A Globus Toolkit Introduction

Developer's Overview

Peter Praxmarer

praxmarer@gup.jku.at

GUP Linz

Johannes Kepler Universität Linz

Austria

Agenda

1. Grid Computing?
2. Key Concepts
3. Globus Toolkit 2 Components (Developer’s View)
    - Resource Management
    - Data Management
    - Security
    - Common Runtime Components
    - Information Services
4. Outlook to OGSA
5. Links

Grid Computing: Basic Definitions

Grid “A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to highend computational capabilities.” (Ian Foster, 1999)

Virtual Organization Is a

- Set of entities (persons, organizations)
- Sharing their resources
- Temporarily
- Controlled

Why Grid Computing?

- Exploiting underutilized resources
- Parallel CPU capacity
- Virtual Organizations for collaboration
- Access to special/additional resources
- Reliability

Grid Middleware: The Globus Project

- Basic research in grid-related technologies
- Development of the Globus Toolkit
- Construction of Testbeds
- → Aims at providing a production-ready *grid middleware*

The Globus Toolkit

- “Bag of services”
- Layered Architecture:

Grid Fabric Layer Information hiding of basic OS services (IO, libc, threads, ...)

Grid Services Layer Basic Grid services for *resource management*, *data management*, *information services*, and *security*

Application Toolkit Layer Specialized services for various problem domains

Application Layer Grid applications using the services of the underlying layers.

Globus Toolkit 2 Components

- Security (authentication, authorization, integrity, confidentiality) → *GSI*
- Remote process invocation/execution → *GRAM*
- Data Management → *GridFTP, GASS*
- Information Services → *MDS*

→ Encapsulated in so-called *modules*

Globus Modules

- A *Globus module* is a software unit encapsulating those functions which logically belong together.
- Five main components
  - Resource management
  - Data management
  - Information services
  - Security
  - Common Libraries

GT2 Development Basics(1)

- Flavors: The Globus Toolkit components can be installed using different 'flavors'. Flavors define the
  - Compiler: gcc, vendorcc, mpicc
  - Architecture: 32, 64
  - Debug-Information
  - Threading: pthread, no-thread
- e.g. `gcc32dbgpthr`
- When compiling your own application NEVER mix different flavors!! (e.g. `gcc -I/opt/globus/include/gcc32dbg -L/opt/globus/lib myapp.c -lglobus_common_gcc32dbgpthr`)
- → use `globus-makefile-header` instead

GT2 Development Basics(2)

- `globus-makefile-header`: Prints all Globus relevant paths and tools; should be included (and used) in the project's makefile
- Example usage:

```
globus-makefile-header --flavor gcc64pthr
> globus_makefile_header.mk
```

GT2 Development Basics(3)

- Include the output in the project's makefile:

```
# Project Makefile
include globus_makefile_header.mk

all: myapp

%.o: %.cc
    $(GLOBUS_CXX) -g -c -I. $(GLOBUS_CPPFLAGS) $< -o $@

myapp: myapp.o
    $(GLOBUS_CXX) -g -o $@ $(GLOBUS_CPPFLAGS) \
    $(GLOBUS_LDFLAGS) $ˆ $(GLOBUS_PKG_LIBS)
```

Globus Module Activation(1)

- Before a Globus-Module can be used it needs to be activated `globus_module_activate()`
- After it’s use it should be deactivated `globus_module_deactivate()`
- Dependent modules are automatically activated
- Modules can be loaded more than once

Globus Module Activation(2)

- Typical pattern:

```
#include ‘‘globus_common.h’’
#include ‘‘globus_io.h’’

int main() {
  // ...

  globus_module_activate(GLOBUS_IO_MODULE);

  // ... (use)

  globus_module_deactivate(GLOBUS_IO_MODULE);

  // ...
}
```

Globus Error Reporting

- Globus functions typically return a `globus_result_t` object
- On success: `GLOBUS_SUCCESS`
- Otherwise the error can be printed using the functions:

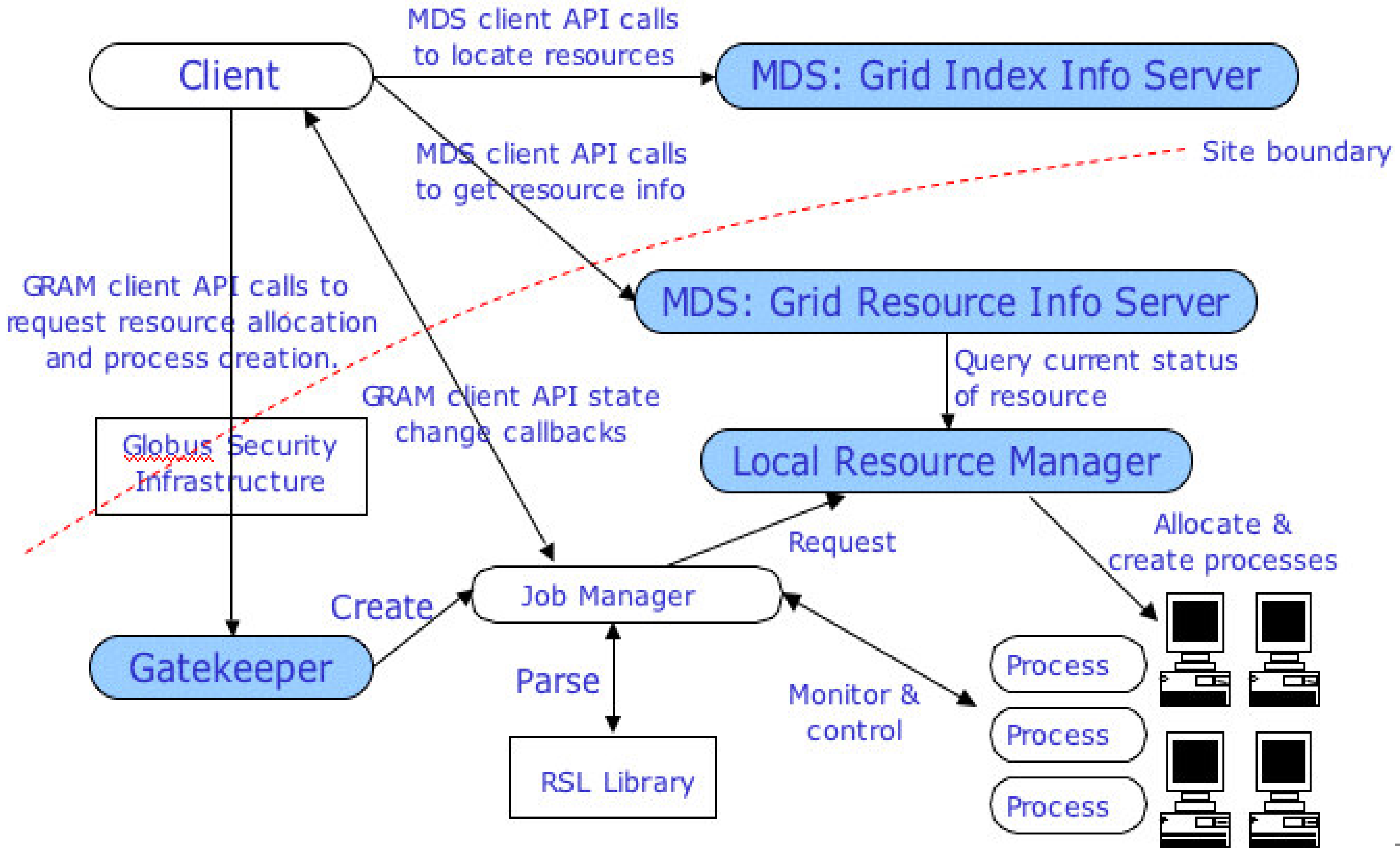
`globus_error_get()` Takes a `globus_result_t` structure and returns a `globus_object_t*` object.

`globus_object_printable_to_string()` Takes a `globus_object_t*` and returns a zero-terminated C-string

Resource Management: GRAM

- Grid Resource Allocation Management (GRAM)
- Resource Specification Language (RSL) is used to communicate requirements
- GRAM provides remote invocation

GRAM Components


Client
MDS client API calls to locate resources
MDS: Grid Index Info Server
MDS client API calls to get resource info
Site boundary
GRAM client API calls to request resource allocation and process creation.
MDS: Grid Resource Info Server
Query current status of resource
GRAM client API state change callbacks
Globus Security Infrastructure
Local Resource Manager
Request
Allocate & create processes
Create
Job Manager
Gatekeeper
Parse
Monitor & control
Process
Process
Process
RSL Library

Resource Specification Language (RSL)

- Common language for specifying a jobs needs
- Requirements are specified by a conjunction of (key=value) pairs

```
&(executable="/bin/ls")
(count=4)
(arguments="-la")
```

- Basic form of each pair: `(attribute op value [ value ...])`
- Operators (op): $<, <=, =, >=, >, != $
- GRAM understands a well-defined set of attributes
- Unknown attributes are passed through

Constraints: “&”

- “Create 5-10 instances of myprog, each on a machine with at least 64 MB memory that I can use for 4 hours”
- Can be expressed using:

```
&(count>=5)(count<=10)
(max_time=240)(memory>=64)
(executable=myprog)
```

Disjunction: “|”

- “Create 5 instances of myprog, each on a machine with at least 64 MB of memory, or 10 instances on machines with at least 32 MB of memory”
- Can be expressed using:

```
&(executable=myprog)
  (|(&(count=5)(memory>=64))
    (&(count=10)(memory>=32))
  )
```

RSL Attributes(1)

`(executable=string)` Program to execute

`(directory=string)` Current/working directory

`(arguments=arg1 arg2 arg3 ...)` Argument list

`(environment=(E1=v1)(E2=v2) ...)` Environment variable list

`(stdin=string)` Stdin for the program; Can be a file path or an URL

`(stdout=string)` Stdout for the program; Can be a file path or an URL

`(stderr=string)` Stderr for the program; Can be a file path or an URL

RSL Attributes(2)

`(count=integer)` Number of processes to run

`(hostCount=integer)` Number of nodes to distribute the 'count' processes across

`(project=string)` Project against which to charge

`(queue=string)` Queue into which to submit the job

`(maxTime=integer)` Maximum wall clock or CPU runtime (scheduler's choice) in minutes

`(maxWallTime=integer)` Maximum wall clock runtime in minutes

RSL Attributes(3)

`(maxMemory=integer)` Maximum amount of memory for each process in megabytes

`(minMemory=integer)` Minimum amount of memory for each process in megabytes

`(jobType=value)` Values are:

- mpi: Job is an mpi-program, thus it is started using `mpirun -np <count>`
- single: Only run a single instance of the program, and let the program start the other <count>-1 processes
- multiple: Start <count> instances of the program using the appropriate schuduler mechanism
- condor: Start <count> Condor processes running in “standard universe”

RSL Attributes(4)

`(gramMyjob=value)` Defines how the `globus_gram_myjob` library will operate on the <count> processes:

- **`collective`** Treat all <count> processes as part of a single job
- **`independent`** Treat each of the <count> processes as an independent uniprocessor job

`(dryRun=true)` Do not actually run the job

RSL Attributes(5)

`(save_state=yes)` Jobmanager (should) save the job state to disc, in order to recover after a jobmanager's crash

`(two_phase=integer)` Implement a two-phase commit for job submission and completion; time out after <int> seconds

`(restart=<old jobmanager contact>)` Start a new jobmanager but instead of submitting a new job, start watching over an existing job

`(std[out|err]_position=integer)` Specified as part of a job restart

- Restart file streaming from this byte

RSL Substitutions

- RSL supports simple variable substitutions
- Defined as a list of pairs:

  ```
  (rslSubstitution=(SUBST1 val1)(SUBST2 val2))
  ```

- Applied by `$(SUBST)`
- Processing order:
  1. Within scope, processed left-to-right
  2. Outer scope is processed before inner scope
  3. Variable definition can refer previously defined variables

RSL Substitution: Example

- This

```
&(rslSubstitution=(URLBASE ‘‘ftp://whereever’’))
(rslSubstitution=(URLDIR $(URLBASE)/dir))
(executable=$(URLDIR)/executable)
```

- is equivalent to

```
&(executable=ftp://whereever/dir/executable)
```

RSL: Predefined Substitutions

- GLOBUS_HOST_MANUFACTURER
- GLOBUS_HOST_CPUTYPE
- GLOBUS_HOST_OSNAME
- GLOBUS_HOST_OSVERSION
- GLOBUS_LOCATION
- HOME
- LOGNAME
- GLOBUS_ID

globus_rsl

- Functions for manipulating RSL expressions
  - Parse a RSL into a data structure
  - Manipulate it
  - Unparse the data structure into a string

GRAM: Tools

`globus-job-run` Submits a job to a gatekeeper.

- e.g. `globus-job-run hydra.gup.uni-linz.ac.at /bin/hostname`
- Lots of options
- Dump the RSL string with `-dumprsl`

`globusrun` Takes a RSL expression and submits it to the specified resource

- `globusrun -r hydra.gup.uni-linz.ac.at ’&(executable=/̈bin/hostname)̈’`
- `globusrun -r hydra.gup.uni-linz.ac.at -f job.rsl`

GRAM: Submitting a MPI Job

```
+
( &(resourceManagerContact="hydra")
  (count=8)
  (maxtime=60)
  (jobtype=mpi)
  (label="subjob 0")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
    (LD_LIBRARY_PATH /opt/globus/lib))
  (directory=/home/guppy/pprax)
  (executable=/home/guppy/pprax/mpiprogram)
  (stdout=/home/guppy/pprax/mpiprogram.out)
  (stderr=/home/guppy/pprax/mpiprogram.err)
)
```

globus_gram_client - API

- Developer’s interface for job submission

GRAM: Submitting jobs - Basics

1. Activate the GLOBUS_GRAM_CLIENT_MODULE
2. Create a client callback using
   `globus_gram_client_callback_allow()`
3. Request a job being executed using
   `globus_gram_client_job_request()`
4. Destroy the client callback using
   `globus_gram_client_callback_disallow()`
5. Deactivate the GLOBUS_GRAM_CLIENT_MODULE

globus_gram_client - API

`globus_gram_client_job_request()` Submit a job to a remote resource

- Resource manager contact string (in)
  - *hostname[:port][/service][:subject]*
  - hostname: required
  - port: defaults to 2119
  - service: defaults to *jobmanager*
  - subject - security subject name of the gatekeeper (e.g. returned by `grid-cert-info -subject -f /etc/grid-security/hostcert.pem`)
- RSL string (in)
- callback contact string (in); previously created by calling `globus_gram_client_job_request()`
- job contact string (out)

Job Contact String

- Returned by `globus_gram_client_job_request()`
- Identifies the job
- Is used by subsequent `globus_gram_client_*()` function calls
- The job contact string can be passed between processes, even on different machines

More GRAM functions

globus_gram_client_job_status() Check the status of the job (one of PENDING, ACTIVE, FAILED, DONE); Status can also be tracked through callbacks!

globus_gram_client_job_cancel() Cancel/kill a pending or active job

Others Not discussed here! See http://www.globus.org/gram/client/function_reference.html

GRAM Example

- GlobusGramClientExample.cc
- Allow gram client callbacks
- Submit the job, registering a callback function
- Track state changes using the registered callback function

Resource Management APIs

- globus_rsl
- globus_gram_client
- globus_gram_myjob
- globus_duroc_control
- globus_duroc_runtime

Data Management

- Data transfer and access

  GASS Simple, multi-protocal file transfer tools; integrated with GRAM

  GridFTP Enhanced FTP protocol

- Data replication and management

  Replica Catalog Provides a catalog service for keeping track of replicated datasets

  Replica Management Provides services for creating and managing replicated datasets

  Not further discussed here!

Global Access to Secondary Storage

- Short: GASS
- Used by GRAM for:
  - File staging:
    - Pull executables from remote location
    - Move stdin/stdout/stderr from/to a remote location
- Provides:
  - GASS file access API
  - Remote cache management utility
  - API to implement special behavior
- Most effectively used for ’small’ files

GASS-APIs

Globus GASS File Access Wraps the basic Unix file operations

GASS Client API Is used to make get and put requests to ftp and x-gass servers.

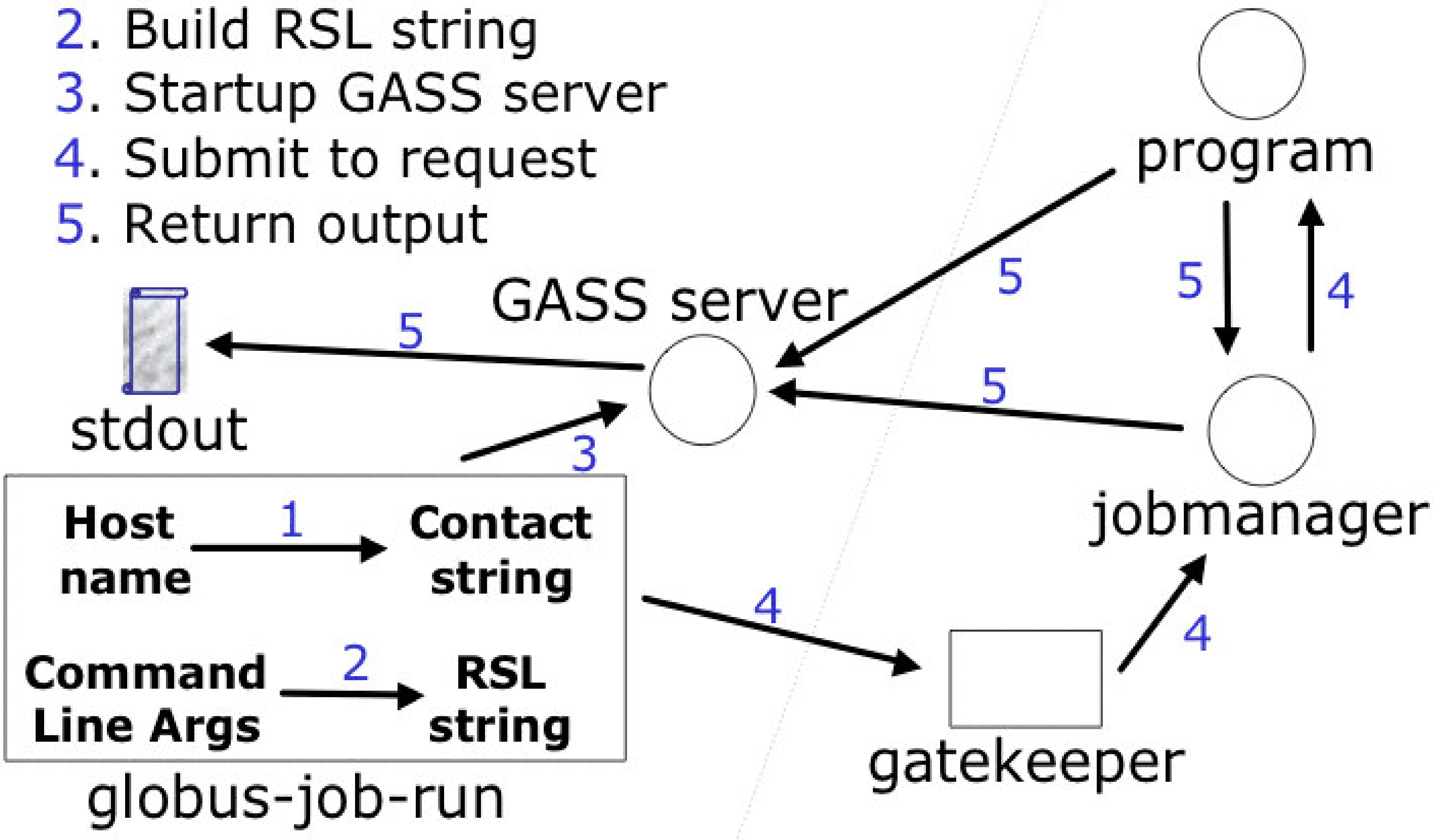
GASS Server-EZ API Is used to construct a server that can service get and put request made by the GASS client API using x-gass URLs.

GASS Server API Same as Server-EZ but low-level

GASS Cache Management API The GASS cache management API defines calls for manipulating a local file cache.

GASS and GRAM Interaction

1. Derive Contact String
2. Build RSL string
3. Startup GASS server
4. Submit to request
5. Return output



GASS - Example

GlobusGassServerGramSubmission.cc

GridFTP: Characteristics

- FTP protocol with various extensions:
  - GSI-based security
  - Stripped data transfer
  - Parallel data transfer
  - Partial file transfer
  - 3rd party transfer
  - Automatic/manual TCP buffer setting
- → Aims at high-performance data transfer for large datasets

GridFTP APIs

globus_ftp_control Provides access to low-level GridFTP control and data channel operations

globus_ftp_client Provides typical client operations (get, put, ...)

globus_gass_copy Convenience API for managing multiple data transfers using GridFTP, HTTP, local file, and memory operations

GridFTP: Tools

`globus-url-copy` Copies a source URL to a destination URL

- Valid protocols are http, https, FTP, gsiftp, and file
- Uses the *globus_gass_copy* API.
- Example usage:

```
globus-url-copy file:///tmp/file1
gsiftp://hydra/tmp/file2
```

GridFTP: Example

GlobusGridFTPClient.cc

Common Runtime Components

- Wraps various C libraries for portability (GLOBUS_COMMON_MODULE)
  - globus_libc
  - globus_thread
  - globus_list
  - globus_fifo
  - globus_hashtable

Security: Grid Security Infrastructure

- Public Key Infrastructure (PKI)
  - Certificate Authorities (CAs)
  - Certificates
- SSL for authentication and message protection
- *Proxies* and delegation for secure single Sign-on
- Programmed using the *Generic Security Services API (GSS-API)*

Security: Public Key Infrastructure

- Asymmetric encryption:
    - Each entity has a **public** and a **private** key
    - The private key is only known to the entity
    - Allows encryption
    - Allows authentication
- Public key is encapsulated in a X.509 certificate

PKI: Certificates

- Binds a *public key* to a *name*
- Is *signed* by a *trusted party*

(Thus the certificate contains at least **Name, Issuer, Public Key, and Signature**)

PKI: Certificate Authorities

- Small set of trusted entities
- Exists only to sign user certificates
- CA signs it’s own certificate which is then is distributed in a trusted manner.

→ Public Key from CA is used to verify other certificates

PKI: Requesting a Certificate

1. `grid-cert-request` generates a key pair
2. The *private key* is stored encrypted with a pass phrase
3. The *public key* is put into a *certificate request*
4. The *certificate request* is sent to the CA
5. The CA verifies the request (Is the name unique with respect to the CA?, Is the name stored in the certificate the real name of the user?, ...)
6. The CA signs the certificate request and issues a certificate for the user

GSI: Tools (1)

grid-cert-request Request a user certificate, host certificate, or ldap certificate

grid-cert-info Get certificate information:

- `-all`
- `-subject`
- `-issuer`
- `-startdate`
- `-enddate`
- `-help`

GSI: Tools (2)

grid-proxy-init Creates a *user-proxy* that

- is used for authentication with other resources
- has limited validity
- ’acts on behalf of the user’

→ User’s private key is not exposed after proxy has been signed

grid-proxy-info Displays proxy details

grid-proxy-destroy Destroys the user-proxy previously created by `grid-proxy-init`

GSI: Files (1)

- /etc/grid-security

  hostcert.pem server certificate (used for authentication with gatekeeper, gsiftp)

  hostkey.pem server's private key

  grid-mapfile maps grid subjects to local user accounts

- /etc/grid-security/certificates

  CA certificates CAs that we trust

  ca-signing-policy.conf defines the subject names that can be signed by each CA

GSI: Files (2)

- $HOME/.globus

  usercert.pem User’s certificate

  userkey.pem User’s private key (encrypted by a passphrase)

- /tmp

  Proxy file(s) Temporary file(s) containing the **unencrypted** proxy private key and certificate; valid only for a ’short’ period

GSI: Delegation

Delegation Remote creation of a proxy credential

1. New key pair is generated remotely on server
2. Proxy certificate + public key sent to client
3. Client signs the proxy certificate
4. Server stores it in /tmp

Types Various types

- Full proxy
- Limited proxy
- Restricted proxy

GSI Programming: GSS-API

- The *Generic Security Service API* is the IETF draft for adding authentication, delegation, message integrity, and message confidentiality to apps
- Seperates security from communication
- Globus Toolkit components use the GSS-API
- But GSS-API is not easy to use: Thus GT 2 provides the `globus_gss_assist` module, which is a wrapper around GSS-API. It’s use is demonstrated later.

Summary: Grid Security Infrastructure

- Lies at the heart of all *Globus Components*
- Uses a Public Key Infrastructure
- Provides a secured TCP connection using SSL
- Can be programmed using the GSS-API or the *globus_gss_assist* module.

Globus IO

- Provides I/O for:
  - Files
  - TCP
  - UDP
- Integrates GSI-Security
- Blocking/Nonblocking

Globus IO: Basic Steps(1)

1. Activate the `GLOBUS_IO_MODULE`
2. Initialize used data structures: `globus_io_attr_t`, using `globus_io_tcpattr_init()`, `globus_io_udpattr_init()`, or `globus_io_fileattr_init()`
3. Create the globus_io_handle_t by calling one of `globus_io_file_open()`, `globus_io_tcp_create_listener()`, `globus_io_tcp_accept()`, `globus_io_tcp_connect()`, `globus_io_udp_bind()`, or use the non-blocking calls

Globus IO: Basic Steps(2)

4. Use the created handle by calling one of *globus_io_[register_]read()*, *globus_io_[register_]write()*, ...
5. Close the handle with *globus_io_close()*, free the allocated memory for the previously initialized data structures, using `globus_io_tcpattr_destroy()`, `globus_io_udpattr_destroy()`, or `globus_io_fileattr_destroy()`

Globus IO: TCP Examples

- GlobusTCPClientTestApp.cc
- GlobusTCPServerTestApp.cc

Information Services - Motivation

- Repository containing answers to questions like:
  - What resources are available?
    - → resource discovery
  - What is the ’state’ of the grid?
    - → resource selection
  - How can the resource use be optimized?
    - → application configuration and adaption
- → Metacomputing Directory Service (MDS)

MDS: Characteristics

- Provides uniform access to static and dynamic information regarding system components
- Basic information for configuration and adaption
- Scalable
- Decentralized maintainance

Metacomputing Directory Service

- Uses LDAP
- Directory is represented by collection of LDAP servers
- Updated by:
  - Information providers
  - Applications
  - Backend tools that generate info on demand
- Information dynamically available to tools and applications

MDS: Architecture

- Two main components:
  - Grid Resource Information Service (GRIS): Supplies information about a specific resource
  - Grid Index Information Service (GIIS): Supplies collection of information previously gathered from multiple GRIS, or GIIS
- Protocols:
  - Grid Resource Registration Protocol (GRRP)
    - Support information/resource discovery
  - Grid Resource Inquiry Protocol (GRIP)
    - Query resource description server for information
    - Query aggregate server for information
    - LDAP v3.0

LDAP Overview

- Lightweight Directory Access Protocol
- IETF Standard
- ’Directory’: Listing of information about objects arranged in some order that give details about each object
- ’Lightweight’: Doesn’t support the full OSI protocol stack as required by the original X.500 (DAP) standard; uses the TCP/IP protocol stack instead
- Organizes directory entries in a hierarchical name space capable of supporting large amounts of information

Querying MDS

grid-info-search General purpose client

- `grid-info-search -h <host> -p <port> -b <base> -T <timeout> [<filter>] [<attributes>]`
- `-x` Anonymous access
- Example:

  ```
  grid-info-search -b
  ’Mds-Device-Group-name=processors,
  Mds-Host-hn=hydra.gup.uni-linz.ac.at,
  Mds-Vo-name=JKU,o=Grid’
  Mds-Cpu-Total-count
  ```
- Standard port is 2135

Querying MDS from an application

- MDS is accessed using the OpenLDAP client library
- Provides functions for:
  - Connecting to server
  - Posing queries which return data structures containing the search result
  - Functions for traversing these data structures

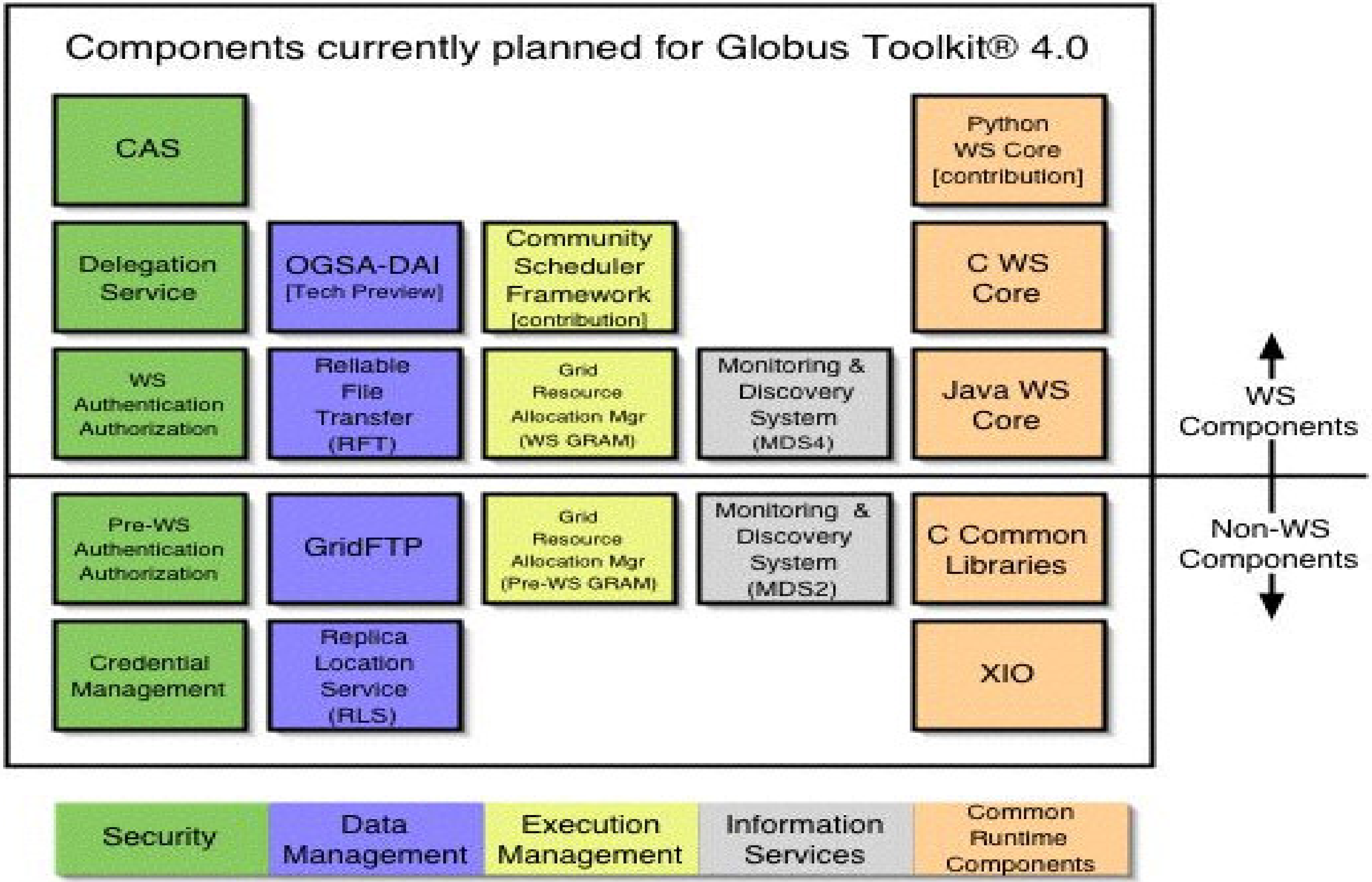
Example

MDSClientExample.cc

Outlook to GT 4 - OGSA

- Open Grid Service Architecture
- Service Orientation to virtualize resources
- Built on GT2
- Uses Web services
  - Standards-based framework for accessing network applications; W3C standardization
  - WSDL: Web Services Description Language
  - SOAP: Simple Object Access Protocol (XML-based RPC protocol)
  - UDDI: Universal Description, Discovery, and Integration (Directory for Web services)

Components in GT4



Components currently planned for Globus Toolkit® 4.0

CAS

Python WS Core [contribution]

Delegation Service

OGSA-DAI [Tech Preview]

Community Scheduler Framework [contribution]

C WS Core

WS Authentication Authorization

Reliable File Transfer (RFT)

Grid Resource Allocation Mgr (WS GRAM)

Monitoring & Discovery System (MDS4)

Java WS Core

WS Components

Non-WS Components

Pre-WS Authentication Authorization

GridFTP

Grid Resource Allocation Mgr (Pre-WS GRAM)

Monitoring & Discovery System (MDS2)

C Common Libraries

Credential Management

Replica Location Service (RLS)

XIO

Security

Data Management

Execution Management

Information Services

Common Runtime Components

New Acronyms

- Security

  CAS Community Authorization Service; Allows a VO to express a policy regarding resources distributed across a number of sites

- Data Management

  RFT Reliable File Transfer; The RFT service uses standard SOAP messages over HTTP to submit and manage a set of 3rd party GridFTP transfer and to delete files using GridFTP.

- Execution Management
- Information Services
- Common Runtime Components

Links

- http://www.globus.org
- http://www.globus.org/developer/api-reference.html