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

D1.3 Implementation and Test Plan

Author(s):	Tom Goodale, Kelly Davis
Document Filename:	Gridlab-1-D1.3-0007.ImplementationAndTestPlan
Work package:	Grid Application Toolkit
Partner(s):	PSNC,ZIB,MU,SZTAKI,VU,ISUFI,CARDIFF,GRIDWARE, COMPAQ,NTUA
Lead Partner:	MPG
Config ID:	GridLab-1-D1.3-0007-0.1
Document classification:	Internal

Abstract: This document describes the plans for implementing and testing the Grid Application Toolkit

Contents

1 Introduction

GridLab WP1's goal is to specify an API — the Grid-Application-Toolkit-API (GAT-API) — and to provide a reference implementation — a Grid-Application-Toolkit (GAT) — of this API. The design of the latter is described in the GAT Technical Specification [1]
The GAT API provides an abstraction layer which abstracts operations which may need to be Grid-aware in such a way that an application may be developed and run independently of the underlying middle-ware actually deployed. The API will be available in several language — C, C++, Java, Fortran, Perl, and Python; other languages may also be made available in the future.
The GAT consists of a library – **the GAT Engine** – which all applications link against, 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 technical reasons for this are outlined in the GAT Technical Specification [1].
The APIs can be split into four parts:

- GAT Application API

  This is the API which provides the "grid" operations.

- GAT Application Utility API

  These are GAT Engine management and interaction operations which an application needs to call to interact with the GAT.

- 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 GATEngine and possibly with other adaptors and utility libraries.

1.1 Purpose of Document

This document provides details of the implementation and test plans for the Grid Application Toolkit.

1.2 Structure of Document

This document is split into two main sections:

section 2 details the tasks in the development of the GAT API and implementations.

section 3 details the policies which will be used in further development of the APIs and implementations.

1.3 Status of this Document

This document is in a preliminary stage. While it identifies tasks, it does not give concrete time schedules for these tasks. This will be amended in a later release of this document.

1.4 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 [3].

2 Implementation Plan

Implementation of the GAT starts with the GAT-API specification development process. With the release of a the first version of the GAT API C-bindings, development of reference implementations can proceed.

2.1 GAT API Development

The GAT API development process is a three-stage process:

1. Identify the fundamental Grid operations and the methods which should be used to access these operations. These methods can be specified abstractly by their inputs, outputs, actions and exception states.

2. Identify the objects the methods act upon. The information from stage (1) can be used to identify objects which the API should act upon, and the API can be reformulated in an object-based manner, which will be more suitable for languages which are natively object-based such as Java and Python. This stage provides a new language-independent API.

3. Specific language bindings. In this stage the document from stage (2) is used to formulate language specific APIs.

Each of these documents should be reviewed, and additionally the application-oriented work-packages WP2, WP3 and WP4 should examine the APIs for suitability.

2.2 C Reference Implementation

With the release of a C-language binding a native C reference implementation will be developed. This implementation will be accompanied by a reference adaptor which provides minimal functionality for each API method.
The combination of the reference implementation and the reference adaptor will be used to generate a test-suite which can be used both to check that modifications to these reference implementations has not changed functionality, but also to check other adaptors against the reference behaviour.
A future revision of this document will contain a list and detailed description of each of these tests.

2.3 Other Language Bindings

With the completion of a working C reference implementation and a test-suite, the other language APIs will be implemented by wrapping the C reference implementation as described in the Technical Specification [1].
A test-suite will be constructed for each one of these bindings based upon the C reference test-suites.

2.4 Java Native Implementation

Two of the application workpackages Triana (WP3) and the Portal (WP4) are Java-based, and thus will require the Java GAT-API. While this is available via the Java binding to the C reference API, a native Java implementation of the GAT would provide efficiency and deployment advantages. WP1 will, in cooperation with these two workpackages provide a native Java implementation of the GAT.
This implementation will be tested against the reference C implementation by use of the test-suites developed for the Java binding as described in section 2.3.

2.5 Adaptor Development

GridLab is developing many services which will be accessible via the GAT API. Access to these services is via adaptors as discussed in the Technical Specification [1]. WP1 will work in cooperation with the service developers to provide adaptors for the GAT to use these services. Such adaptors will be developed both for the reference C implementation and for the native Java implementation.
All adaptors will be tested for minimal functionality via the test-suites, and additional test-suites will be generated to test functionality which would not be available via the minimal reference adaptor.

2.6 Documentation

A Users' Guide will accompany each release of the API. Additionally the reference C and the Java native implementations will contain adaptor-writers' guides.
The Users' Guide will be based upon the abstract API, and will explain the use of every API call. It will also detail how each API call maps into specific languages. This approach will allow us to maintain just one version of the Users' Guide, rather than one for each language which could become a heavy workload and is a potential source of error.
Adaptor-writing is heavily language dependent, and so will not follow this model, but rather there will be one document per native implementation.

2.7 Further Testing

In co-operation with the application oriented workpackages WP2, WP3 and WP4, WP1 will develop enhanced tests which demonstrate and test the ability of the API to implementations to perform scenarios identified by these workpackages.

3 Evolution of APIs and Implementation

Section 2 is effectively a task list for the API development. However once the achievement of these tasks is an iterative process, and at all stages deficiencies may be found in the results of previous stages. In this event the deficiencies must be addressed and the relevant documents and implementations corrected, re-reviewed, and re-released.

3.1 API Evolution

We plan to hold regular reviews of the API, based upon application experiences of use of the API and any issues reported to us. If new operations are identified the first stage document will

be amended as necessary, and changes propagated to the object based API document (stage 2), and then these changes will be propagated to the language specific (stage 3) documents.
The stage 2, object based, document is to be counted as the master API specification from which all language-specific ones are derived. Any deficiencies or changes which are identified should be fed into this document and revised language-specific API specification derived.

3.2 Implementation Evolution

Changes in the language bindings will be fed into new releases of implementation. If the changes require a revision of the test-suites these will be updated as necessary. At all stages the C reference implementation will be the standard against which other implementations or language bindings should be tested.
As the design of the implementations is revised, these revisions will be fed back into the Technical Specification [1]

3.3 Documentation Evolution

Updates to the APIs will naturally lead to updates of the Users' Guide. An updated Users' Guide will accompany each new release of the C reference implementation.

A Procedures

A.1 Coding Guidelines

The GridLab coding guidelines will be used in all source files.

A.2 Version Control

Source code and document development will make use of CVS for version control and change tracking. The GridLab CVS server will be used.
Version numbering in releases will follow the standard open-source policy of a set of three numbers — major, minor, micro – of which odd numbers in the minor indicate a development sequence and even numbers indicate a stable sequence which consists of bug-fixes only.

A.3 Bug/Issue Tracking

The GridLab Bugzilla server will be used to track the status of bugs, feature requests and other issues.

B Task List

This is a preliminary task list, when the time-scales have been decided, this table will be updated with dates.

Task	Status
API: Language Independent	Under Review
API: Language Independent: Object Based	Under Review
API: C	Started
API: C++	
API: Java	Started
API: Fortran	
API: Perl	
API: Python	
Documentation: Users' Guide	Started
Documentation: Revised Technical Specification	Started
Implementation: C reference	Started
Documentation: C Adaptor Writers' Guide	Started
Test-suite: C	
Bindings: C++	
Bindings: Java	
Bindings: Fortran	
Bindings: Perl	
Bindings: Python	
Test-suite: C++	
Test-suite: Java	
Test-suite: Fortran	
Test-suite: Perl	
Test-suite: Python	
Implementation: Java native	
Documentation: Java Adaptor Writers' Guide	

References

[1] K.Davis and T.Goodale, "D1.2 Technical Specification", http://www.gridlab.org/Resources/Deliverables/D1.2b.pdf

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

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