeout — this may be a 0 timeout to indicate no timeout at all, or a specific time length.

AddSecurityContext Adds the passed GATSecurityContext.

Inputs:

GATSecurityContext — securityContext — Instance to add.

RemoveSecurityContext Removes the passed GATSecurityContext.

Inputs:

GATSecurityContext — securityContext — Instance to remove.

GetSecurityContexts Gets the List of GATSecurityContexts associated with this GATContext.

Outputs:

List of GATSecurityContexts — securityContexts — GATSecurityContexts associated with this GATContext.

GetSecurityContextsByType Gets a List of GATSecurityContexts of the specified type associated with this GATContext.

Inputs:

Integer — type — GATSecurityContext type.

Outputs:

List of GATSecurityContexts — securityContexts — GATSecurityContexts of the specified type associated with this GATContext.

GetStatus Gets the GATStatus instance of the last operation associated with this GATContext. Note that in languages supporting exceptions, this MAY also have been thrown as an exception by that operation.

Outputs:

GATStatus — status — GATStatus of last GAT operation associated with this context.

4.3 class GATSecurityContext specializes GATObject

Description

A container for security information. Each context has a type associated with it. The type indicates if the GATSecurityContext instance corresponds to a "password" GATSecurityContext or a "certificate" GATSecurityContext.

Currently we provide additional auxiliary operations to create a context based upon password information or upon credentials stored in a file. GATContexts based upon these mechanisms can be used by adaptors to create further contexts containing opaque data objects, e.g. GSSAPI credentials.

Operations

Constructor Creates a new security context of a specific type. The type indicates the means by which this instance allows "secure" communications to be established. The allowed values for this type are the various public class variables of this class established for this purpose.

Inputs:

Integer — type — Integer indication of the type of this instance.

Equals Tests this GATSecurityContext for equality with the passed GATObject.

If the given GATObject is not a GATSecurityContext, then this operation immediately returns False.

For two GATSecurityContexts to be considered equal requires that they must be acquired over the same mechanisms and must refer to the same name.

Destructor Destroys a security context.

SetPasswordAuthenticate Makes this a "Password" type security context and stores the username and password in the context.

Inputs:

String — name — Username associated with password.

String — password — Password.

GetPasswordAuthenticate If this is a "Password" type security context get the username and password from the context.

Outputs:

String — name — Username associated with password.

String — password — Password.

SetCertificateAuthenticate Makes this a "Certificate" type security context and stores the information about the location of keyfile and certificate file in the context.

Inputs:

String — keyfile — Keyfile, containing valid absolute or relative local path to keyfile. A relative path will be converted to an absolute path based upon the current working directory.

String — certificate — Certificate, containing valid absolute or local path to certificate file. A relative path will be converted to an absolute path based upon the current working directory.

String — passphrase — Passphrase (OPTIONAL)

GetCertificateAuthenticate If this is a "Certificate" type security context get the information about the location of keyfile and certificate file stored in the context.

Outputs:

String — keyfile — Keyfile, containing valid absolute or relative local path to keyfile. A relative path will be converted to an absolute path based upon the current working directory.

String — certificate — Certificate, containing valid absolute or local path to certificate file. A relative path will be converted to an absolute path based upon the current working directory.

String — passphrase — Passphrase

SetRemoteAuthenticate Makes this a "Remote" type security context and stores the information about the location of remote credential server in the context.

Inputs:

GATLocation — location — Location (URL) for remote credential server.

String — name — Username associated with the credential.

String — passphrase — Passphrase associated with the credential.

GetRemoteAuthenticate If this is a "Remote" type security context get the information about the location of remote credential server stored in the context.

Inputs:

GATLocation — location — Location (URL) for remote credential server.

String — name — Username associated with credential.

String — passphrase — Passphrase associates with credential.

GetType Gets the type associated with this instance.

Outputs:

Integer – Type associated with this instance.

Clone Clone is used to clone a specified GATSecurityContext instance, copying all state and security information. The new GATSecurityContext is completely independent from the original one, which may be destroyed with no effect on the new one. This method is used when cloning a GATContext.

Outputs:

GATSecurityContext — secContext — New GATSecurityContext.

Class Constants

GAT_Password — Integer constant used to specify the type of an instance of this

GAT_Certificate — Integer constant used to specify the type of an instance of this

GAT_Remote — Integer constant used to specify the type of an instance of this class.

4.4 template class GAT<T>CredentialService

Description

Classes binding to specific values of the parameter <T> provide methods to return specific security objects, given an instance of a GATSecurityContext. For example a GATGSICredentialService provides mechanisms to get GSI credentials, a GATSSLCredentialService provides access to an SSL security object.

Operations

Constructor Constructs a GAT<T>CredentialService instance.
Inputs:

GATContext — context — GATContext

GetFullCredentialList Gets a List of credentials of type <T> from the security contexts associated with the GATContext.
Outputs:

List of <T> objects — credential_list — List of credential objects

GetCredentialList Gets a List of credentials of type <T> from a given security context associated with the GATContext. It is an error if passed GATSecurityContext instance is not associated with the GATContext used in the constructor of this GAT<T>CredentialService instance.
Inputs:

SecurityContext — context — A security context to get the <T> credential objects from.

Outputs:

List of <T> objects — credential_list — List of credential objects

4.5 class GATSelf specializes GATObject

Description

This class corresponds to the current GAT job. There is only ever one instance of this class, which is obtained by the GetInstance method. This object can be used to change various properties of this job, such as whether it is checkpointable or not, and what metrics or events it can report. It can also provide the GATJob instance associated with this job, which may then be advertised.

Operations

GetInstance This class level operation returns the GATSelf object. While the GATSelf is not associated with any particular GATContext, this operation requires one to allow implementations to maintain thread-safety.
Inputs:

GATContext — context — GATContext for thread safety.

Outputs:

GATSelf — self — The GATSelf instance

SetExecutionEnvironment This class level operation announces the execution environment to the GAT Engine. The Engine can hence react on the given command line arguments and environemnt. The need for that method arises, amongst others, for the correct instanciation of the GATJob object representing this program instance (see GetJob method).
Inputs:

Integer — argc — number of command line arguments

List of Strings — argv — command line arguments

String — path — complete path to executable

List of Strings — environment — ”KEY=VALUE” formated environment strings

AddRequestListener Add a listener for specific GATRequests. If this is an information request listener, an application monitoring this application may see this as a new entry in the list of available metrics. If this is a command request, it must be “checkpoint”, and this application will now appear as checkpointable.
Inputs:

GATRequestListener — listener — an objectrealizing the GATRequestListener interface.

Integer — type — corresponding to the type of GATRequest served by this listener — command or information.

GATTable — Parameters — contains further information describing this request type. In the case of an information request this must provide the “Metric Parameters”, “Metric measurement type”, “Metric data type” and “Metric unit” as detailed in the GATMetric class.

String — name — Name for the GATRequest.

RemoveRequestListener Remove the request listener. If it was an information request listener, it will no longer be available to monitoring clients. If it was a checkpoint command listener the application will no longer be marked as checkpointable.
Inputs:

String — name — Name for the GATRequestListener to be removed.

GetJob Gets a GATJob instance which is associated with this job. This can then be advertised to allow other jobs to manipulate this one.
Inputs:

GATContext — context — GATContext to be used for that GATJob.

Outputs:

GATJob — job — a GATJob associated with this job.

Class Constants

GAT_CommandRequest — Integer constant used to specify that a GATRequestListener is for command requests.

GAT_InformationRequest — Integer constant used to specify that a GATRequestListener is for information requests.

4.6 class GATLocation specializes GATObject

Description

An instance of this class represents the location of an abstract or physical resource. The location of an abstract or physical resource is represented by a URI as defined by the standards

- RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax
- RFC 2732: Format for Literal IPv6 Addresses in URLs.

One should refer to these standards to determine the allowed forms for URIs. This class provides a means to create a GATLocation instance from a "URI in String form," operations for accessing the various components of the contained URI, and various other utility operations.

Operations

Constructor Constructs a GATLocation instance by parsing the given String as a URI. This constructor parses the given String exactly as specified by the grammar in RFC 2396, Appendix A, except IPv6 addresses are permitted for the host component. An IPv6 address must be enclosed in square brackets ('[' and ']') as specified by RFC 2732. The IPv6 address itself must parse according to RFC 2373. IPv6 addresses are further constrained to describe no more than sixteen bytes of address information, a constraint implicit in RFC 2373 but not expressible in the grammar.

Inputs:

String — uri — URI for the Location.

Equals Tests this GATLocation for equality with the passed GATObject.

If the given GATObject is not a GATLocation, then this operation immediately returns False.

For two GATLocations to be considered equal requires that either both are opaque or both are hierarchical. Their schemes must either both be undefined or else be equal without regard to case, and similarly for their fragments.

For two opaque GATLocations to be considered equal, their scheme-specific parts must be equal.

For two hierarchical GATLocations to be considered equal, their paths must be equal and their queries must either both be undefined or else be equal. Their authorities must either both be undefined, or both be registry-based, or both be server-based. If their authorities are defined and are registry-based, then they must be equal. If their authorities are defined and are server-based, then their hosts must be equal without regard to case, their port numbers must be equal, and their user-information components must be equal.

When testing the user-information, path, query, fragment, authority, or scheme-specific parts of two GATLocations for equality, the raw forms rather than the encoded forms of these components are compared and the hexadecimal digits of escaped octets are compared without regard to case.

Destructor Destroys this GATLocation object.

GetAuthority Returns the decoded authority component of this GATLocation.
A sequence of escaped octets is decoded by replacing it with the sequence of characters that it represents in the UTF-8 character set. UTF-8 contains US-ASCII, hence decoding has the effect of de-quoting any quoted US-ASCII characters as well as that of decoding any encoded non-US-ASCII characters. If a decoding error occurs when decoding the escaped octets then the erroneous octets are replaced by '\\uFFFD', the Unicode replacement character.
The String returned by this operation is equal to that returned by the GetRawAuthority operation except that all sequences of escaped octets are decoded.

Outputs:

String — authority — Decoded authority component of this GATLocation, or null if authority is undefined.

GetRawAuthority Returns the raw authority component of this GATLocation.
The authority component of a GATLocation, if defined, only contains the commercial-at character ('@') and characters in the unreserved, punct, escaped, and other categories. If the authority is server-based then it is further constrained to have valid user-information, host, and port components.

Outputs:

String — rawAuthority — Raw authority component of this GATLocation, or null if authority is undefined.

GetFragment Returns the decoded fragment component of this GATLocation.
A sequence of escaped octets is decoded by replacing it with the sequence of characters that it represents in the UTF-8 character set. UTF-8 contains US-ASCII, hence decoding has the effect of de-quoting any quoted US-ASCII characters as well as that of decoding any encoded non-US-ASCII characters. If a decoding error occurs when decoding the escaped octets then the erroneous octets are replaced by '\\uFFFD', the Unicode replacement character.
The String returned by this operation is equal to that returned by the GetRawFragment operation except that all sequences of escaped octets are decoded.

Outputs:

String — ragment — The decoded fragment component of this GATLocation, or null if fragment is undefined.

GetRawFragment Returns the raw fragment component of this GATLocation.
The fragment component of a GATLocation, if defined, only contains legal URI characters.

Outputs:

String — rawFragment — Raw fragment component of this GATLocation, or null if fragment is undefined.

GetHost Returns the host component of this GATLocation.
The host component of a GATLocation, if defined, will have one of the following forms:

- A domain name consisting of one or more labels separated by period characters ('.'), optionally followed by a period character. Each label consists of alphanum characters as well as hyphen characters ('-'), though hyphens never occur as the first or last characters in a label. The last, or only, label in a domain name begins with an alpha character.
- A dotted-quad IPv4 address of the form digit+.digit+.digit+.digit+, where no digit sequence is longer than three characters and no sequence has a value larger than 255.
- An IPv6 address enclosed in square brackets ('[' and ']') and consisting of hexadecimal digits, colon characters (':'), and possibly an embedded IPv4 address. The full syntax of IPv6 addresses is specified in RFC 2373: IPv6 Addressing Architecture.

The host component of a GATLocation cannot contain escaped octets, hence this operation does not perform any decoding.

Outputs:

String — host — Host component of this GATLocation, or null if host is undefined.

GetPath Returns the decoded path component of this GATLocation.
A sequence of escaped octets is decoded by replacing it with the sequence of characters that it represents in the UTF-8 character set. UTF-8 contains US-ASCII, hence decoding has the effect of de-quoting any quoted US-ASCII characters as well as that of decoding any encoded non-US-ASCII characters. If a decoding error occurs when decoding the escaped octets then the erroneous octets are replaced by `'\\uFFFD'`, the Unicode replacement character.
The String returned by this operation is equal to that returned by the GetRawPath operation except that all sequences of escaped octets are decoded.

Outputs:

String — path — Decoded path component of this GATLocation, or null if path is undefined.

GetRawPath Returns the raw path component of this GATLocation.
The path component of a URI, if defined, only contains the slash character ('/'), the commercial-at character ('@'), and characters in the unreserved, punct, escaped, and other categories.

Outputs:

String — rawPath — Raw path component of this GATLocation, or null if path is undefined.

GetPort Returns the port number of this GATLocation.
The port component of a URI, if defined, is a non-negative integer.

Outputs:

Integer — port — Port component of this URI, or -1 if the port is undefined.

GetQuery Returns the decoded query component of this GATLocation.
A sequence of escaped octets is decoded by replacing it with the sequence of characters that it represents in the UTF-8 character set. UTF-8 contains US-ASCII, hence decoding has the effect of de-quoting any quoted US-ASCII characters as well as that of decoding any encoded non-US-ASCII characters. If a decoding error occurs when decoding the escaped octets then the erroneous octets are replaced by `'\\uFFFD'`, the Unicode replacement character.
The String returned by this operation is equal to that returned by the GetRawQuery operation except that all sequences of escaped octets are decoded.

Outputs:

String — query — Decoded query component of this GATLocation, or null if query is undefined.

GetRawQuery Returns the raw query component of this GATLocation.
The query component of a URI, if defined, only contains legal URI characters.

Outputs:

String — rawQuery — Raw query component of this GATLocation, or null if query is undefined.

GetScheme Returns the scheme component of this GATLocation.
The scheme component of a URI, if defined, only contains characters in the alphanum category and in the String "-.+". A scheme always starts with an alpha character.
The scheme component of a URI cannot contain escaped octets, hence this operation does not perform any decoding.

Outputs:

String — scheme — Scheme component of this GATLocation, or null if scheme is undefined.

GetSchemeSpecificPart Returns the decoded scheme-specific part of this GATLocation.
A sequence of escaped octets is decoded by replacing it with the sequence of characters that it represents in the UTF-8 character set. UTF-8 contains US-ASCII, hence decoding has the effect of de-quoting any quoted US-ASCII characters as well as that of decoding any encoded non-US-ASCII characters. If a decoding error occurs when decoding the escaped octets then the erroneous octets are replaced by `'\\uFFFD'`, the Unicode replacement character.
The String returned by this operation is equal to that returned by the GetRawSchemeSpecific-Part operation except that all sequences of escaped octets are decoded.

Outputs:

String — Decoded scheme-specific component of this GATLocation (never null),

GetRawSchemeSpecificPart Returns the raw scheme-specific part of this GATLocation.
The scheme-specific part is never undefined, though it may be empty.
The scheme-specific part of a URI only contains legal URI characters.

Outputs:

String — Raw scheme-specific component of this GATLocation (never null).

GetUserInfo Returns the decoded user-information component of this GATLocation.
A sequence of escaped octets is decoded by replacing it with the sequence of characters that it represents in the UTF-8 character set. UTF-8 contains US-ASCII, hence decoding has the effect of de-quoting any quoted US-ASCII characters as well as that of decoding any encoded non-US-ASCII characters. If a decoding error occurs when decoding the escaped octets then the erroneous octets are replaced by `'\\uFFFD'`, the Unicode replacement character.
The String returned by this operation is equal to that returned by the GetRawUserInfo operation except that all sequences of escaped octets are decoded.

Outputs:

String — Decoded user-information component of this GATLocation, or null if it is undefined.

GetRawUserInfo Returns the raw user-information component of this GATLocation.
The user-information component of a URI, if defined, only contains characters in the unreserved, punct, escaped, and other categories.

Outputs:

String — Raw user-information component of this GATLocation, or null if it is undefined.

ToString Returns the content of this GATLocation as a String.
A String equivalent to the input string given to the GATLocation constructor, or to the String computed from the originally-given components, as appropriate, is returned.

Outputs:

String — stringLocation — The string form of this GATLocation.

Clone Returns a deep clone of this GATLocation.

Outputs:

GATLocation — clone — A clone of this GATLocation.

4.7 class GATPreferences specializes GATObject

Description

An instance of this class represents the user's preferences for selecting adaptors. Currently this class is a place holder for the user preferences, the structure of which is in development; the matching algorithm outlined in the Match operation is used by the GAT implementation to determine which capability provider is used to satisfy a GAT API operation.

Operations

Constructor Creates a new GATPreferences instance.

Destructor Destroys this GATPreferences instance.

Add This adds a name/value pair in which the name is a String and the value is a String to this GATPreferences instance.

Inputs:

String — name — Name of the attribute to add.

String — value — Value of the attribute to add.

Remove Removes the name/value pair with the passed name from this GATPreferences instance.

Inputs:

String — name — Name of the name/value to remove.

Set Sets the given GATTable of name/value pairs as preferences.

Input:

GATTable — preferences — Table of name/value pairs to be used as preferences.

Get Gets the current preferences as name/value pairs in a GATTable
Output:

GATTable — preferences — Table of name/value pairs containing the current preferences.

Match Matches the GATPreferences instance against another GATPreferences instance expressing criteria. For Match to return true, all keys present in the criteria table must be present in the original and must match: a string-valued value in the original table is matched by a regular expression in the criteria table, and a numeric-valued key in the original table by a string holding an arithmetical expression (e.g., "< 5") in the criteria table.
Inputs:

GATPreferences — criteria — Matching criteria.

Outputs:

Bool — match — True if the GATPreferences instance matches the given criteria.

Clone Clone is used to clone this GATPreferences instance, copying the GATTable. The new GATPreferences instance is completely independent from the original one, which may be destroyed with no effect on the new one. This method is used when cloning a GATContext.

Outputs:

GATPreferences — preferences — Cloned GATPreferences.

4.8 class GATStatus specializes GATObject

Description

An instance of this class represents an error or an information message from a GAT operation or from an underlying adaptor. instances of this class are used to provide an audit trail which the application user can use to trace the sequence of events which happened in any particular GAT operation; this may then be used by the application, adaptor or service developers or providers to debug problems.
Since the GAT Engine and adaptors may do several independent operations each of which may have associated errors or status messages, a GATStatus instance forms a node in a tree of GATStatus instances, rather than the more normal parent-child process of a try-catch type error mechanism.

Use cases

The application discovers that the last GAT operation had an error. It gets the status object, and tracks through the tree of child-errors printing out information to the user indented as per what depth it has in the tree.

Application invokes GATContext.GetStatus to get the GATStatus object, S, from the last GAT operation.

Application gets the messages associated with S by invoking S.GetMessages, and prints them out.

Application gets the status code associated with S by invoking S.GetStatusCode and prints that out.

Application invokes S.GetChildren and, for each child, C, invokes C.GetMessages, C.GetStatusCode and C.GetChildren as above, indenting the messages depending on depth in the tree.

Operations

Constructor Constructs an instance of this class with the passed message.
Inputs:

String — message — Message associated with this GATStatus.

Destructor Destroys this GATStatus instance.

SetStatusCode Sets the status code of this GATStatus.

Inputs:

Integer — code — Status code for this GATStatus.

GetStatusCode Gets the status code of this GATStatus.

Outputs:

Integer — code — Status code for this GATStatus.

AddChild Adds a child GATStatus instance to this one.

Inputs:

GATStatus — child — Child GATStatus instance.

GetChildren Gets the child GATStatus instances of this one.

Outputs:

List of GATStatus objects — children — Child GATStatus instances.

AddMessage Adds a message to this GATStatus.

Outputs:

String — message — Message for this GATStatus.

GetMessages Returns the List of messages associated with this GATStatus.

Outputs:

List of Strings — messages — Messages associated with this GATStatus.

GetParent Returns the parent GATStatus of this GATStatus.

Outputs:

GATStatus — parent — Parent GATStatus of this GATStatus.

4.9 class GATTime specializes GATObject

Description

An instance of this class represents a point in time.

Operations

Constructor This operation constructs a GATTime instance corresponding to the passed time.

Inputs:

Integer — time — Number of milliseconds after January 1, 1970, 00:00:00 GMT.

Destructor Destroys this GATTime instance.

Equals Tests this GATTime for equality with the passed GATObject.

If the given GATObject is not a GATTime, then this operation immediately returns False.

If the passed GATObject is a GATTime, then it is deemed equal if it has a numerically equivalent time to the passed GATTime instance.

GetTime This operation returns the time as the number of milliseconds after January 1, 1970, 00:00:00 GMT, an Integer

Outputs:

Integer — duration — Number of milliseconds after January 1, 1970, 00:00:00 GMT.

4.10 class GATTimePeriod specializes GATObject

Description

An instance of this class represents a time duration, a length of time with uncertain start point.

Operations

Constructor This operation constructs a GATTimePeriod instance corresponding to the passed duration.

Inputs:

Integer — duration — Number of milliseconds this period of time lasts.

Destructor Destroys this GATTimePeriod.

Equals Tests this GATTimePeriod for equality with the passed GATObject.

If the given GATObject is not a GATTimePeriod, then this operation immediately returns False.

If the passed GATObject is a GATTimePeriod, then it is deemed equal if it has a numerically equivalent time duration to the passed GATTimePeriod instance.

GetDuration This operation returns the number of milliseconds this time period lasts, an Integer

Outputs:

Integer — duration — Number of milliseconds this time period lasts.

4.11 class GATTable

Description

An instance of the GATTable class maps keys to values. Any non-null instance of String can be used as a key, any String, primitive type (such as Integers and Floats) or GATObject instance can be used as a values. We call these "valid types" in this section. Apart from GATObjects and Strings, the other valid types are language dependend, and will be defined in the language specific GAT API specifications.

Note: For some languages, native equivalents of GATTables may exist (e.g. hashtables in Perl). For specific language bindings, these native equivalents MAY be used instead of the GATTable class.

Operations

Constructor This operation constructs an instance of the class GATTable.

Equals Tests this GATTable for equality with the passed Object.

If the given Object is not a GATTable, then this operation immediately returns False.

For two GATTable instances to be considered as equal they must have a map from the set of keys in the first GATTable to the set of keys in the second GATTable such that Equals() when evaluated on the pairs of keys generated by this map returns True. In addition, the values for the key pairs generated by this map must be equal as determined by the Equals() operation on each value. In addition the same must be True exchanging the roles of the passed GATTable and the called GATTable.

Destructor Destroys this GATTable.

Add Maps the specified key to the specified value in this GATTable.

Inputs:

String — key — Key to add to the GATTable instance.

Valid Type — value — Value to map to the passed key in the GATTable instance.

Get Returns the value to which the specified key is mapped in this GATTable.

Inputs:

String — key — Key to get the value for in this GATTable instance.

Outputs:

Valid Type — value — Value to which the key is mapped in this GATTable instance.

Remove Removes the specified key is from this GATTable.

Inputs:

String — key — Key to get the value for in this GATTable instance.

GetType Returns the data tyoe for the value of the specified key is maped in this GATTable. The realization of that operation MAY be different, dependend on implementation language.
Inputs:

String — key — Key to get the value type for in this GATTable instance.

Outputs:

Integer — type — Class constant describing the value type to which the key is mapped in this GATTable instance.

GetKeys Returns a list of keys from GATTable.

Outputs:

List of Strings — keys — List containing all keys used in that GATTable instance.

A External Classes

The present document relies on various occasions on classes which are not GATObjects. The realisation and implementation of these types is in general language dependent, and in many cases can be reduced to native classes. This appendix describes the *minimal* functionality the GAT expects from these types. When in some language an equivalent of such classes does not exist, GAT has to implement it.
For some of the listed classes, like Buffer, a functionally extended equivalent MAY be added as GAT*Class* (e.g. GATBuffer) in a later version of the specification.

A.1 class List
specializes GATObject
realizes GATAdvertiseable

Description

An instance of this class is a list of instances. Some languages have "native" List types which MAY be used. If no such native List type exists, then a GAT implementation must include a List type.

Equals Tests this List for equality with the passed instance.

If the passed instance is not a List instance, then this operation immediately returns False.

If the passed instance is a List instance, then it is deemed equal to this instance if it consists of the "same" elements, as determined by the Equals operation on the respective elements, in the same order as this List instance.

A.2 class String specializes GATObject realizes GATAdvertiseable

Description

An instance of the String class represents a character string. The class String provides various operation for manipulation of String instances.

Operations

Constructor Constructs a new String by decoding the specified List of bytes using the specified charset. The length of the new String is a function of the charset, and hence MAY not be equal to the length of the byte List. The behaviour of this constructor when the given bytes are not valid in the given charset is unspecified.
The currently supported charsets are as follows

- **US-ASCII** — Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set
- **ISO-8859-1** — ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1
- **UTF-8** — Eight-bit UCS Transformation Format
- **UTF-16BE** — Sixteen-bit UCS Transformation Format, big-endian byte order
- **UTF-16LE** — Sixteen-bit UCS Transformation Format, little-endian byte order
- **UTF-16** — Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark

Inputs:

Buffer — buffer — The bytes to be decoded into characters.

String — charset — Name of a supported charset.

Equals Compares this string to the specified object. The result is True if and only if the argument is not null and is a String object that represents the same sequence of characters as this object

Destructor Destroys this String object.

GetBytes Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte List. The behaviour of this operation when this string cannot be encoded in the given charset is unspecified.

Inputs:

String — charset — Charset to use for conversion.

Outputs:

Buffer — buffer — Resultant bytes.

A.3 class Buffer realizes GATAdvertiseable

Description

A buffer is a container for arbitrary data. In most languages, this can be represented as an array of bytes. The application is responsible for reading and writing such buffers, and in particular for correct (de)serialisation of primitive and complex types during that process.

The GAT API specification MAY in a later version be extended with a GATBuffer class providing more sophisticated handling of typed data.

Thread safety Buffers need not to be safe for use by multiple concurrent threads. If a buffer is to be used by more than one thread then access to the buffer SHOULD be controlled by appropriate synchronisation.

B Glossary

advertisement — A public notice. An entry in the Advert Directory.

advertise — To make publicly and generally known.

ancestor — An element found by following a path of one or more parent relationships.

attribute — A description of a named slot of a specified type in a class; each object of the class separately holds a value of the type.

binding — The assignment of values to parameters to produce an individual element from a parameterised element.

class— The descriptor for a set of objects that share the same attributes, operations, methods, relationships, and behaviour.

component— A physical replaceable part of a system that packages implementation and conforms to and provides a realization of a set of interfaces.

connection — A bi-directional communication channel.

constructor — A class-scope operation that creates and initialises an instance of a class.

descriptor — A model element that describes the common properties of a set of instances, including their structure, relationships, behaviour, constraints, purpose, and so on.

destructor — A class-scope operation that destroys an instance of a class.

instance— An individual entity with its own identity and value. A descriptor specifies the form and behaviour of a set of instances with similar properties. An instance has identity and values that are consistent with the specification in the descriptor.

interface — A named set of operations that characterise the behaviour of an element.

model element — An element that is an abstraction drawn from the system being modelled.

node— A run-time physical object that represents a computational resource, which generally has at least memory and often processing capability.

null — null defines in a language and type dependent way the absence of a value or object.

object — A discrete entity with a well-defined boundary and identity that encapsulates state and behaviour.

operation— A specification of a transformation or query that an object may be called to execute.

private — A visibility value indicating that the given element is not visible outside its own namespace even to descendants of the namespace.

process— A heavyweight unit of concurrency and execution in an operating system.

public — A visibility value indicating that the given element is visible outside its own namespace.

resource — An entity providing some capability. Example for resources are computers (providing compute power), networks (providing data transport capabilities), services (providing capabilities specific to the service). Resource Management Systems manage (discover, query, reserve, schedule, utilize, ...) one or more types of resources, but typically not all types simultaneously.

realize — To provide the implementation for a specification element.

specialization — To produce a more specific description of a model element by adding children.

state — A condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

stream — A uni-directional communication channel.

template — A parameterised model element. To use it, the parameters must be bound to actual values.

References

[1] K. Davis and T. Goodale, "GAT API Specification", ID: GridLab-1-GAS-0003-0.1DRAFT.

[2] K. Davis and T. Goodale, "D1.2 Technical Specification" ID: Gridlab-1-D1.2-0002.TechnicalSpecification.

[3] "Grid Resource Management System", http://www.gridlab.org/WorkPackages/wp-9/index.html.

[4] S.Bradner, "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels," http://www.ietf.org/rfc/rfc2119.txt.

[5] J.Rumbaugh, I. Jacobson, G. Booch, "The Unified Modeling Language Reference Manual", Addison-Wesley, Reading, Massachusetts, 1999.

[6] http://www.triana.com.

[7] Jeffrey, E. F. Friedl, "Mastering Regular Expressions", O'Reilly & Associates; 2nd edition (July 15, 2002).

[8] Henry Spencer, "regex — POSIX 1003.2 regular expressions", Unix Manpages, chapter 7.