

The Grid Application Toolkit

Abstracting the Grid for Application Programmers

Hartmut Kaiser

hartmut.kaiser@aei.mpg.de

MPI for Gravitational Physics, Albert-Einstein-Institut, Golm

Tom Goodale, Gabrielle Allen, Edward Seidel
Kelly Davis, Thilo Kielmann, André Merzky
The GridLab Project
www.gridlab.org

- Why another Grid-API?
- What does it offer/solve?
- Simple samples
- Architecture / Implementation
- Conclusions

Why another Grid-API?

The situation today:

- Grids: everywhere

- Supposedly. At least many projects

- Grid applications: nowhere

- Almost. At least our experience that this is difficult, GGF APPS group

Why is this?

- **Application programmers accept the Grid as a computing paradigm only very slowly.**

- Problems: (multifold and often cited - amongst others)

- Interfaces are **NOT** simple (see next slides. . .)

- Typical Globus code... ahem...

- Different and evolving interfaces to the 'Grid'

- Versions, new services, new implementations, WSDL does not solve all problems at all

- Environment changes in many ways

- Globus, grid members, services, network, applications, ...

Copy a File: Globus GASS

```

int RemoteFile::GetFile (char const* source,  char const* target) {
globus_url_t          source_url;
globus_io_handle_t   dest_io_handle;
globus_ftp_client_operationattr_t source_ftp_attr;
globus_result_t      result;
globus_gass_transfer_requestattr_t source_gass_attr;
globus_gass_copy_attr_t source_gass_copy_attr;
globus_gass_copy_handle_t gass_copy_handle;
globus_gass_copy_handleattr_t gass_copy_handleattr;
globus_ftp_client_handleattr_t ftp_handleattr;
globus_io_attr_t      io_attr;
int                  output_file = -1;

if ( globus_url_parse (source_URL, &source_url) != GLOBUS_SUCCESS ) {
    printf ("can not parse source_URL \"%s\"\n", source_URL);
    return (-1);
}

if ( source_url.scheme_type != GLOBUS_URL_SCHEME_GSIFTP &&
    source_url.scheme_type != GLOBUS_URL_SCHEME_FTP &&
    source_url.scheme_type != GLOBUS_URL_SCHEME_HTTP &&
    source_url.scheme_type != GLOBUS_URL_SCHEME_HTTPS ) {
    printf ("can not copy from %s - wrong prot\n", source_URL);
    return (-1);
}

globus_gass_copy_handleattr_init (&gass_copy_handleattr);
globus_gass_copy_attr_init (&source_gass_copy_attr);

globus_ftp_client_handleattr_init (&ftp_handleattr);
globus_io_fileattr_init (&io_attr);

globus_gass_copy_attr_set_io (&source_gass_copy_attr, &io_attr);
globus_gass_copy_handleattr_set_ftp_attr (&gass_copy_handleattr,
&ftp_handleattr);
globus_gass_copy_handle_init (&gass_copy_handle,
&gass_copy_handleattr);

if (source_url.scheme_type == GLOBUS_URL_SCHEME_GSIFTP ||
    source_url.scheme_type == GLOBUS_URL_SCHEME_FTP ) {
    globus_ftp_client_operationattr_init (&source_ftp_attr);
    globus_gass_copy_attr_set_ftp (&source_gass_copy_attr,
&source_ftp_attr);
}
else {
    globus_gass_transfer_requestattr_init (&source_gass_attr,
source_url.scheme);
    globus_gass_copy_attr_set_gass (&source_gass_copy_attr,
&source_gass_attr);
}

output_file = globus_libc_open ((char*) target,
O_WRONLY | O_TRUNC | O_CREAT,
S_IRUSR | S_IWUSR | S_IRGRP |
S_IWGRP);

if ( output_file == -1 ) {
    printf ("could not open the file \"%s\"\n", target);
    return (-1);
}
/* convert stdout to be a globus_io_handle */
if ( globus_io_file_posix_convert (output_file, 0,
&dest_io_handle)
!= GLOBUS_SUCCESS) {
    printf ("Error converting the file handle\n");
    return (-1);
}

result = globus_gass_copy_register_url_to_handle (
&gass_copy_handle, (char*)source_URL,
&source_gass_copy_attr, &dest_io_handle,
my_callback, NULL);
if ( result != GLOBUS_SUCCESS ) {
    printf ("error: %s\n", globus_object_printable_to_string
(globus_error_get (result)));
    return (-1);
}
globus_url_destroy (&source_url);
return (0);
}

```

Copy a File: CoG/RFT

```

package org.globus.ogsa.gui;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.net.URL;
import java.util.Date;
import java.util.Vector;
import javax.xml.rpc.Stub;
import org.apache.axis.message.MessageElement;
import org.apache.axis.utils.XMLUtils;
import org.globus.*
import org.gridforum.ogsi.*
import org.gridforum.ogsi.holders.TerminationTimeTypeHolder;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class RFTClient {
    public static void copy (String source_url, String target_url) {
        try {
            File requestFile = new File (source_url);
            BufferedReader reader = null;
            try {
                reader = new BufferedReader (new FileReader (requestFile));
            } catch (java.io.FileNotFoundException fnfe) { }
            Vector requestData = new Vector ();
            requestData.add (target_url);
            TransferType[] transfers1 = new TransferType[transferCount];
            RFTOptionsType multirftOptions = new RFTOptionsType ();

            multirftOptions.setBinary (Boolean.valueOf (
                (String)requestData.elementAt (0)).booleanValue ());
            multirftOptions.setBlockSize (Integer.valueOf (
                (String)requestData.elementAt (1)).intValue ());
            multirftOptions.setTcpBufferSize (Integer.valueOf (
                (String)requestData.elementAt (2)).intValue ());
            multirftOptions.setNotpt (Boolean.valueOf (
                (String)requestData.elementAt (3)).booleanValue ());
            multirftOptions.setParallelStreams (Integer.valueOf (
                (String)requestData.elementAt (4)).intValue ());
            multirftOptions.setDcau (Boolean.valueOf (
                (String)requestData.elementAt (5)).booleanValue ());

            int i = 7;
            for (int j = 0; j < transfers1.length; j++)
            {
                transfers1[j] = new TransferType ();

                transfers1[j].setTransferId (j);
                transfers1[j].setSourceUrl ((String)requestData.elementAt (i++));
                transfers1[j].setDestinationUrl ((String)requestData.elementAt (i++));
                transfers1[j].setRftOptions (multirftOptions);

                TransferRequestType transferRequest = new TransferRequestType ();
                transferRequest.setTransferArray (transfers1);

                int concurrency = Integer.valueOf
                    ((String)requestData.elementAt(6)).intValue();

                if (concurrency > transfers1.length)
                {
                    System.out.println ("Concurrency should be less than the number"
                        "of transfers in the request");
                    System.exit (0);
                }
                transferRequest.setConcurrency (concurrency);

                TransferRequestElement requestElement = new TransferRequestElement ();
                requestElement.setTransferRequest (transferRequest);

                ExtensibilityType extension = new ExtensibilityType ();
                extension = AnyHelper.getExtensibility (requestElement);

                OGSIServiceGridLocator factoryService = new OGSIServiceGridLocator ();
                Factory factory = factoryService.getFactoryPort (new URL (source_url));
                GridServiceFactory gridFactory = new GridServiceFactory (factory);

                LocatorType locator = gridFactory.createService (extension);
                System.out.println ("Created an instance of Multi-RFT");

                MultiFileRFTDefinitionServiceGridLocator loc
                    = new MultiFileRFTDefinitionServiceGridLocator();
                RFTPortType rftPort = loc.getMultiFileRFTDefinitionPort (locator);
                ((Stub)rftPort)._setProperty (Constants.AUTHORIZATION,
                    NoAuthorization.getInstance ());
                ((Stub)rftPort)._setProperty (GSIConstants.GSI_MODE,
                    GSIConstants.GSI_MODE_FULL_DELEG);
                ((Stub)rftPort)._setProperty (Constants.GSI_SEC_CONV,
                    Constants.SIGNATURE);
                ((Stub)rftPort)._setProperty (Constants.GRIM_POLICY_HANDLER,
                    new IgnoreProxyPolicyHandler ());

                int requestid = rftPort.start ();
                System.out.println ("Request id: " + requestid);

            }
            catch (Exception e)
            {
                System.err.println (MessageUtils.toString (e));
            }
        }
    }
}

```

Copy a File: GAT/C

```
#include <GAT.h>

GATResult RemoteFile_GetFile (GATContext context,
    char const* source_url, char const* target_url)
{
    GATStatus status = 0;
    GATLocation source = GATLocation_Create (source_url);
    GATLocation target = GATLocation_Create (target_url);
    GATFile file = GATFile_Create (context, source, 0);
    if (source == 0 || target == 0 || file == 0) {
        return GAT_MEMORYFAILURE; }
    if ( GATFile_Copy (file, target, GATFileMode_Overwrite) != GAT_SUCCESS )
    {
        GATContext_GetCurrentStatus (context, &status);
        return GATStatus_GetStatusCode (status);
    }
    GATFile_Destroy (&file);
    GATLocation_Destroy (&target);
    GATLocation_Destroy (&source);

    return GATStatus_GetStatusCode (status);
}
```

Copy a File: GAT/C++

```
#include <GAT++.hpp>

GAT::Result RemoteFile::GetFile (GAT::Context context,
                                std::string source_url,
                                std::string target_url)
{
    try
    {
        GAT::File file (context, source_url);
        file.Copy      (target_url);
    }
    catch (GAT::Exception const &e)
    {
        std::cerr << "Some error: " << e.what() << std::endl;
        return e.Result();
    }
    return GAT_SUCCESS;
}
```

Copy a File: GAT/C++

```
#include <GAT++.hpp>

GAT::Result RemoteFile::GetFile (GAT::Context context,
                                std::string source_url,
                                std::string target_url)
{
    try
    {
        GAT::File file (context, source_url);
        file.Copy      (target_url);
    }
    catch (GAT::Exception const &e)
    {
        std::cerr << "Some error: " << e.what() << std::endl;
        return e.Result();
    }
    return GAT_SUCCESS;
}
```

Code	GASS	CoG	C GAT	C++ GAT
Lines	100	80	20	15
total	30	30	5	2
Action	30	30	1	1
Cleanup	20	10	9	2
Languag	10	5	5	5

e

- Globus, Unicore, my_service, your_service, . . .
The same functionality has different interfaces all over the place.
 - But you don't want to recompile your app every time, not to speak of recoding...
 - WSDL does not mean end of all problems (see CoG code), but begin of new ones... - on application level, WSDL is not trivial enough
- Restricting yourself to Globus does not help either: version changes every couple of months (2.4.x, 3.2.y, 4.a.b)
 - and gets bug fixes. Changes often are MAJOR - we have seen a number of them over the last couple of years...

The application that runs today will fail tomorrow!

Right now, it is basically impossible for a programmer to focus on the science, not on IT (i.e. Grid) problems.

- Services (and interfaces) get exchanged (“upgraded”) on regular basis
 - That is related to the point above, but also a social problem!
 - Institutions (resources, services, applications) join/leave **YOUR** grid without (much) notice.
 - The grid is designed to ease and simplify that kind of fluctuation - its not a bug, its a feature!
 - But the applications are not able to make use of that feature right now ...
 - The Grid changes **AT RUNTIME** – services go down, resources get busy/free, disks and storage nodes are empty/full, . . . **THINGS CONSTANTLY CHANGE.**
 - Today Grid middleware allows to cope with that, but utilizing that in an intelligent way is a major programming effort, and blows the application with code that needs constancy maintenance...
- Applications need **LOTS** of code for handling transient problems.*
- Most applications share most of these problems, but code reuse is difficult/impossible.
 - We can reuse the Globus libraries, right, but isn't every project re-inventing its own abstraction layer for these? In our experience/projects: they do!

Aren't we all re-inventing abstraction layers for this?

The key objective for application programmers

- Remember: an applications programmer is a physicist, chemist, linguist , medical

Simple API's should:

- be easy to use
 - Simple, finite, consistent API which allows error tracing
- be invariant: make upgrades really, really simple
 - Well defined API which rarely changes.
 - Implementation which allows dynamic exchange of key elements and provides runtime abstractions.
- avoid refactoring/recoding/recompilation
 - Same applications runs today and tomorrow; here and there; on Globus and Unicore; on Globus 2.2.4 and Globus 2.4; on Linux and on Mac; local and on grid;
- focus on well-known programming paradigms
(e.g., for a file provide a file API – without services to services to files. . .)
 - Files are best example: expect open, close, read, write, seek. Do not introduce fancy things like the need to ask a service discovery service to tell me the location of an service which is able to tell me the location of my file...

...and what they GAT:

- Enough of 'we want this' and 'we don't want that' - you got the picture, right? So, here is what we do:

- An API that allows to implement **basic** Grid use cases
- **Stay simple! As simple as possible! But not simpler!**
- Focus on applications, and scientists, rather than Grid nerds 😏

- As you and me
- Next slides will give an overview of what we think is essential, and how we envision usage of that. Remind: this is version 0.9 - our first shot - we know its not perfect, but we are converging to something we can work with already.

- Files
- Monitoring and Events
- Resources, Jobs
- Information Exchange
- Utility classes (error handling, security, preferences...)

NOTHING ELSE

API: Sub Systems

File Subsystem		
GATFile	GATLogicalFile	
GATEndpoint	GATPipeListener	GATPipe
Monitoring and Event Subsystem		
GATRequestListener	GATRequestNotifier	GATAction
GATMetricListener	GATMetric	GATMetricEvent
Information Exchange Subsystem		
GATAdvertisable	GATAdvertService	
Resource Management Subsystem		
GATSoftwareDescription	GATResourceDescription	GATResource
GATJobDescription	GATResourceBroker	GATReservation
GATJob		
Utility Subsystem		
GATSelf	GATContext	GATSecurityContext
GATStatus	GATPreferences	URL, Time, ...

- The pipe stuff in the file subsystem is 'historical', pipe is VERY simple - we don't do real communication and data exchange a la MPI!
- The actual API is somewhat larger (especially the resource part): 34 objects as opposed to 27 shown here. BUT THAT'S IT!!!

Examples in GAT

- Read remote physical file
- Read a logical file
- Spawn a Subtask
- Migrate a Subtask

Read a remote physical File

```
try {
    char data[25];

    GAT::File file (context, source_url);

    file.open (RD_ONLY);
    file.seek (100, SEEK_SET);
    file.read (data, sizeof(data));
    file.close ();
}
catch (GAT::Exception const &e)
{
    std::cerr << "Some error: " << e.what() << std::endl;
    return e.Result();
}
```

- Well known paradigm.
- Whatever service/lib/... implements that, the programmer does not know (no reference to Globus...)
- Whatever the URL/protocol (ftp://, gsiftp://, http:// file://) no code changes! No service specific parameter settings (can be drawback BUT SIMPLIFIES)

Read a logical file

```
try {
    char data[25];

    GAT::LogicalFile      logical_file (context, name);
    list<GAT::File> files = logical_file.get_files ();

    files[0].open        (RD_ONLY);
    files[0].seek        (100, SEEK_SET);
    files[0].read        (data, sizeof(data));
    files[0].close      ();
}
catch (GAT::Exception const &e)
{
    std::cerr << "Some error: " << e.what() << std::endl;
    return e.Result();
}
```

- SAME paradigm + 'private name space'
- URL Unknown! Service unknown! complete abstraction ==> Virtualization!
- Still simple

Spawn a Subtask

```
GAT::Table sdt;    sdt.add ("location",    "/bin/date");
GAT::Table hdt;    hdt.add ("machine.type", "i686");

GAT::SoftwareDescription    sd  (sdt);
GAT::HardwareResourceDescription    hrd (hdt);

GAT::JobDescription jd (context, sd, hrd);
GAT::ResourceBroker rb (context, prefs);

GAT::Job j = rb.submit (jd);
```

Migrate a Subtask

```
GAT::Table sdt;    sdt.add ("location",    "/bin/sleep");
                  sdt.add ("arguments",    "36000");
GAT::Table hdt;    hdt.add ("machine.type", "i386");

GAT::SoftwareDescription    sd  (sdt);
GAT::HardwareResourceDescription    hrd (hdt);

GAT::JobDescription    jd  (context, sd, hrd);
GAT::ResourceBroker    rb (context, prefs);

GAT::Job    job = rb.submit (jd);

hdt.set ("machine.name", "fs0.das2.cs.vu.nl");

list<GAT::Resource>    resources = rb.find_resources (hrd);

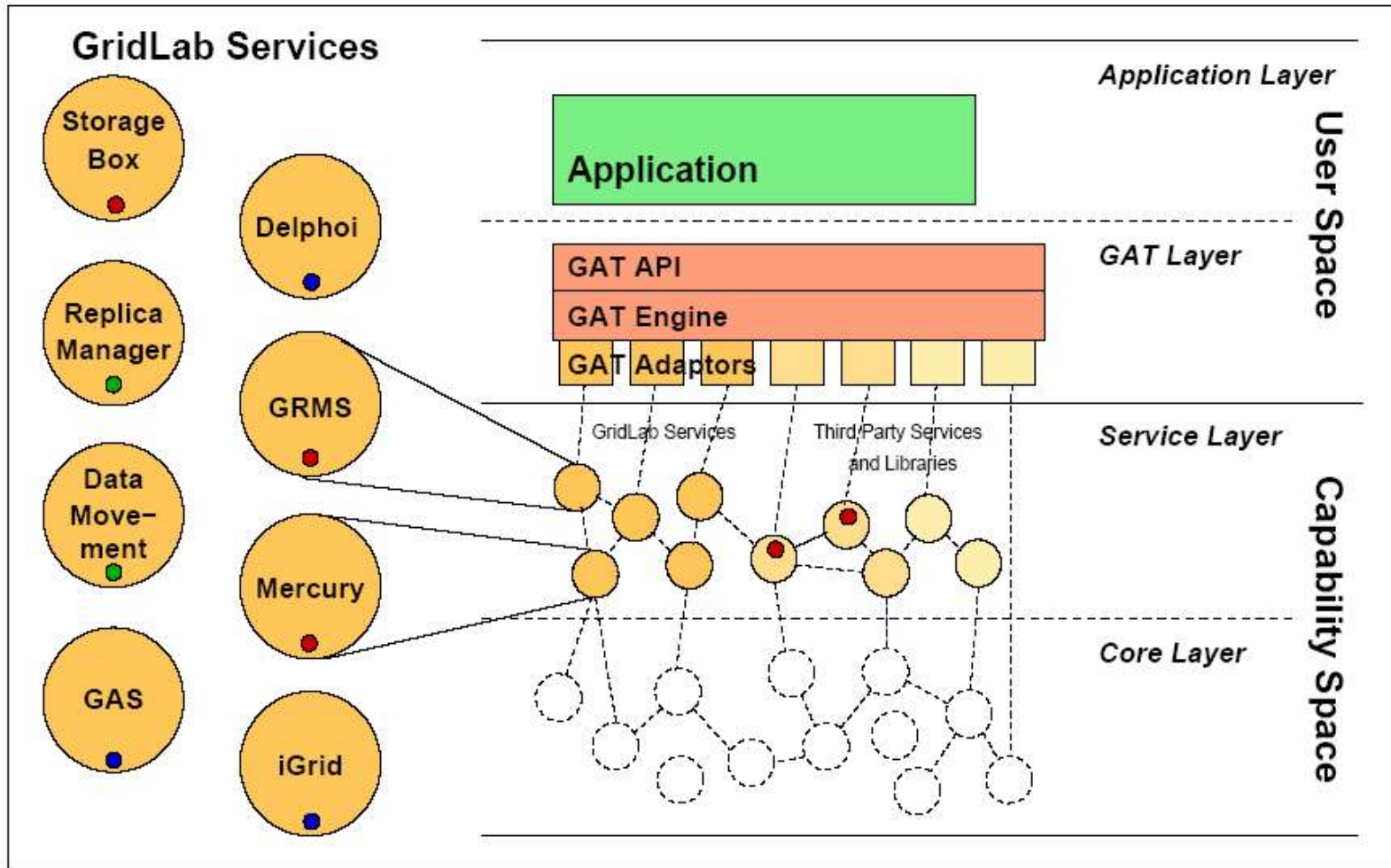
job.Migrate (resources[0]);

if (GATJobState_Running == job.GetState ())
    job.Stop ();
```

- Version: 0.9 – currently converging towards 1.0
- Is object oriented, but language neutral
- Defines syntax and semantics of GRID access
- Specification is open process
 - Hopefully gets input from many communities
 - Will evolve along with the findings of GGF's new SAGA-WG (Simple APIs for Grid Applications)

- C Version fully implemented (99.9%)
- C++ Wrapper to C (implementation prototype)
- Java Version started
- Fortran, Perl, Python (wrappers to C) to follow
- Focus: portability, lightness, flexibility, adaptivity

- **API** is a very thin layer, provides capabilities by itself
- **Adaptors** implement **capabilities** mirroring the API (bind to the Grid-Environment)
- **Engine** mediates between API and adaptors:
 - switch adaptors at runtime (shared libraries)
 - error tracing and fallback mechanisms (default local adaptor set)
- **CPI** is also well defined - adaptors are interchangeable



- The **GAT** provides a simple and stable API to various Grid environments
- Downloads:
 - www.gridlab.org
 - Currently, snapshots available
 - GAT 1.0, available in early June 2004
 - Platforms: Linux, Windows, Mac OS X, SGI Irix, True64 UNIX
- Support via mailing list **gat@gridlab.org**