Accelerated Development of Grid Applications

17th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2005)

Artur Andrzejak
Zuse Institute Berlin (ZIB)
Germany

Tutorial overview

- Grid and Grid the programming problem

- What is GAT / JavaGAT?
- Fundamental JavaGAT Classes and Interfaces
- Manipulating Files (in JavaGAT)
- FileStreams – Reading and Writing Files
- Advertising Services
- Interprocess Communication
- Job Management

- Triana Workflow Programming System
- Triana demo

- Installing JavaGAT and Hands-On Exercises

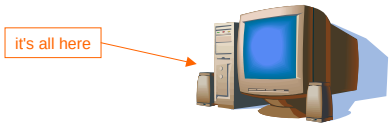
Computing locally is easy..

INSTRUCTIONS

1. Turn on the computer
2. Put your data on the harddrive
3. Compute

Computing locally: *why* is it easy?

- You know where the data is
- You know where your processor(s) is/are (hopefully)
- The communication (data – processor, processor –processor,..) is transparent, fast and reliable
- You start, stop, and monitor the jobs directly
- You control the machine (well..)



Computing in Grids is not so easy..

Local	Grids	Domain
You know where the data is	You have to move / make available the data first	**Grid I/O** **Data management**
You know where your processor(s) is / are	You have to find the appropriate and free resources	**Resource Discovery** **Resource Management**
The communication is transparent, fast and reliable	Communication speed varies, failures are normal, transparency – just forget	**Interprocess Communication** **Data Management** **Programming Models**
You start, stop, and monitor the jobs directly	You need tools & credentials to manage jobs	**Job Management** **Access Control (?)**
You control the machine	Your job and data is at a mercy of somebody who *might not even be a dog*	**Security** **Fault-tolerance**

The Grid programming problem

- **The primary activities to solve these problems focused on developing infrastructure(s):**
  - **Globus TK**
  - **Condor**
  - **Unicore**
  - **ProActive**
  - **..**
- **Simultaneously, two big issues have been created**
  1. **The (unnecessary) complexity of APIs and infrastructures**
  2. **The heterogeneity and lacking interoperation between the solutions**

1. The complexity of existing APIs

- **The Globus file copy (in C++)**

```
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);
                                   &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);
}
```

2. The heterogeneity / missing interoperation

- **Changing interfaces**
  - **several new Globus versions per year**
  - **code becomes a "legacy" solution even before the application is finished!**
- **Instead of focusing on a single API / framework, there is obviously more interest in developing "yet another infrastructure"**
  - **frequently "own authorship" reasons**
- **Usually a Virtual Organization requires same software version on all participating machines**
  - **Ensuring a uniform software basis becomes a larger interoperability problem than overcoming organizational boundaries**

Grid Application Toolkit (GAT)

An easier way to program Grid applications

What is GAT?

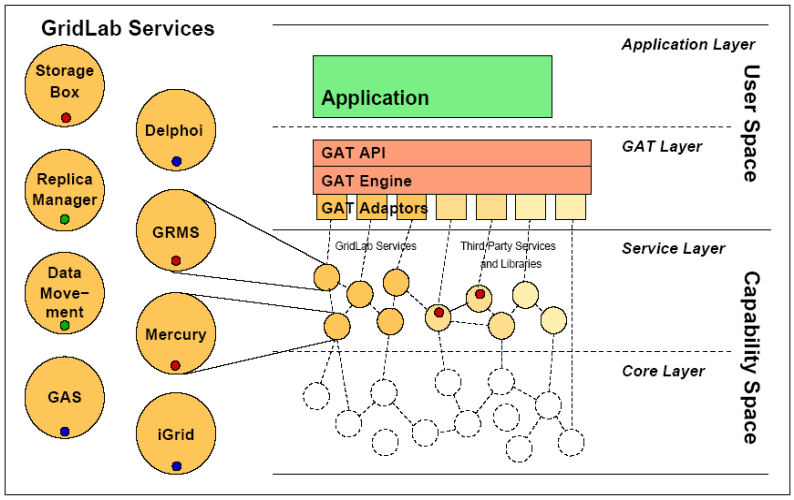
- **GAT: Grid Application Toolkit**
- **Goal: ensure *technology independence* and *simplicity* for programming of Grid applications**
  - **Officially: *"API and Toolkit for developing and running portable grid applications independently of the underlying grid infrastructure and available services"***
- **GAT consists of**
  - **simple interface (GAT API)**
  - **(transparent) call dispatcher (engine)**
  - **(hidden) adapters (service providers)**

  for accessing various Grid middleware implementations

What is GAT? (2)

- **Realisation:**
  - **GAT-API: Provides a common set of interfaces (in C, C++, Python, Java) for simple access to Grid services**
  - **GAT Engine: implements the GAT-API and dynamically dispatches the calls to the underlying service / functionality provider**
  - **GAT Adaptors: functionality providers which connect to the GAT Engine as *plugins***
    - **examples: local, Globus, Condor, Unicore,..**

GAT Architecture



What advantages has programming with GAT?

1. **Simpler code: file copy with GAT (C++)**

```
#include <GAT++.hpp>
void RemoteFile::GetFile {
        GAT::Context context, std::string source_url,
        std::string target_url) throws GAT::Exception {
                GAT::File file (context, source_url);
                file.Copy(target_url);
}
```

2. **"Write once, run *somewhere*"**
   - **applications don't need to know which underlying infrastructure(s) are available**
   - **the GAT engine determines dynamically which adaptor(s) provide the requested operation**
   - **if one adaptor fails, the next one is tried**

Existing Adaptors – Java GAT 1.2

LOCAL ADAPTORS

adaptor	status
File Adaptor:	done
FileStreams:	done
RandomAccessFile:	done
Logical file adaptor:	done
pipe (sockets):	done
Advertservice Adaptor:	done
ResourceBroker Adaptor:	done
monitoring:	done

Existing Adaptors – Java GAT 1.2 (2)

REMOTE ADAPTORS

adaptor	status
GridLab service discovery (igrid):	coding
GridLab File Adaptor (file movement&browsing):	done
GridLab LogicalFile Adaptor:	done
GridLab Advertservice Adaptor (storagebox):	done
GridLab ResourceBroker Adaptor:	done
Gridlab monitoring:	coding
LogicalFile on top of File adaptor:	done
Pipe adaptor (sockets):	done
Globus Gridftp File adaptor:	done
Globus Gridftp FileStreams:	done
Globus ResourceBroker:	done
FTP File:	done
SSH File:	coding
FTP FileSteams:	done
HTTP FileStreams:	done
HTTPS FileStreams:	done

GAT is a part of GridLab

GridLab is a large EU-FP5 project with several workpackages

- Grid Application Toolkit, GAT
  - high level API to complex and dynamic Grid Environments
- Grid Portals
  - GridSphere (Grid-Portal development framework
- Grid Resource Management
  - GRMS (Grid Resource Management and Brokering Service)
- Grid Security
  - GAS (Grid Authorization Service)
- Grid Monitoring
  - Mercury Monitor for grid performance monitoring
- Grid Data and Visualisation Services
  - Storagebox, ZIBDMS - Services to manage public and user-private files
- Grid Information Services
  - iGrid, the GridLab Information Service
- Grid Adaptive Services
  - Delphoi, adaptive components for GAT, e.g. performance-prediction models
- Grid Mobile Services
  - Grid Services supporting wireless technologies
- Grid Testbed
  - A large-scale Grid testbed, a prototype of future production Grids

- **The basis for all packages of GridLab is GAT**
- **ZIB (my institute) participated in GridLab, but is *not* the primary developer of GAT**

What can GAT do for you?

Functionality	GAT Package
Grid I/O, Data management	**File Management** **FileStream Management** **LogicalFile Management**
Find the appropriate and free resources	**Advert Management** **Resource Management**
Starting, stoping, and monitoring jobs	**Job Management** **Monitoring**
Communicate	**Interprocess Communication**
Security & Reliability	**Mechanisms "embedded" in functions**

What can GAT do for you? (2)

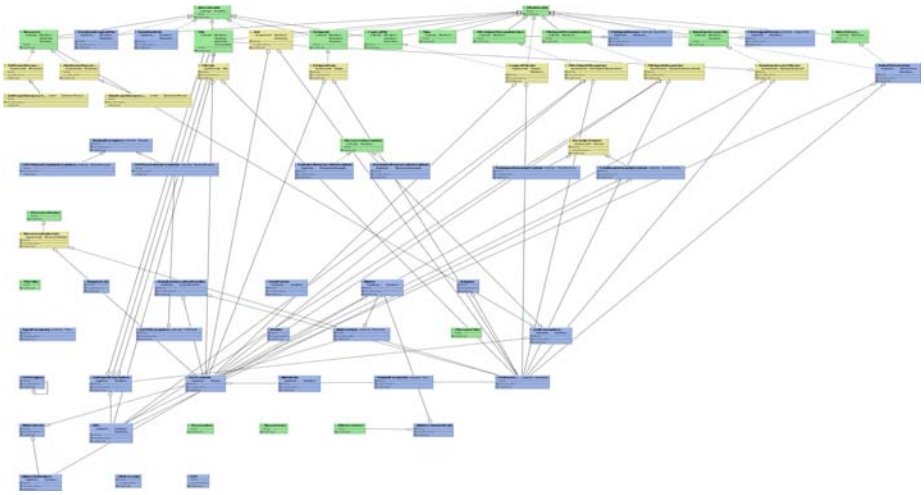
All functions are *protocol and plugin independent*

- **File management**
  - **creating and destroying files**
  - **copying and moving**
  - **examining files**
- **File streams**
  - **reading, writing and seeking on FileStreams (files)**
- **Logical files**
  - **handling of file replicas: creating, destroying, adding, removing and replicating**
- **Advert Management**
  - **creating and destroying of AdvertServices**
  - **adding and deleting Advertiseables**
  - **finding and obtaining Advertisables**

What can GAT do for you? (3)

- Resource management
  - creating and destroying a ResourceBroker
  - creating and destroying a ResourceDescription
  - finding resources and reserving resources
- Interprocess communication
  - creating and destroying Endpoints
  - advertising Endpoints
  - listening and connecting on Endpoints
  - writing and reading via pipes
- Job management
  - creating and destroying jobs
  - scheduling and un-scheduling jobs (adding/removing from queue)
  - stopping, checkpointing, cloning jobs
- Monitoring
  - creating Metrics
  - Listening und Un-Listening to Metrics

Java GAT Interfaces and Classes



Documentation for Java GAT

- Unfortunately, as of October 2005, the documentation of Java GAT is pretty sparse
  - there is a tutorial on the www.gridlab.org
  - for detailed info how things are done in C GAT, you can use the excellent "GAT User's Guide" by Kelly Davis
- Most functionality is explained by examples: ~/JavaGAT/JavaGATEngine/test
  - there directories: advert, examples, file
- My advice: use Eclipse (or other tool with "auto-completion") for exploring sources while your program
- Start your projects by modifying examples

Fundamental JavaGAT Classes and Interfaces

GAT - GAT objects factory

public class GAT (org.gridlab.gat)

- primary factory for creating GAT objects (by static methods)

- main methods (polymorphic variations skipped):
  - createFile
  - createRandomAccessFile
  - createLogicalFile
  - createFileInputStream
  - createFileOutputStream
  - createEndpoint
  - createAdvertService
  - createResourceBroker

GATContext - primary GAT state object

public class GATContext (org.gridlab.gat)

- an instance of this class is the primary GAT state object

- main methods:
  - addPreferences, removePreferences, getPreferences
    - Preferences object in the GATContext is used to choose between adaptors and for adaptor-specific settings
  - addSecurityContext, removeSecurityContext, getSecurityContexts, getSecurityContextsByType
    - adds / manipulates SecurityContexts for this instance

SecurityContext - Holds security information

public abstract class **SecurityContext** (org.gridlab.gat.security)

- A container for security Information. Each context has a data object associated with it. The data object is opaque to the GAT API and is used and manipulated by adaptors based upon their interpretation of the type.

public class PasswordSecurityContext extends SecurityContext

- main methods:
  - public PasswordSecurityContext (String username, String password)
    - constructor, allows for storing username/password

public class CertificateSecurityContext extends SecurityContext

- main methods:
  - public CertificateSecurityContext (URI certificate, String passphrase, URI keyfile)
    - constructor, makes this instance a "certificate" type security context and stores the information about the location of keyfile and certificate file in the context

SecurityContext - restricting access to the context

- Using notes you can restrict access to a set of hosts or adaptors
- Without notes, any adaptor can use context for any host
- Example:

```
GATContext context = new GATContext();
SecurityContext pwd = new PasswordSecurityContext(username, password);

// create notes to restrict access
pwd.addNote("hosts", "hostnameA:portA1,  hostnameB:portB2");
pwd.addNote("adaptors", "ssh");

// add them to the GAT context
context.addSecurityContext(pwd);
```

GAT Preferences – adaptor settings

- Preferences are key/value pairs which allow for
  - determining which adaptors should be used
  - adaptor-specific settings
  - security-related setting
- Global preferences are attached to GATContext, yet can be overriden by local ones
- Example:

```
GATContext context = new GATContext();
Preferences globalPrefs = new Preferences();
prefs.put("File.adaptor.name", "GRAM");
context.addPreferences(globalPrefs);

src = new URI("hello");

// use global preferences
file = GAT.createFile(context, src);

// or use local preferences to override globals:
file = GAT.createFile(context, localPrefs, src);
```

GAT URIs

- GAT URIs are similar to java.net.URI
- Basic schema is protocoll://machine/<path>file
- By leaving some fields blank you can create abbreviations:
  - file:///local-file.txt in current directory
  - file:////local-file.txt in root (/) directory
  - file:///tmp/local-file.txt in /tmp directory
  - ftp://130.73.72.101/remote-file.txt in default ftp directory

- The protocol segment determines the adaptor
  - let JavaGAT engine choose (recommended)
    - any://
  - enforce an adaptor:
    - ftp://, gsiftp://, http://, file://, ...

Manipulating Files in JavaGAT

Manipulating Files

GAT file management offers:

- creating and destroying files
- copying and moving
- examining files

Easy to refactor existing java code – JavaGAT subclasses "java.io" classes

Main interface : File

- Has many adaptor implementations (not important to know – in fact, *you shouldn't*)
  - FTPFileAdaptor
  - GlobusFileAdaptor
  - GridFTPFileAdaptor
  - LocalFileAdaptor
  - SshFileAdaptor
  - ..

Manipulating Files – Interface File

public interface **File** (org.gridlab.gat.io)

- An abstract representation of a physical file. An instance of this class presents an abstract, system-independent view of a physical file. A physical file in GAT is identified by a URI.
- Main methods:
  - public void copy (URI target)
    - copies the physical file represented by this File instance to a physical file identified by the passed URI
  - public void move (URI target)
  - createNewFile, delete, deleteOnExit
  - exists, isDirectory, isFile
  - length, lastModified, canRead, canWrite, ..
  - renameTo, setLastModified, setReadOnly
  - list, listFiles, mkdir, mkdirs

Manipulating Files Example (Remote Copy)

```
public static void main(String[] args) {
    File file;
    GATContext context = new GATContext();
    Preferences localPrefs = new Preferences();
    localPrefs.put("artur",args[0]);          // args[0] contains private key for user artur
    try {
        URI src = new URI("any://localhost/test/file.txt");
        URI dest = new URI("any://artur@csr-pc19.zib.de/GAT/test.txt");
        file = GAT.createFile(context, localPrefs, src);
    } catch (Exception e) {
        System.err.println("File creation failed: " + e); e.printStackTrace();
        System.exit(1);
    }
    try {
        file.copy (dest);
        System.exit(0);
    } catch (Exception e) {
      ...
}
```

Files Example 2 (Listing Directory Contents)

```
public static void main(String[] args) {
    File file;
    GATContext context = new GATContext();
    Preferences localPrefs = new Preferences();
     localPrefs.put("artur",args[0]);          // args[0] contains private key for user artur
    try {
        URI src = new URI("any://localhost/test/");
        File file = GAT.createFile(context, localPrefs, src);
    } catch (Exception e) {
        System.err.println("File creation failed: " + e); e.printStackTrace();
        System.exit(1);
    }
    try {
        String[] names = file.list();
        for (int i = 0; i < names .length; i++)
            System.out.println("    " + names [i]);
    } catch (Exception e) {
      ...
}
```

FileStreams – Reading and Writing Files

FileStreams

GAT FileStream services offer:

- reading, writing and seeking on FileStreams (files)

Easy to refactor existing java code – JavaGAT subclasses "java.io" classes

- Main classes:
  - FileInputStream (org.gridlab.gat.io)
    - extends java.io.InputStream
    - reading in different variations
    - repositioning: skip, mark/reset
  - FileOutputStream (org.gridlab.gat.io)
    - extends java.io.OutputStream
    - writing in different variations
    - flush, close

FileStream Example (Writing File Contents)

```
public static void main(String[] args) {
    GATContext context = new GATContext();
    FileOutputStream out;
    try {
        URI loc = new URI("test/fileStreamExample.txt");
        out = GAT.createFileOutputStream(context, loc);
    } catch (Exception e) {
        System.err.println("FileStream creation failed: " + e);
        System.exit(1);
    }
    try {
        PrintWriter p = new PrintWriter(out);
        p.println("Hello, World!");
        p.close();
    } catch (Exception e) {

      ...
}
```

Advertising Services

Advertising Services

Advert Management

- creating and destroying of AdvertServices
- adding and deleting Advertiseables
- finding and obtaining Advertisables

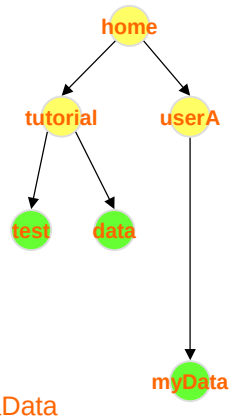
Advertising Services – Advertisable

- Allows arbitrary objects to be made persistent, in order to store or move them across systems
- Any such object must implement a marker interface

```
public interface Advertisable extends
    java.io.Serializable {
      public String marshal();
}
```

Advertising Services – AdvertService

- AdvertService is a container for Advertisables, and supports adding, removing, and querying of them
- It is a directory with hierarchical namespace (aka java.util.Preferences)
- Each Advertisable can have associated MetaData, which is basically a map key -> property, e.g. "name" -> "AdvertisableName"
- Each Advertisable can be found via find(MetaData md) by its MetaData, and also accessed directly by the unique path in the directory via getAdvertisable(String path)



MetaData
↓
Advertisable

Interface AdvertServices

Functions:
- adding and deleting Advertiseables
- finding and obtaining Advertisables

Implementations:
- LocalAdvertServiceAdaptor - in-memory storage, efficient
- StorageBoxAdvertAdaptor – uses Felix Hupfeld's storagebox as backend
- AdvertServiceCpi – persistent, hierarchical meta data directory (not implemented yet)

Methods:
- void add(Advertisable advert, MetaData metaData, String path)
- void delete(String path)
- Advertisable getAdvertisable(String path)
  - use if you know the path in the directory
- MetaData getMetaData(String path)
- String[] find(MetaData metaData)
  - use if you don't know the path in the directory, returns paths
- void setPWD(String path), String getPWD()

Advertiseable and AdvertServices Example

```
public static void main(String[] args) {
    GATContext c = new GATContext();
    Preferences prefs = new Preferences();
    prefs.put("advert.adaptor.name", "storagebox");
    try {
      Endpoint e = GAT.createEndpoint(c);
      AdvertService a = GAT.createAdvertService(c, prefs);
      MetaData m = new MetaData();
      m.put("name", "myEndpoint");
      a.add(e, m, "/tutorial/testadvert");

      Endpoint other = (Endpoint) a.getAdvertisable("/tutorial/testadvert");

      System.err.println("got endpoint back: " + other);
    } catch (Exception x) {..}
```

name in MetaData

path in directory

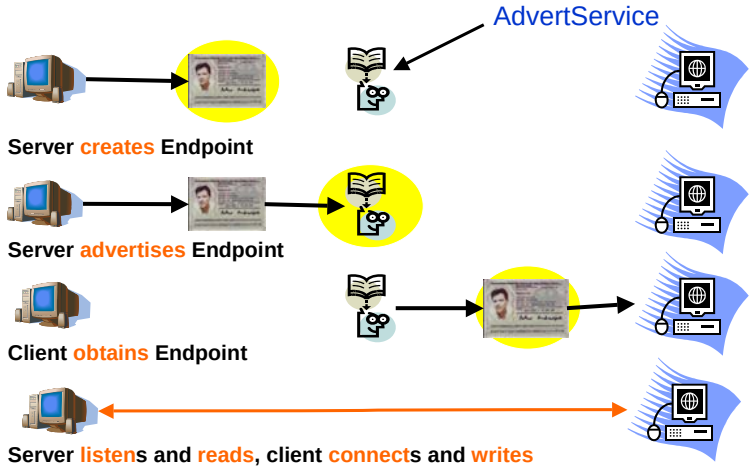
Interprocess Communication

Interprocess Communication

- Interprocess communication
  - creating and destroying Endpoints
  - advertising Endpoints
  - listening and connecting on Endpoints
  - writing and reading via pipes

Main interface: Endpoint (org.gridlab.gat.io)
  - primary abstraction for "the end of a line" (bytestream)
  - implements Adversitable, so can be advertised and found
  - methods:
    - connect()
    - listen()

Interprocess Communication – How to Connect



1. Server creates Endpoint
2. Server advertises Endpoint
3. Client obtains Endpoint
4. Server listens and reads, client connects and writes

Pipe Example – Server (1)

1. Server creates Endpoint
2. Server advertises Endpoint

```
public static void main(String[] args) {
    GATContext context = new GATContext();
    try {
        // 1. create Endpoint
        Endpoint end = GAT.createEndpoint(context);

        AdvertService advert = GAT.createAdvertService(context);
        MetaData meta = new MetaData();
        meta.put("name", "myEndpoint");

        // 2. advertise Endpoint
        advert.add(end, meta, "/home/tutorial/endpoint");
```

Pipe Example – Server (2)

3. --
4. Server listens and reads

```
Pipe pipe = end.listen();                // Blocking call!
InputStream instream = pipe.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(instream));

// Reading the message(s)
// The last message will be 'null'
String szRead;
do {
   szRead = reader.readLine();
   System.out.println("PR: " + szRead);
} while (szRead != null);

// Delete the Endpoint from the AdvertService
advert.delete("/home/tutorial/endpoint");
// Close the stream
reader.close();
```

Pipe Example – Client (1)

1., 2., --
3. Client obtains Endpoint

```
public static void main(String[] args) {
    GATContext context = new GATContext();
    try {
      // Instantiate an Endpoint and the AdvertService
      Endpoint end = GAT.createEndpoint(context);
      AdvertService advert = GAT.createAdvertService(context);
      // Create the data to search the Endpoint in the AdvertService
      MetaData meta = new MetaData();
      meta.put("name", "myEndpoint");

      // Find the data in the AdvertService
      String[] szPaths;
      szPaths = advert.find(meta);
      // Get the first (and only) path from the AdverService and print it.
      end = (Endpoint) advert.getAdvertisable(szPaths[0]);
```

Pipe Example – Client (2)

4. client connects and writes

```
Pipe pipe = end.connect();

OutputStream outstream = pipe.getOutputStream();
PrintWriter writer = new PrintWriter(new OutputStreamWriter(outstream));

// Writing the message(s).
String szWrite = null;
szWrite = "Hello GAT Endpoint...";
writer.println(szWrite);
writer.flush();
szWrite = "... how is life on the other side?";
writer.println(szWrite);
writer.flush();

// Done writing, so close the stream.
writer.close();
```

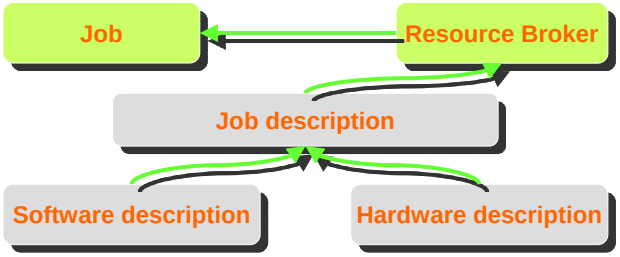
Job Management

Job Management

Job management

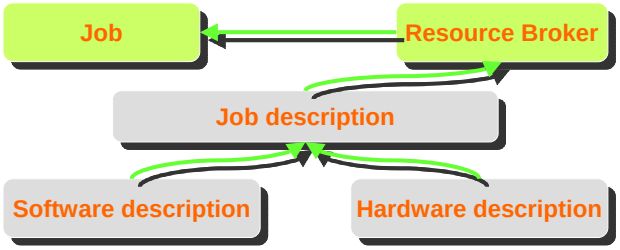
- creating and destroying jobs
- scheduling and un-scheduling jobs (adding / removing from "queue" such as pbs)
- stopping, checkpointing, cloning jobs

- Main components to deal with:



How to Create a Job?

1. Choose the adaptor (currently: local, Globus, GRMS)
2. Create SoftwareDescription sd and HardwareDescription rd
3. Use both (sd and rd) to create JobDescription jd
4. Create ResourceBroker
5. Use the ResourceBroker to submit a Job
6. Use job.getInfo() to obtain the status of the job periodically



Creating a Job – Example (1)

1. Choose the adaptor (currently: local, Globus, GRMS)

```
public static void main(String[] args) {
    GATContext context = new GATContext();
    Preferences prefs = new Preferences();
    prefs.put("ResourceBroker.adaptor.name", "local");
    // prefs.put("ResourceBroker.adaptor.name", "grms");
    // prefs.put("ResourceBroker.adaptor.name", "globus");
```

Creating a Job – Example (2a)

2. Create SoftwareDescription sd ...

sets executable location

location of the executable

```
URI exe;
try {
        exe = new URI("////bin/ls");
        // exe = new URI("file:////WINDOWS/system32/dvdplay.exe");
} catch (URISyntaxException e) { .. }
SoftwareDescription sd = new SoftwareDescription();
sd.setLocation(exe);
try {
        File stdout = GAT.createFile(context, new URI("tutorialTest.txt"));
        sd.setStdout(stdout);
} catch { .. }
```

redirects stdout into a file

Creating a Job – Example (2b)

2. ... and hardware description

resource could be remote (what else should we change for this?)

```
Hashtable hardwareAttributes = new Hashtable();
// hardwareAttributes.put("machine.node", "cluster3.zib.de");
ResourceDescription rd = new
        HardwareResourceDescription (hardwareAttributes);
```

Creating a Job – create JobDescribtion (3)

3. Use both (sd and rd) to create JobDescription jd

```
JobDescription jd = null;
try {
    jd = new JobDescription (sd, rd);
} catch..
```

Creating a Job – create ResourceBroker (4)

4. Create ResourceBroker

```
ResourceBroker broker = null;
try {
        broker = GAT.createResourceBroker (context, prefs);
} catch..
```

Creating a Job – Submitting it to a Resource

5. Use the ResourceBroker to submit a Job

```
Job job = null;
try {
        job = broker.submitJob (jd);
} catch (Exception e) {
```

Creating a Job – Obtaining Job Status

6. Use job.getInfo() to obtain the status of the job periodically

```
while (true) {
        try {
           Map info = job.getInfo();
           System.err.print("job info: ");
           System.err.println(info);
           Exception pe = (Exception) info.get("postStageError");
           if (pe != null)  pe.printStackTrace();
           String state = (String) info.get("state");
           if (state.equals("STOPPED") || state.equals("SUBMISSION_ERROR"))
                   break;
           Thread.sleep(10000);
        } catch (Exception e) { .. }
}
```

Job Management – What else?

- Different batch systems can be used

```
prefs.put("ResourceBroker.adaptor.name", "globus");
prefs.put("ResourceBroker.jobmanager", "pbs");
```

- Redirecting stdIn, stdOut, stdErr
  - SoftwareDescription sd = new SoftwareDescription();
  - sd.setLocation(exe);
  - sd.setStdin(inFile);
  - sd.setStdout(outFile);
  - sd.setStderr(errFile);

Job Management – What else? (2)

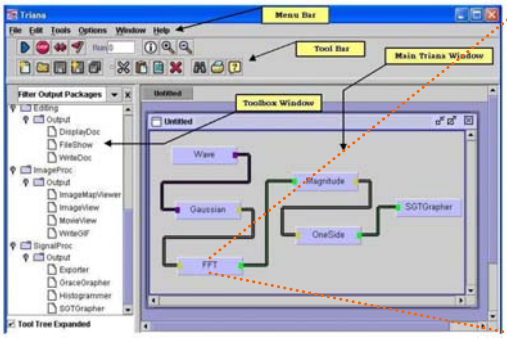
- Instead of actively monitoring (polling), you can use *callbacks* for status change tracking

```
public synchronized void ProcessMetricEvent (MetricValue val) {
   notifyAll();
}
..
   Job job = broker.submitJob(jd);
   MetricDefinition md = job.getMetricDefinitionByName ("job.status");
   Metric m = md.createMetric(null);
   job.addMetricListener (this, m);
   synchronized (this) {
            while (job.getState() != Job.STOPPED
                                 && job.getState() != Job.SUBMISSION_ERROR)
                        wait();
   }
```

Triana Workflow Programming System

What is Triana?

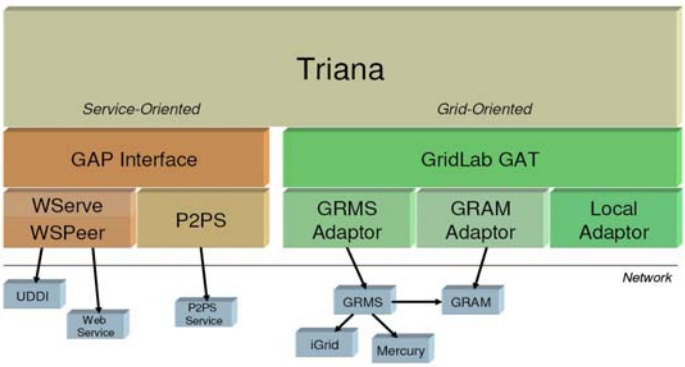
- Triana is a graphical environment that allows you to create workflow-like computer programs and to run them locally or in a distributed way
- You assemble workflows via drag-and-drop from units which are supplied or own java classes
- Writing own units is easy, and there are wizards for creating units and their GUIs
- Triana has been developed at University of Wales, Cardiff, UK, and is public domain software
- Download it frow www.trianacode.com (requires Java 1.4.2 or later)




```
public class FFT extends OldUnit {

   String style = "Automatic";
   String WindowFunction = "(none)";
   boolean padding = false;
   String opt = "MaximumSpeed";
   FFTC fft;

    /**
     ***********************************************
     *** A Java FFT algorithm    ***
     ***********************************************
     */
   public void process() throws Exception {
      GraphType result = null;
      GraphType input;

         int points = 1;
         int points0 = 1;
         int kk, j;
         int targetN = 1;
         double sf=0;
         double sf0 = 0;
         ...
```

Distributed computing with Triana

- Triana has two ways of writing parallel applications
  - Web-Service-oriented via the (proprietary) GAP interface and several implementations
  - Grid-oriented components (units) using JavaGAT (our focus)



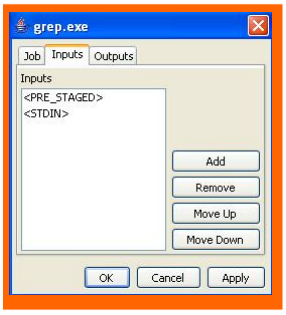
Accessing JavaGAT in Triana

- **Basically, we need only three types of units**
  - **local, "normal" processing units**
  - **Job – local or remote job submission/execution (/Triana Tools/Common/)**
  - **File – local or remote files (/Triana Tools/Common/)**
- **The connections between those units "polymorphically" indicate the required action**

Source	Target	Action
File	File	file transfer or local copy
File	local unit	reading from a file for local processing
local unit	File	writing a local output to a file
File	Job	pre-staging (automatic file transfer *before* execution)
Job	File	post-staging (automatic file transfer *after* execution)

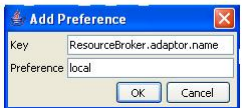
The unit Job

- **The unit "Job" allows for specifying input files, output files, and job settings**
- **Input: pre-staged files, stdin**
- **Ouput: post-staged files, stdout, stderror**
- **Job: host address, binary, arguments**



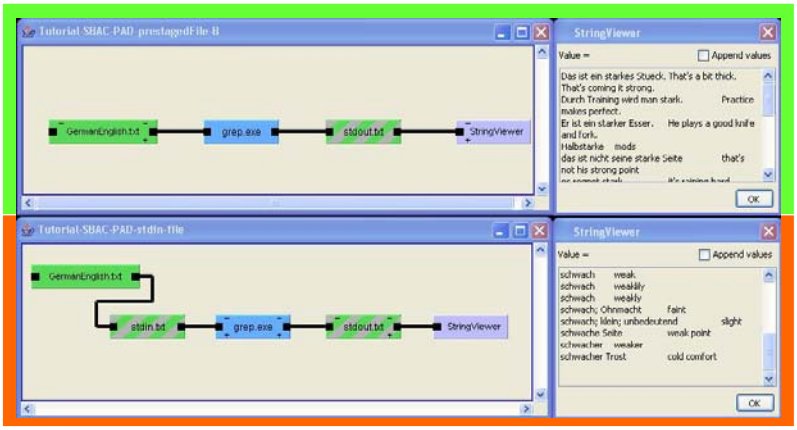
Unit file / JavaGAT settings

- The unit "file" only needs to know the path/name of the file, and whether it is temporary
- In menu Services/Preferences you can set JavaGAT preferences, just as in "Preferences" object from Part I



Example – running Grep on a remote machine

- We run "grep" on a remote machine, with two variations
  1. grep <pattern> <staged-in-file.txt>
  2. grep <pattern> (input from stdin)



Example – running "grep" on a remote machine

<Demo>

What about other JavaGAT parts in Triana?

- Currently, no support for other JavaGAT packages (resource management, interprocess communication etc.)
- However, Triana is still under development so expect those parts soon

Installing JavaGAT and Hands-On Exercises

(Linux and sh, zsh)

Installing Java GAT (Building)

1. If not installed, install Java (need 1.4.2 or higher for GAT) and add to path
   - $ export PATH=$PATH:*<java bin directory>*
2. If not installed, install Ant and add to path
   - $ export PATH=$PATH:*<ant bin directory>*
3. Download Java GAT 1.2 from www.gridlab.org into the working directory ~/JavaGAT
4. Build the GAT engine
   - $ cd *~/JavaGAT/JavaGATEngine*
   - $ ant
5. Build the GAT adaptors
   - but first set the property "engine_path" in build.xml to the adaptors directory to tell the adaptors where the engine lives!
   - $ cd *~/JavaGAT/JavaGATAdaptors*
   - $ ant

Installing Java GAT (Tutorial and Paths)

6. Download and unzip the archive from www.zib.de/andrzejak/GT into the directory ~/JavaGAT/Tutorial

7. Make the files run_gat_app and compile_gat_app in ~/JavaGAT/Tutorial executable
   $ chmod u+x run_gat_app compile_gat_app

8. .. and possibly edit them by setting the correct paths:
   - GAT_ENGINE_LOCATION=$HOME/JavaGAT/JavaGATEngine
   - GAT_ADAPTOR_LOCATION=$HOME/JavaGAT/JavaGATAdaptors

9. Add the directory ~/JavaGAT/Tutorial to your PATH:
   - $ export PATH=$PATH:$HOME/JavaGAT/Tutorial

10. Test with the "Hallo GAT world" program
    $ cd ~/JavaGAT/Tutorial
    $ compile_gat_app HalloGATworld.java
    $ run_gat_app HalloGATworld

Setting up the SSL Keys

1. Create a pair private/public key using ssh-keygen
   $ ssh-keygen -t dsa -f *keyfile*
   Generating public/private dsa key pair.
   Enter passphrase (empty for no passphrase): <press enter>
   Enter same passphrase again:<press enter>
   Your identification has been saved in keyfile.
   Your public key has been saved in keyfile.pub.
2. Copy the keyfile into the working directory of your installation
   $ cp *keyfile ~/JavaGAT/Tutorial*
3. On the remote machine, copy over the keyfile.pub file and append it to ~/.ssh/authorized_keys
   $ ftp *remote-host*
   $ (..)
   $ put keyfile.pub
   $ quit
   $ ssh remote-host
   $ (..)
   $ cat keyfile.pub >> ~/.ssh/authorized_keys

More Info:
www.gridlab.org
www.trianacode.org

Thank you!