



IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

D1.2 Technical Specification

Author(s):	Kelly Davis, Tom Goodale
Document Filename:	Gridlab-1-D1.2-0002.TechnicalSpecification
Work package:	Grid Application Toolkit
Partner(s):	PSNC,ZIB,MU,SZTAKI,VU,ISUFI,CARDIFF,GRIDWARE, COMPAQ,NTUA
Lead Partner:	MPG
Config ID:	GridLab-1-D.1.2-0002-1.0
Document classification:	Internal

Abstract: This document presents a rough draft technical specification for the GAT. It describes the architecture of the software components making up a GAT and presents plans for its implementation.



Contents

1	Introduction	3
1.1	Purpose of Document	3
1.2	History	3
1.3	Structure of Document	3
1.4	Status of this Document	3
1.5	RFC 2119 and this Document	3
2	Requirements	3
2.1	Requirements	4
2.1.1	WP1	4
2.1.2	WP2	7
2.1.3	WP3	7
2.1.4	WP4	8
2.1.5	WP5	8
2.1.6	WP6	8
2.1.7	WP7	9
2.1.8	WP8	9
2.1.9	WP9	9
2.1.10	WP10	10
2.1.11	WP11	10
2.1.12	WP12	10
2.1.13	WP13	10
2.1.14	WP14	10
3	Architecture of the Grid Application Toolkit	10
3.1	GAT Engine	11
3.1.1	API function bindings	11
3.1.2	Capability Registry	12
3.1.3	Property Tables	13
3.2	Adaptors	13
3.2.1	Initialisation and Registration Mechanisms	13
3.2.2	Function Bindings	13
3.3	GAT Operation	14
3.3.1	Initialisation	14
3.3.2	API Function Invocation	14
3.4	Other Architecture Issues	16
3.4.1	GAT Contexts	16
3.4.2	Security	16
4	Technology Survey	16
4.1	C Implementation	16
4.1.1	Standards and tools	16
4.1.2	Adaptor Discovery and Loading	17
4.1.3	Property Tables	17
4.2	Java Binding	17
4.3	Perl Binding	17
4.4	Python Binding	18

4.5	Fortran Binding	18
4.6	Adaptor Technologies	18
4.6.1	Globus	18
4.6.2	GSI	18
4.6.3	WSDL, WSIL	18
4.6.4	UDDI	18
4.6.5	OGSA	19
A	Appendix: Glossary	20

1 Introduction

1.1 Purpose of Document

This document is a rough outline of the technical specification of the GAT. It presents a high-level view of the architecture of the GAT and then presents some details of the implementation. This document is not a complete design document. The full design will be published either in updates to this document or in a separate document.

1.2 History

The design presented in this document is a refinement of that developed during a two day meeting between Gabrielle Allen, Tom Goodale, Jarek Nabrzyski, Jason Novotny, Michael Russell, John Shalf, Ed Seidel, and Ian Taylor in San Francisco in October 2001. Rough slides from this meeting were presented at the pre-kickoff meeting in Frascati [1].

1.3 Structure of Document

This document is split into three main sections.

Section 2 gathers all the requirements from work-packages which are relevant to WP1 into one place.

Section 3 gives a high level view of the GAT architecture, describing the major subdivisions of the GAT and how they inter-operate with each other and with applications.

Section 4 provides a brief survey of some of the technologies which may be used to realise the design. It covers the major issues involved in the canonical C implementation, and brief notes as to how bindings to other languages may be achieved.

1.4 Status of this Document

This document is a working draft version. As a working draft, this specification may be updated, replaced, or made obsolete at any time. It is distributed for discussion purposes only, and should not be used as a reference. Readers are encouraged to send comments to the WP1 mailing list.

1.5 RFC 2119 and this Document

The key words “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 [2].

2 Requirements

This section attempts to to achieve the following goals:

1. *Gather* all requirements GAT and the GAT-API must satisfy.
2. *Clarify* all requirements GAT and the GAT-API must satisfy.

Gathering the requirements for GAT and the GAT-API is critical in that currently all of the various requirements that GAT and the GAT-API must satisfy are spread across a plethora of

documents and are expressed using a multitude of differing vocabularies.

Clarifying these requirements is a natural “next step.” Initially, when dispersed across the diaspora of documents, the requirements were expressed using differing, often conflicting, terms which were at times vague and ill-defined. So, the process of *gathering* the requirements presented, and presents, an opportunity for all requirements to be expressed in a common language where all terms are, hopefully, well-defined.

2.1 Requirements

This subsection *gathers* all of the requirements placed upon GAT and the GAT-API. These requirements are grouped by work package. The requirements a given work package places on GAT and the GAT-API are listed under a section with the title of the work package. Also, the original requirement identifier is listed next to the new requirement identifier, **WPX.Y**. WP1 suggests that in all future documentation requirements placed on WP1 use the new naming scheme. Beyond the above, effort has been made to *clarify* all of the requirements placed on GAT and the GAT-API.

2.1.1 WP1

These are the set of requirements which were *gathered* as part of D1.1. These requirements have been grouped as in D1.1 into general requirements, developer requirements, and GAT requirements. In addition, the requirements have been grouped into “non-quantifiable requirements” and “quantifiable requirements.” Quantifiable requirements are those whose presence or absence can be quantified in a binary manner. Non-quantifiable requirements are requirements that are not quantifiable.

1. General Requirements

(a) Non-Quantifiable Requirements

- **WP1.1**(RU-1) GAT and the GAT-API SHOULD “facilitate” the construction of **applications**.
- **WP1.2**(RU-2) GAT and the GAT-API SHOULD allow **applications** to employ the **resources** of one or more **virtual organisations**.
- **WP1.3**(RU-3) GAT and the GAT-API SHOULD supply “flexible,” “easy-to-use,” and “simple” interfaces to **resources**.
- **WP1.5**(RU-9) GAT and the GAT-API SHOULD allow **applications** to run on computers which have “minimal” computational **resources**.
- **WP1.6**(RU-10) GAT SHOULD be “easily” deployable.
- **WP1.7**(RU-11) GAT and the GAT-API SHOULD “hide” any “implementation complexity” from the **application**.
- **WP1.8**(RU-11) GAT and the GAT-API SHOULD “hide” any “implementation complexity” from the **application programmer**.
- **WP1.9**(RU-12) GAT and the GAT-API MUST consist of “well” documented code.
- **WP1.10**(RU-12) GAT and the GAT-API MUST have “good” user training documentation.

- **WP1.11**(RU-12) GAT and the GAT-API **MUST** have “many” code examples illustrating the use of GAT and the GAT-API.
- **WP1.12**(RU-13) GAT and the GAT-API **MUST** have interfaces which are “informative” and “transparent.”
- **WP1.13**(RU-15) GAT and the GAT-API **SHOULD** facilitate a “collaborative infrastructure.”
- **WP1.14**(RU-16) GAT and the GAT-API **SHOULD** allow the **application programmer** to delegate “absolute control” to GAT through the GAT-API.
- **WP1.15**(RU-16) GAT and the GAT-API **SHOULD** allow the **application programmer** to maintain “absolute control” of the operations of the GAT through the GAT-API.
- **WP1.16**(RU-17) GAT and the GAT-API **MUST** allow for applications which require “abundant” **resources** as well as application which require “minimal” **resources**.
- **WP1.17**(RU-18) GAT and the GAT-API **MUST** be “extensible” and “future proof.”

(b) Quantifiable Requirements

- **WP1.18**(RU-2) GAT and the GAT-API **SHOULD** enable **applications** to employ **resources** which are behind, relative to the **application**, a **firewall**, where the relevant security criteria can be met.
- **WP1.19**(RU-2) GAT and the GAT-API **SHOULD** enable **applications** to employ **resources** independently of the architecture on which the **resource** is deployed.
- **WP1.20**(RU-5) GAT and the GAT-API **SHOULD** supply the guarantee that **applications** are running in an environment with a **trust level** greater than or equal to that specified by the user.
- **WP1.21**(RU-5) GAT and the GAT-API **MUST** supply the guarantee that **applications** are running in an environment with a **security level** greater than or equal to that specified by the user.
- **WP1.22**(RU-6) GAT and the GAT-API **MUST** be **fault tolerant**.
- **WP1.23**(RU-7) GAT and the GAT-API **MUST** allow **applications** to run on a computer with no network access.
- **WP1.24**(RU-8) GAT and the GAT-API **MUST** allow **applications** to run on a **mobile computer**.
- **WP1.25**(RU-13) GAT and the GAT-API **MUST** have interfaces which provide clear error messages.
- **WP1.26**(RU-13) GAT and the GAT-API **MUST** have interfaces which provide clear status messages.
- **WP1.27**(RU-2) GAT and the GAT-API **MUST** be able to employ **resources** beyond those which are part of GridLab.
- **WP1.28**(RUO-1) GAT **MAY** be open source.
- **WP1.29**(RUO-2) GAT and the GAT-API **MAY** be localisable.

2. Developer Requirements

(a) Non-Quantifiable Requirements

- **WP1.30**(RAD-3) GAT and the GAT-API **MUST** enable “easy” access to the GridLab testbed.
- **WP1.30**(RAD-3) GAT and the GAT-API **SHOULD** enable “easy” access to resources.
- **WP1.31**(RAD-4) GAT and the GAT-API **SHOULD** enable “fast” prototyping of **applications**.
- **WP1.32**(RAD-4) GAT and the GAT-API **SHOULD** enable “fast” testing of **applications**.
- **WP1.33**(RAD-4) GAT and the GAT-API **SHOULD** allow “fast” prototyping of **resources**.
- **WP1.34**(RAD-4) GAT and the GAT-API **SHOULD** allow “fast” testing of **resources**.
- **WP1.35**(RAD-4) GAT and the GAT-API **SHOULD** allow “fast” use of **resources**.
- **WP1.36**(RAD-7) GAT and the GAT-API **SHOULD** allow an **application programmer** to “easily” modify legacy programs to use GAT and the GAT-API so as to become **applications**.
- **WP1.37**(RAD-6) GAT and the GAT-API **SHOULD** provide an **application programmer** with a means to work as independently as possible from current **resource deployment**.
- **WP1.38**(RSD-6) GAT and the GAT-API **SHOULD** provide an **capability programmer** with a means to work as independently as possible from current **resource deployment**.

(b) Quantifiable Requirements

- **WP1.39**(RAD-5) GAT and the GAT-API **MUST** provide documentation which indicates how an **application programmer** can add checkpointing to an **application**.
- **WP1.40**(RAD-5) GAT and the GAT-API **MUST** provide documentation which indicates how an **application programmer** can write a portable **application**.
- **WP1.41**(RAD-5) GAT and the GAT-API **MUST** provide documentation which indicates how an **application programmer** can write a **fault tolerant application**.

3. GAT Requirements

(a) Non-Quantifiable Requirements

- **WP1.42**(RGAT-1) GAT and the GAT-API **SHOULD** be as portable as is possible.
- **WP1.43**(RGAT-2) GAT and the GAT-API **SHOULD** be “easy” to build.
- **WP1.44**(RGAT-5) GAT and the GAT-API **SHOULD** be “consistent,” i.e. naming schemes, function calls, language bindings,...
- **WP1.45**(RGAT-6) GAT and the GAT-API **SHOULD** be “lightweight.”
- **WP1.46**(RGAT-7) GAT and the GAT-API **SHOULD** have an API for all “major” programming languages.

(b) Quantifiable Requirements

- **WP1.47** GAT and the GAT-API **MUST** be deployable on the GridLab testbed.

- **WP1.48**(RGAT-3) GAT and the GAT-API MUST be thread safe.
- **WP1.49**(RGAT-8) GAT and the GAT-API MUST supply error, diagnostic, logging, and other reporting capabilities.
- **WP1.50**(RGAT-11) GAT and the GAT-API MUST supply test suites to guarantee correct GAT functioning.
- **WP1.51**(RGAT-12) GAT and the GAT-API MUST make use of and enforce a **security policy**.
- **WP1.52**(RGAT-13) GAT and the GAT-API MUST allow for the use of multiple **credentials**.
- **WP1.53**(RGAT-16) GAT and the GAT-API MUST provide extensibility with a dynamic **pluggable architecture**.
- **WP1.54**(RGAT-18) GAT and the GAT-API MUST provide the ability to discover and use **capability providers** at runtime.
- **WP1.55**(RGAT-20) GAT and the GAT-API MUST provide an audit trail facility.

2.1.2 WP2

WP2's requirements will be enumerated by examining the forthcoming scenarios document.

2.1.3 WP3

These are the set of requirements which were *gathered* as part of D3.1. As in D3.1, the requirements have been grouped into **application user** requirements and **application programmer** requirements. Also, the requirements have been grouped further into “non-quantifiable requirements” and “quantifiable requirements.”

1. Application User Requirements

(a) Non-Quantifiable Requirements

- **WP3.1**(4.1.1.7) GAT and the GAT-API MUST provide a **resource** “description language” which is “sufficiently expressive” so as to allow for “fine-grained” **resource discovery**.
- **WP3.2**(4.1.1.13) GAT and the GAT-API MUST provide a means for an **application** to obtain “all required information” relating to a **hardware resource**.

(b) Quantifiable Requirements

- **WP3.3**(4.1.1.2) GAT and the GAT-API MUST provide a Java interface to all functionality.
- **WP3.4**(4.1.1.3) GAT and the GAT-API MUST provide a means for an **application** to identify itself in a manner consistent with the **trust level** and **security level** required by the **application**.
- **WP3.5**(4.1.1.6) GAT and the GAT-API MUST provide a means for an **application** to engage in **resource reservation**.
- **WP3.6**(4.1.1.8) GAT and the GAT-API MUST provide a means for an **application** to start a **software resource**.
- **WP3.7**(4.1.1.9) GAT and the GAT-API MUST provide a means for an **application** to stop a **software resource**.

- **WP3.8**(4.1.1.10) GAT and the GAT-API MUST provide a means for an **application** to suspend a **software resource**.
- **WP3.9**(4.1.1.10) GAT and the GAT-API MUST provide a means for an **application** to resume a **software resource**.
- **WP3.10**(4.1.1.11) GAT and the GAT-API MUST provide a means for an **application** to subscribe to and receive notification about changes in the “abstract state” of a **resource**.
- **WP3.11**(4.1.1.12) GAT and the GAT-API MUST provide a means for an **application** to obtain a description of the **hardware resources** required for a given **software resource** to process a given set of data.
- **WP3.12**(4.1.1.15) GAT and the GAT-API MUST provide a means for an **application** to migrate a currently running **software resource** from its current host **hardware resource** to different **hardware resource**.
- **WP3.13**(4.1.1.15) GAT and the GAT-API MUST provide a means for an **application** to migrate a **software resource** from its current host **hardware resource** to a different **hardware resource**.

2. Application Programmer Requirements

(a) Non-Quantifiable Requirements

- **WP3.14**(4.1.1.1) GAT and the GAT-API MUST provide a “well” documented Java interface.

2.1.4 WP4

WP4 has indicated no requirements that it has of WP1.

2.1.5 WP5

WP5 only imposes a single requirement on WP1. This requirement, **WP5.1**, is that GAT and the GAT-API support the **Grid Security Infrastructure**, **GSI** for short, as defined in the paper “A Security Architecture for Computational Grids” [5]. As part of the goal of GridLab was to abstract away an explicit dependence upon Globus, WP1 will, during design and implementation, remain compatible with the Globus implementation of the **security policy** and **security architecture** as defined in [5] but do so in a manner that does not tie GAT and the GAT-API directly to Globus.

2.1.6 WP6

These are the set of requirements which were *gathered* as part of D6.1. As in D6.1, the requirements have been grouped into general requirements, authorisation service requirements, and system layer requirements. Also, the requirements have been grouped further into “non-quantifiable requirements” and “quantifiable requirements.” The majority of the requirements presented by WP6 are requirements on WP6. However, many, if not most, requirements on WP6 have ramifications which impose requirements on other WPs. This is the origin of the majority of these WP6 requirements.

1. General Requirements

(a) Non-Quantifiable Requirements

- **WP6.1**(§1) GAT and the GAT-API MUST allow an **application** to define multiple “collaborative groups” within the **virtual organisations**.
- **WP6.2**(§1) GAT and the GAT-API SHOULD supply a **security architecture** that is “transparent” to **application users** and **applications**.

(b) Quantifiable Requirements

- **WP6.3**(§1) GAT and the GAT-API MUST support various types of **resources**.
- **WP6.4**(§1) GAT and the GAT-API MUST use existing and proven **security mechanisms**.

2. Authorisation Service Requirements

(a) Quantifiable Requirements

- **WP6.5**(§2) GAT and the GAT-API MUST support an **authorisation** policy based upon mutual **authentication**.
- **WP6.6**(§2) GAT and the GAT-API MUST support an **authorisation** policy which controls access to objects in the set: logins to user accounts, **resources**, data, **application user** defined objects, and **application programmer** defined objects.

3. System Layer Requirements

(a) Quantifiable Requirements

- **WP6.7** GAT and the GAT-API MUST support supply a method for communication between the **application** and **software resources** which has a **security level** equal to or greater than that specified by the **application**.
- **WP6.8**(§2) GAT and the GAT-API MUST supply infrastructure for **authenticating** all possible **system entities**.

2.1.7 WP7

As described in D7.1, WP1 is a client of WP7. WP7 provides any “adaptive component” functionality that WP1 requires. Hence, WP1 places requirements on WP7, but WP7 does not place requirements on WP7. Thus there are no requirements to enumerate in this section.

2.1.8 WP8

As described in D8.1, see in particular section 4 “WP8 Requirements to other Work Packages,” WP8 places no requirements on WP1. Thus there are no requirements to enumerate in this section.

2.1.9 WP9

As described in D9.1, see in particular section 2 “GridLab Resource Management System - general requirements,” WP9 will define an API, the GRMS-API used for resource management. WP9 requires that this GRMS-API be accessible via the GAT-API. The remainder of D9.1 elaborates on the various requirements that WP9 must satisfy as well as the various requirements WP9 places on other WPs.

- **WP9.1**(§2) The GAT-API MUST support the GRMS Resource Description Language.

2.1.10 WP10

The current deliverables of WP10 provide no requirements placed on WP1 by WP10.

2.1.11 WP11

The current deliverables of WP11 provide no requirements placed on WP1 by WP11, see in particular section 5 “Requirements from Other Work Packages” of D11.1.

2.1.12 WP12

The current deliverables of WP12 provide no requirements placed on WP1 by WP12.

2.1.13 WP13

The current deliverables of WP13 provide no requirements placed on WP1 by WP13.

2.1.14 WP14

The current deliverables do not imply and functional requirements upon WP1, see D14.1. D14.1, on the contrary implies methodological requirements on the development process used within WP1. We introduce the term **GridLab software methodology** to encompass all of the methodological procedures defined in D14.1. Hence, the requirement imposed on WP1 by WP14 can be expressed as follows:

- **WP14.1(D14.1) GAT and the GAT-API MUST be constructed using the GridLab software methodology.**

3 Architecture of the Grid Application Toolkit

The requirements for flexibility and independence of actual resource deployment have lead to a design where the GAT consists of two packages: the *GAT Engine* and a set of *GAT Adaptors* (see figure 1).

The GAT Engine provides the function bindings for the GAT API. When a GAT API call is made, the GAT Engine searches through a database of adaptors and, upon finding an appropriate adaptor providing this capability, dispatches the call to that adaptor. This package is always available to the application developer and the application.

A *GAT Adaptor* provides the interface between the GAT Engine and one or more capabilities. These adaptors translate the user requests into the appropriate interface syntax for accessing the given capability provider; this may utilise grid service discovery mechanisms. The set of active adaptors can change dynamically, based upon the availability of capabilities. Thus an application can operate, with reduced functionality, in the absence of adaptors providing access to certain capabilities.

This section talks about the GAT in abstract terms, and doesn't mention, apart from as examples, any specific technologies, such as WSDL (see 4.6.3) or OGSA (see 4.6.5) which may be used by the GAT Engine or adaptors. Section 4 provides a discussion of such possible technologies.

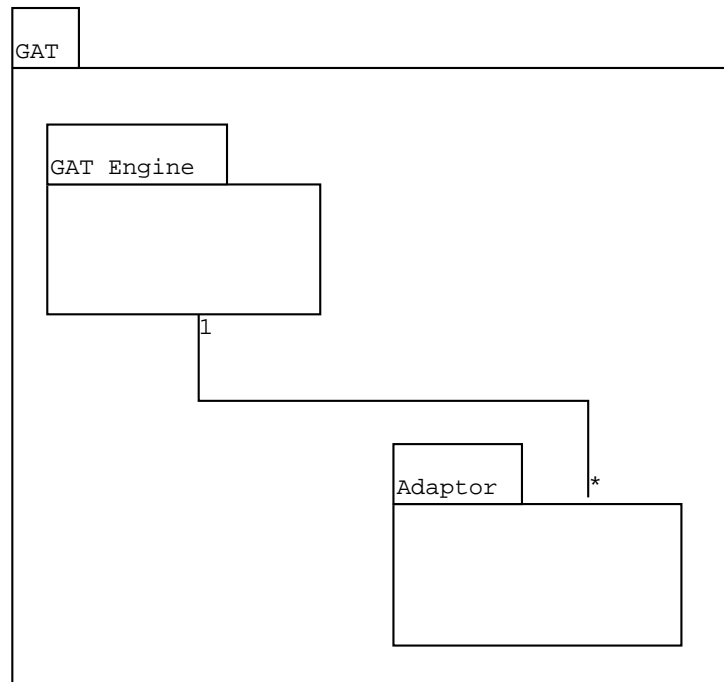


Figure 1: The GAT consists of the GAT Engine and one or more GAT Adaptors. (Note that in principle UML doesn't allow associations between packages, and the association here will be between classes when the design has reached that stage.)

3.1 GAT Engine

The GAT Engine is the part of the GAT responsible for translating a GAT API call made by an application into one or more calls to appropriate capability providers. In order to accomplish this task the GAT must use a **Capability Registry** to track which capabilities are provided by which adaptors. In order to facilitate communication between the different layers, arguments to calls are encapsulated into **Property Tables**.

3.1.1 API function bindings

For each function in the GAT API, the GAT Engine provides a function binding. One of the arguments to the function binding is a property table. This property table contains arguments to be passed to the underlying capability provider, plus hints as to how to pick an appropriate capability provider.

This allows all GAT API functions to have similar signatures, which greatly simplifies the communication between the GAT Engine and adaptors, and also the construction of multi-language bindings. It also allows the construction of a generic function which may be used to access any GAT API function, or to access capabilities which are not available by the standard calls in the currently deployed GAT Engine. This helps the GAT API to be extensible and future proof.

The function binding uses some decision mechanism, which may be a complex function involving the hints passed in, or as simple as trying each possible adaptor in turn, to decide which adaptor's registered function to dispatch the call to. If the call fails the function may either return with an appropriate error code, or it may attempt to use another suitable adaptor. A simple collaboration diagram showing the interaction between an application, a function binding, the

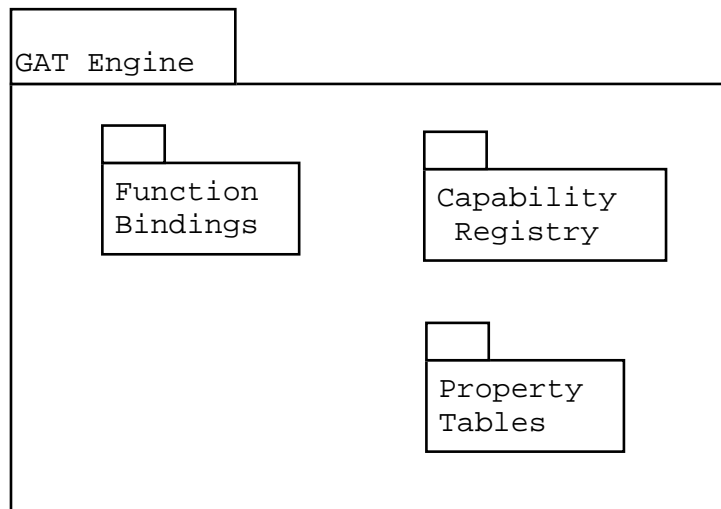


Figure 2: The GAT Engine consists of a capability registry, a set of function bindings, and a property table interface.

capability registry and an adaptor is given in figure 3.

The property table will also be used to specify if the operation is synchronous or asynchronous - in the latter case a new state object will be created which tracks this particular operation, and a subsequent GAT API function will need to be invoked to complete the operation.

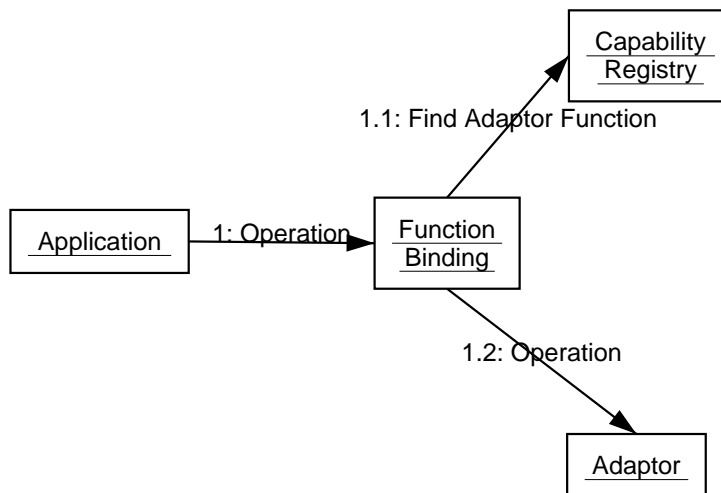


Figure 3: Simple collaboration diagram of an API function call.

3.1.2 Capability Registry

On initialisation, and in response to application- or adaptor-driven events, the GAT Engine constructs or modifies an internal registry of capabilities and capability providers. This registry provides a mapping between capabilities and adaptors providing them. Associated with each mapping is a set of properties which can be used to select between two adaptors providing the same capability.

This is not a general registry, and some adaptors may register the ability to provide many capabilities, and then use external registry services such as UDDI (section 4.6.4), OGSA (section 4.6.5) or WSIL (section 4.6.3) at the time an API call is made to determine the actual capability provider to be used. Thus fine-grained resource discovery, as needed by the TGAT work-package, may be achieved by use of suitable adaptors.

3.1.3 Property Tables

Property tables are opaque objects which can be used to store key-value pairs. The keys are string-valued, and the associated values may be numeric (integer or floating point), strings, or other property tables. These tables will be used to pass data in a transparent manner. Interfaces will be provided to create, destroy, populate and query property tables.

3.2 Adaptors

Adaptors are used to encapsulate mechanism by which a specific capability provider is accessed. Adaptors communicate with the GAT Engine via a standard *Adaptor API*, and with a capability provider through that capability provider's own API. E.g. the adaptor may use ODBC [6] to contact a database server for some capability, use web-service technology to contact an OGSA service [3], use UDP/IP and a custom protocol to contact another service, or make a function call if the capability provider is a library.

Note that this document does not detail the internal design of any adaptors. These will be detailed in the individual adaptors' design documents.

3.2.1 Initialisation and Registration Mechanisms

Each adaptor provides an initialisation function which is called when it is loaded. This function is responsible for using the Adaptor API to register each capability it can service with the capability registry in the GAT Engine.

The adaptor can at any time register its ability to provide different capabilities, or notify the registry that it can no longer deal with a capability.

Registration consists of informing the capability registry that a particular function, provided by the adaptor, should be invoked when the capability is needed, plus some meta-data which can be used to select between different functions providing access to the same capability.

Registration of a function does not necessarily imply that a specific capability provider has been selected at this point, e.g. adaptors providing access to WSDL (section 4.6.3) based services may choose to query a registry service such as UDDI (section 4.6.4) to discover capability providers dynamically.

3.2.2 Function Bindings

As detailed in section 3.1.1 every GAT API call is mapped, via the capability registry to a call to a function in an adaptor. When the adaptor's registered function is called, the adaptor should translate the GAT API function arguments into whatever form is appropriate for the specific binding of the capability or capabilities it is calling, e.g. into a set of SOAP messages for a web service.

The adaptor may choose to use some decision making mechanism to decide which capability provider(s) to contact — e.g. it may use UDDI (section 4.6.4) to discover a service.

In the event that the adaptor cannot honour the application request, it should return an appropriate error code to the GAT Engine.

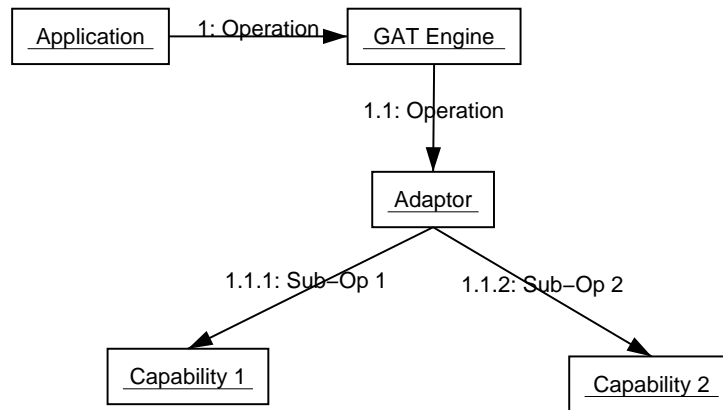


Figure 4: Collaboration diagram showing the case when an adaptor needs to combine operations from two capability providers to honour a request from the GAT API.

3.3 GAT Operation

This section provides details of the operation of the GAT, and the interaction between the GAT Engine and the Adaptors.

3.3.1 Initialisation

Before invoking any GAT functionality an application must initialise the GAT. Upon initialisation the GAT Engine will attempt to initialise any adaptors which it has knowledge of — there are several possible mechanisms to determine this initial list: information passed in by the application in the initialisation call; information available from the environment in which the GAT Engine finds itself; or information provided by adaptors loaded by the previous mechanisms. Each adaptor has an initialisation function, which is called by the GAT Engine. These initialisation functions in turn register details about the capabilities the adaptor can provide with the GAT Engine’s capability registry (section 3.1.2). An example sequence diagram of the GAT initialisation is given in figure 5.

3.3.2 API Function Invocation

On invocation, a GAT API function queries the capability registry (section 3.1.2) to determine which registered function can best provide the requested capability. Once the appropriate registered function has been identified, the API function calls the registered function, see section 3.1.1. The registered function is then responsible for contacting one or more capability providers to provide the necessary functionality, see section 3.2.2.

Figure 6 shows the simplest case, whereby the API call is provided by just one adaptor, and the adaptor in turn only requires one capability provider to service this request. Note that this diagram shows the operation as synchronous, however there is no necessity for this, and, in practise the communication between the adaptor and the capability provider is likely to be asynchronous when a network-enabled capability provider is involved.

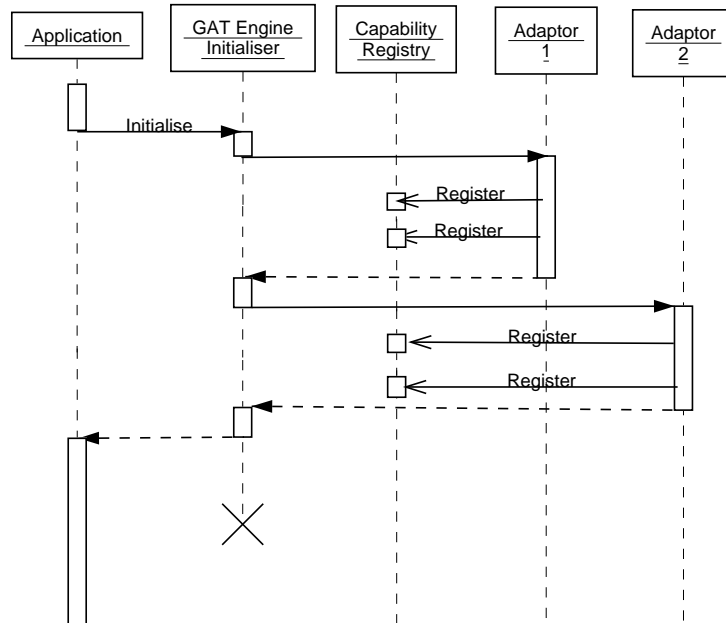


Figure 5: GAT Initialisation — two adaptors are shown for clarity, there can be any number (including zero) of adaptors. Each of these adaptors is shown registering two functions to deal with capabilities

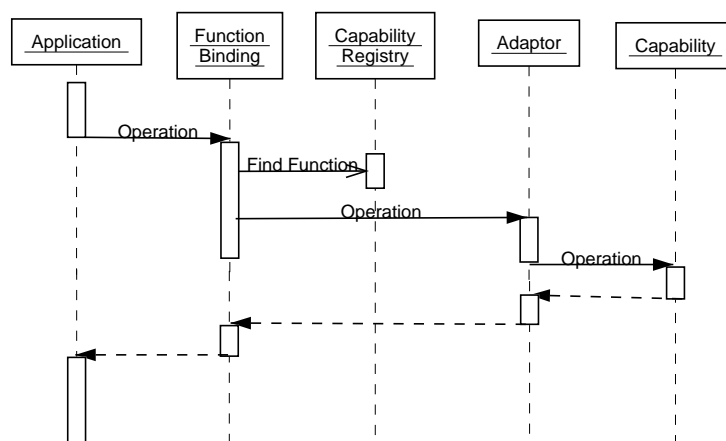


Figure 6: Sequence diagram showing the simplest case of a synchronous GAT API call — only one adaptor and only one capability provider is involved.

3.4 Other Architecture Issues

3.4.1 GAT Contexts

Some applications may operate on behalf of more than one client — either a user (e.g. a portal) or other applications. In order to keep the states of the GAT as seen by different clients distinct, all GAT calls will carry a specific **GAT context** which maintains the current state of active adaptors, outstanding asynchronous GAT API calls, security information (see section 3.4.2), etc. On initialisation a default context will be created, and there will be mechanisms to create, clone or destroy contexts.

3.4.2 Security

Special attention needs to be paid to security within the GAT architecture. There are many possible security mechanisms available, e.g. GSSAPI [7], SSL/TLS, Kerberos, etc. In principle each of these would have an associated adaptor and would register to be able to fulfil security related GAT API calls.

However other adaptors need to make use of these security services. There are two possible mechanisms for this. The first way is for an adaptor to make a GAT API call which returns security information. Another way is to introduce a set of **security contexts** which store the information necessary to make use of particular security infrastructures. In this mechanism when a GAT API call is made to get new credentials from some security provider, a new **security context** is created which is then passed to any GAT API call operating within the same **GAT context** (see section 3.4.1).

If an adaptor function requires some security mechanism to access the capability provider, or needs to provide security information to the capability provider, it may then examine the available security contexts and choose an appropriate one.

4 Technology Survey

In order to realise the design, a number of technologies must be utilised. The precise technology used will depend upon the implementation language.

The canonical implementation will be in C, with Java, Perl, Python and Fortran bindings being supplied. In collaboration with work-package 3 (TGAT), a native Java implementation will be developed. Native Perl and Python implementations may be subsequently developed if this seems advantageous.

While the design has not been carried out to date down to the implementation level, it does seem worthwhile to detail the various technologies which may be used to achieve the desired functionality.

4.1 C Implementation

The C implementation will be the primary development used to prototype the design and API, and will provide the standard behaviour to which other implementations should conform.

4.1.1 Standards and tools

The C implementation should be capable of use on all platforms used in the GridLab project. This implies that the code should conform to the ISO C and POSIX standards. Where non-standard functionality is required the presence of this functionality and means to access it should be determined by use of the *Autoconf* [10] package.

In order to facilitate compilation and installation of the GAT, the standard GNU build tool chain — Autoconf, Automake [11] and make will be used.

4.1.2 Adaptor Discovery and Loading

Adaptor discovery can be done in many ways, some examples of which are listed in 3.3.1. Adaptors will be implemented as dynamically loadable libraries on platforms which support it. Once the library file for a particular adaptor is identified, the GAT Engine must verify that it is in fact the desired adaptor, load it, and initialise it.

- **Verification:** Since the application may be handling security credentials on behalf of one or more users, it is important that any adaptors which are loaded can be trusted. This could be provided by digital signatures associated with each adaptor, which may then be verified.

- **Loading:**

The mechanism to load a dynamic library varies widely from machine to machine. The use of the freely available *libtool* [12] package is one possible solution to this problem. This tool also contains *libltdl* which allows uniform access to libraries independent of the platform specific dynamic library interfaces, and provides the same interface on platforms without dynamic libraries. Loading may either be as a single stage process — load and then initialise, or the library may be loaded, examined for suitability, and then unloaded or initialised depending on the outcome of the inspection.

- **Initialisation:**

Upon loading the adaptor's initialisation function must be called. Since dynamically loadable libraries are not available on all platforms, it is necessary that names are unique, it is proposed that the name of the initialisation function be qualified by the name of the adaptor.

4.1.3 Property Tables

Property tables are opaque objects, with calls to create, destroy, populate and query them. The population mechanisms should be easy to use and as intuitive for programmers — it is proposed that along with calls to set individual properties, calls are provided which populate the tables from a Condor class-ads [13] style string, and from XML with a given schema.

It will be advantageous if the internal representation of the objects can be accessed by native mechanisms in other languages, rather than all bindings to the C implementation of the GAT needing to make calls to the C property table functions.

4.2 Java Binding

Java provides the Java Native interface [14] which may be used to allow Java functions to call C functions. There is support in recent versions of SWIG [15] (version 1.3.6 and above) to create Java bindings.

4.3 Perl Binding

There are two well-established mechanisms for wrapping C libraries to make them accessible from Perl. XS and SWIG [15]. XS is distributed with Perl, SWIG is freely available, and also supports other languages.

4.4 Python Binding

SWIG [15] supports Python bindings. Another possible packages is Weave [16],

4.5 Fortran Binding

For simple interfaces Fortran bindings are simple to produce. Special care needs to be taken with subroutine names and string arguments. The Cactus Computational Toolkit [17] has some scripts which ease the process.

4.6 Adaptor Technologies

Adaptors may make use of many technologies to connect to capability providers, or indeed as capability providers. Some of the core technologies which may be used are Globus, GSI, WSDL, UDDI and OGSA.

4.6.1 Globus

The Globus project [18] has been one of the key players in the evolution of the grid. Globus provides many services which may be connected to via an adaptor, or via an intermediate GridLab service. Future releases of the Globus Toolkit will use OGSA [3] (see section 4.6.5) for inter-operation of services.

4.6.2 GSI

The Grid Security Infrastructure (GSI) [8],[19] is an infrastructure for enabling secure authentication and communication over an open network, based upon the GSSAPI [7] standard. As stated in section 2, work-package 6, the GridLab project will use GSI, and probably use the associated Community Authentication Service (CAS) [20], for authentication and secure communication.

4.6.3 WSDL, WSIL

At the technical board meeting in May 2002 a recommendation was made to use The Web Service Description Language (WSDL) [21] to facilitate communication between services. WSDL is an XML format for format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. It is expected that most adaptors for GridLab services will use WSDL, in combination with the Web Service Inspection Language (WSIL) [22] or other discovery mechanisms, to discover or contact capability providers. For a review of where WSIL fits into the discovery process, please see [23].

4.6.4 UDDI

Universal Description, Discovery and Integration (UDDI) [24] provides another mechanism for web service discovery. Currently there are no plans to use UDDI within the GridLab project.

4.6.5 OGSA

The Open Grid Services Architecture (OGSA) [3] is a proposed evolution of the current Globus Toolkit [18] (see section 4.6.1) towards a Grid system architecture based on an integration of Grid and Web services concepts and technologies. Initial proposed technical specifications have been developed by the Globus Project and IBM, and are being put forward at the Global Grid Forum for discussion, refinement, and eventual standardization.

Whilst the GridLab project predates the OGSA specification, the flexible adaptor architecture of the GAT allows GridLab to make use of it in the same way as the project can make use of existing web service technologies. Once the OGSA project has released a working code base upon which to develop OGSA services, it is likely that the GridLab project will migrate to OGSA.

A Appendix: Glossary

Application - A software component which uses the GAT-API.

Application Layer - A software layer which contains an **application**.

Application Programmer - A programmer who uses GAT and the GAT-API to make an **application**.

Application User - A person who uses an **application**.

Audit Trail Facility - a facility to trace the operations performed in some process in order to log or debug these operations.

Authentication - The process of verifying an identity claimed by or for a **system entity**.

Authorisation - A right or a permission that is granted to a **system entity** to access a system **resource**.

Capability - a piece of functionality. Such functionality may be provided by a library linked into an application, or by an external process. The precise set of capabilities which are accessible to applications via a GAT API call is to be decided by discussion with the CGAT, TGAT and Portals work-packages.

Capability Programmer - A programmer who programs a **capability provider**.

Capability Provider - An object, not part of the application or GAT, which provides some functionality . The object may either be a library or an external process.

Capability Providers Layer - A software layer which contains **capability providers**.

Collaborative Infrastructure - An infrastructure in which many users can collaborate to achieve their goals — e.g. many users controlling or examining a simulation at the same time.

Credential - Data that is transferred or presented to establish either a claimed identity or the **authorisations** of a **system entity** [9].

Fault Tolerant - Describes something which does not fail if one or more things which it utilizes fails.

Firewall - An internetwork gateway that restricts data communication traffic to and from one of the connected networks (the one said to be “inside” the firewall) and thus protects that network’s system resources against threats from the other network (the one that is said to be “outside” the firewall) [9].

GAT Layer - A software layer which contains GAT and the GAT-API.

Grid Service - “A Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgradeability. We expect also to address authorization and concurrency control as OGSA evolves. Two other important issues, authentication and reliable invocation, are viewed as service protocol bindings and are thus external to the core OGSA definition.” [3]

GridLab Service - A **service** provided by the GridLab project.

Hardware Resource - One of the following: processor, volatile memory, non-volatile memory, display, network, or ...

Mobile Computer - A computer whose location may change and with an intermittent and slow network connection.

Protocol - A set of rules (i.e., formats and procedures) to implement and control some type of association (e.g., communication) between systems [9].

Pluggable Architecture - A software organization principle which allows for new functionality to be added to a software component.

Resource - A **hardware resource** or a **software resource**.

Resource Deployment - The process of making a **resource** available to GAT/GAT-API.

Resource Reservation - The process of scheduling a **resource** for use.

Security Level - The combination of a hierarchical classification level and a set of non-hierarchical category designations that represents how secure a system is.

Security Mechanism - A process (or a device incorporating such a process) that can be used in a system to implement a **security service** that is provided by or within the system [9].

Security Policy - A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources [9].

Security Service - A processing or communication service that is provided by a system to give a specific kind of protection to system resources [9].

Service - “**A service is a network-enabled entity that provides a specific capability.** [...] A service is defined in terms of the protocol one uses to interact with it and the behavior expected in response to various protocol message exchanges (i.e., service = protocol + behavior). A service definition may permit a variety of implementations. [...] A service may or may not be persistent (i.e., always available), be able to detect and/or recover from certain errors; run with privileges, and/or have a distributed implementation for enhanced scalability. [...]” [4]

Software Resource - A **service**, **web service**, **grid service**, or a **GridLab service**.

System Entity - An active element of a system [9].

Trust Level - A characterisation of a standard of security protection to be met by a computer system [9].

User Space - A **software layer** which is the union of the **application layer** and the **GAT layer**.

Virtual Organization - A dynamic collection of individuals, institutions, and **resources** [4].

Web Service - "The term **Web services** describes an important emerging distributed computing paradigm that differs from other approaches such as DCE, CORBA, and Java RMI in its **focus on simple, Internet-based standards** (e.g., eXtensible Markup Language: XML [25] [26]) to address heterogeneous distributed computing. Web services define a technique for describing software components to be accessed, methods for accessing these components, and discovery methods that enable the identification of relevant service providers. Web services are programming language-, programming model-, and system software-neutral. Web services standards are being defined within the W3C and other standards bodies and form the basis for major new industry initiatives such as Microsoft (.NET), IBM (Dynamic e-Business), and Sun (Sun ONE). We are particularly concerned with three of these standards: **SOAP, WSDL, and WS-Inspection.**" [3]

References

- [1] <https://www.gridlab.org/WorkPackages/wp-1/Architecture.html>
- [2] S. Bradner, "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels", <http://www.ietf.org/rfc/rfc2119.txt>
- [3] Foster, I., Kesselman, C., Nick, J. and Tuecke, S. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Globus Project, 2002, <http://www.globus.org/research/papers/ogsa.pdf>
- [4] Foster, I., Kesselman, C. and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001. <http://www.globus.org/research/papers/anatomy.pdf>
- [5] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke, "A Security Architecture for Computational Grids," ACM Conference on Computers and Security, pp. 83-91 (1998)
- [6] <http://www.microsoft.com/data/odbc/default.htm>
- [7] J. Linn, "Generic Security Service Application Program Interface: Version 2, Update 1," <http://www.ietf.org/rfc/rfc2743.txt>
- [8] V. Welch, S. Tuecke, D. Engert, and S. Meder, "GSS-API Extensions," http://www.gridforum.org/security/ggf3_2001-10/drafts/draft-ggf-gss-extensions-04.pdf
- [9] R. Shirey, "RFC 2828: Internet Security Glossary," <http://www.ietf.org/rfc/rfc2828.txt>
- [10] <http://www.gnu.org/software/autoconf/autoconf.html>
- [11] <http://www.gnu.org/software/automake/automake.html>
- [12] <http://www.gnu.org/software/libtool/libtool.html>
- [13] <http://www.cs.wisc.edu/condor/classad>
- [14] Sheng Liang, "The Java Native Interface", Addison Wesley, 1999, ISBN 0-201-32577-2

- [15] <http://www.swig.org>
- [16] http://www.scipy.org/site_content/weave
- [17] <http://www.cactuscode.org>
- [18] <http://www.globus.org>
- [19] <http://www.globus.org/security>
- [20] Pearlman, L, Welch, V, Foster, I, Kesselman, C, Teucke, S,
"A Community Authorization Service for Group Collaboration",
http://www.globus.org/security/CAS/CAS_2002_Revised.pdf
- [21] <http://www.w3.org/TR/wsdl>
- [22] <http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html>
- [23] <http://www.webservicesarchitect.com/content/articles/modi01.asp>
- [24] <http://www.uddi.com>
- [25] Bray, T., Paoli, J. and Sperberg-McQueen, C.M., "The Extensible Markup Language (XML) 1.0.," 1998.
- [26] Fallside, D.C., "XML Schema Part 0: Primer. W3C, Recommendation," 2001,
<http://www.w3.org/TR/xmlschema-0/>.