DRMAA: Distributed Resource Management Application API

Peter Tröger,
Hasso-Plattner-Institute (HPI) @ University of Potsdam

GGF 14 DRMAA session

Chicago, June 28, 2005

Agenda

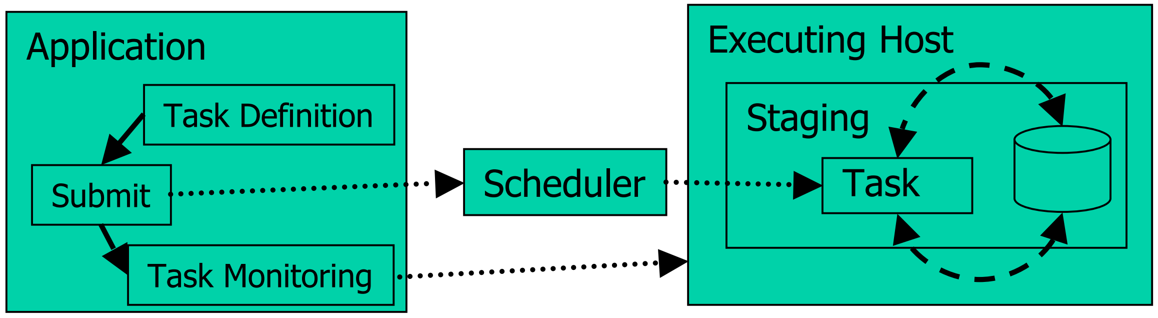
- First things first
  - GGF IP
  - Sign-up sheet
  - Note takers
- DRMAA introduction
- Status of the DRMAA implementations
- Status of the DRMAA documents
- Open floor, open issues

Scope: Run a Job API

(Steps from: Ten Actions when SuperScheduling", GGF SchedWD 8.5, J.M. Schopf, July 2001)

- **Phase 1: Resource Discovery**
  - Step 1 Authorization Filtering
  - Step 2 Application requirement definition
  - Step 3 Minimal requirement filtering
- **Phase 2 System Selection**
  - Step 4 Gathering information (query)
  - Step 5 Select the system(s) to run on
- **Phase 3 Run job**
  - Step 6 (optional) Make an advance reservation
  - **Step 7 Submit job to resources**
  - Step 8 Preparation Tasks
  - **Step 9 Monitor progress (maybe go back to 4)**
  - **Step 10 Find out Job is done**
  - Step 11 Completion tasks

Resource Management Systems Differ Across Each Component



	Interface Format	Execution Environment	Platform Mix
LSF	Has API plus Batch Utilities via “LSF Scripts”	User: Local disk exported System: Remote initialized (option)	Unix ←/ → Windows
Grid Engine	GDI API Interface plus Command line interface	System: Remote initialized, with SGE local variables exported	Unix only
PBS	API (script option) Batch Utilities via “PBS Scripts”	System: Remote initialized, with PBS local variables exported	Unix ←→ Windows
DataSynapse	Proprietary API.	User: Remote initialized	Unix ← → Windows

DRMAA Charter

- Develop an API specification for **the submission and control of jobs** to one or more Distributed Resource Management (DRM) systems.
- The scope of this specification is all the high level functionality which is necessary for an application to consign a job to a DRM system including **common operations on jobs** like termination or suspension.
- The objective is to **facilitate the direct interfacing of applications to today's DRM systems** by application's builders, portal builders, and Independent Software Vendors (ISVs).

DRMAA History

- BOF at GGF 3 in Frascati, Oct 2001
- WG status at GGF 4, Toronto, February 2002

- Participation from Altair (PBS), Sun Microsystems (SGE), Intel, IBM (LoadLeveler), University of Wisconsin (Condor), Cadence (Rocks system), Globus project
- Sideline engagement from EnFuzion, Entropia, Platform (LSF), GridIron project, United Devices

- June 2004: DRMAA 1.0 document accepted as proposed recommendation by GFSC

What have been the Issues?

- **General features**
  - Session concept
  - Asynchronous job monitoring
  - Scalability
  - Native features
- **Language bindings**
  - C/C++
  - Perl, Python
  - Fortran, Java
- **Libraries**
  - Serial / thread safe
  - Tracing / diagnosis
- **Advanced features**
  - Debugging support
  - File staging
  - Security
  - Job categories

Submit, control & monitor, and query status of jobs

DRMAA API Function Groups

- Init / Exit
- Job template handling
  - Allocation / Deletion
  - Job template parameter setter/getter routines
- Job submission
  - Individual jobs
    - One time
    - Multiple times – just re-adjust the job template (parameter sweep)
  - Bulk jobs - implicit parameterization
- Job monitoring and control
- Auxiliary or system routines
  - Error message routines
  - Informational interfaces

Job Template

- Description of all job requirements / parameters
- Mandatory and optional parameters
- Same intention as JSDL, but designed as ‘smallest common denominator’ between possible backend’s
- Functions to create/delete job templates
  – job_template *drmaa_allocate_job_template (void)
  – void drmaa_delete_job_template (job_template *jt)
- Setter/getter job template routines
  – int drmaa_set_attribute(job_template *jt, char *name, char *value);
  – int drmaa_set_vector_attribute(job_template *jt, char *name, char **values);
  – char* drmaa_get_attribute(job_template *jt, char *name);
  – char** drmaa_get_vector_attribute(job_template *jt, char *name);

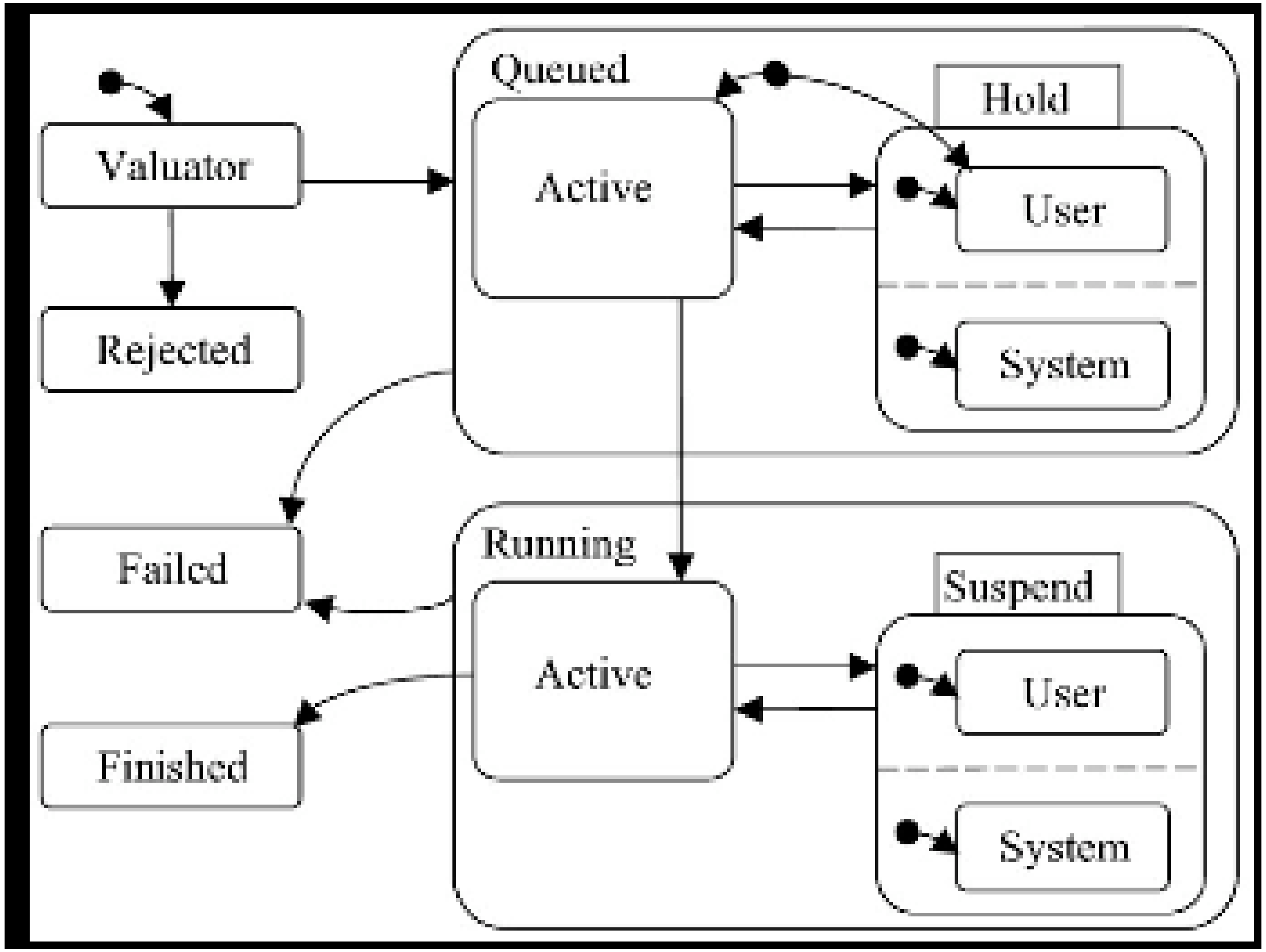
Job Submission

- Jobs submitted to the DRM system are identified via an opaque job identifier (char*)
- Single job identifiers are returned by
  - int drmaa_run_job(job_template *jt, char *job_id)
- Bulk job submissions return multiple job identifiers
  - int drmaa_run_bulk_job(char **job_ids, job_template *jt, int start, int end, int incr)

Job Monitoring, Control, and Status

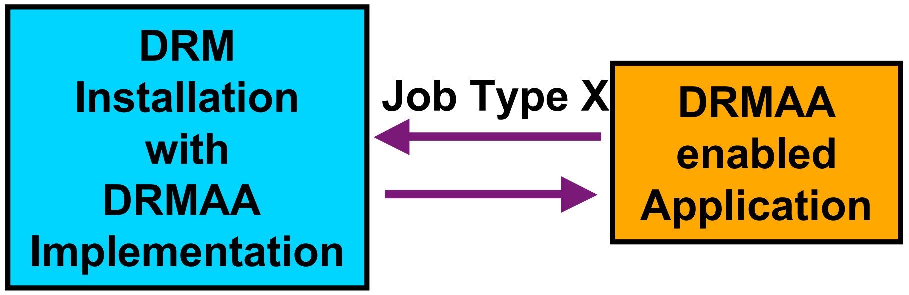
- Monitoring/Control functions
  - int drmaa_control(char *job_id, int action);
  - int drmaa_synchronize(char **job_ids);
  - int drmaa_job_ps(char *job_id, int *remote_ps);
- Blocking and non-blocking waiting for one or more jobs to finish (like POSIX wait4(2))
  - char *drmaa_wait(char *jobid, int *status, int timeout, char **rusage);
  - drmaa_wif[exited|signaled|aborted] and friends to get more information about failed jobs

Job State Transition



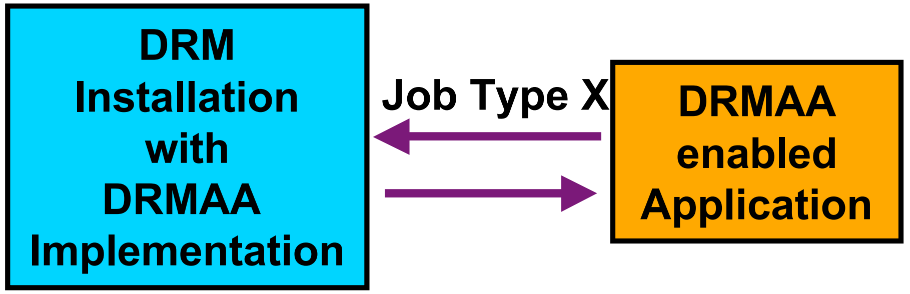
Job Categories

Site A


DRM Installation with DRMAA Implementation
Job Type X
DRMAA enabled Application

- Cluster consists of machines where X jobs run and others where they don't run

Site B


DRM Installation with DRMAA Implementation
Job Type X
DRMAA enabled Application

- X jobs run at all machines in cluster

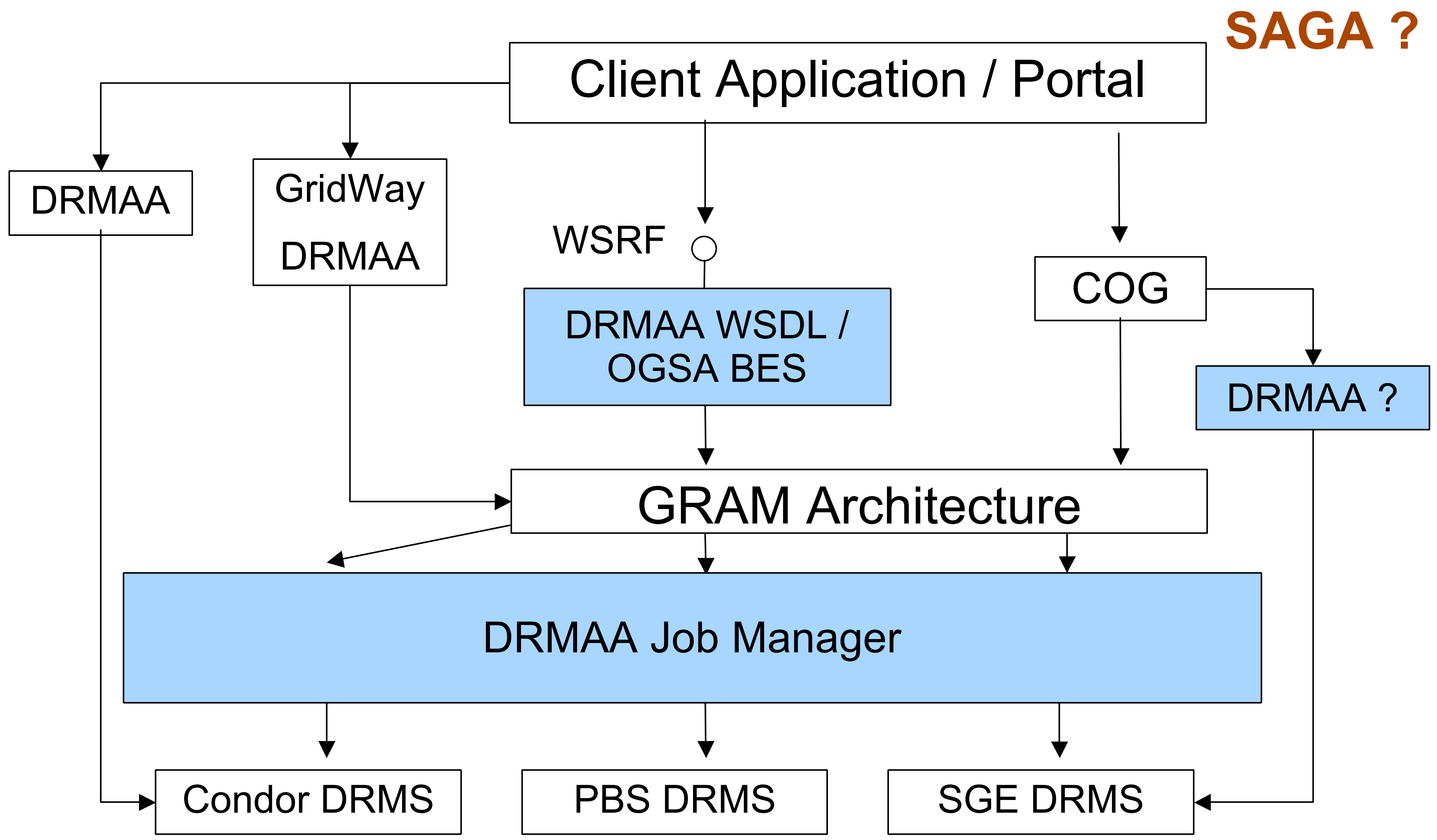
Native DRMS Options

- The end user interacts with the DRMS via native_specification parameter
  - Simple solution
  - DRMAA implementation ignores the DRMAA DRMS implicitly used and disallowed options
  - Dist. Appls. Developers and DRMS vendors are not involved in the local environment spec.
  - The burden is on the end users to define the execution environment
    - Need to know DRM
    - Need to know the remote application installation

DRMAA Placement

- On top of DRM systems
- On top of Globus
- Beneath GRAM
- UNICORE TSI interface to DRMSs
- CoG adapter
- On top of CoG
- Interfaced by a Portal, application, shell
- Portable command line utilities (qsub, qstat)

A World of Submission API's



DRMAA in Practice

- Multiple implementations since 2004
  - Product implementation in Sun Grid Engine 6
    - C- and Java-binding implementation
  - Prototype in Condor 6.7 series
    - C-binding implementation
  - CPAN Perl DRMAA module (Tim Harsch)
    - On-top-of DRMAA C-library
  - GridWay DRMAA implementation
    - Allows DRMAA on-top-of Globus
  - Prototype for Globus 3 DRMAA job manager (HPI)
    - Based on DRMAA Perl implementation
- Tutorials, programming examples, test suites
  - http://gridengine.sunsource.net
  - http://www.dcl.hpi.uni-potsdam.de/research/drmaa
  - GGF12 tutorial, JavaOne 05 tutorial materials

Need for Improvement !?!

- Feedback from practical usage of available implementation(s) fed‘s back into spec
  - Just look at the GridForge tracker and the SGE / Condor mailing lists
- Some details under-specified
  - Behavior in multi-threaded environments
- C-centric style makes it hard to develop conformant object-oriented bindings
  - Massive feedback from Java and .NET language binding specification work
- Several missing error codes
- But: Keep the API as small as it is !!!

DRMAA IDL Spec

- Started work in early 2005
- Based on Java- and .NET-binding experiences
- Specification through standardized OMG Interface Definition Language (IDL)
  - No, that does not mean CORBA ;-)
  - Example: W3C DOM specification
  - DRMAA language bindings will (and should) **not** rely on IDL language bindings from OMG
    - Complicated, weired semantics
    - Simple custom binding by specifying consistent mapping rules
    - Examples for Java in the IDL-spec
  - Usage of IDL avoids wording issues (i.e. ‚attribute‘ vs. ‚property‘)
  - Allows for true language-independent description of namespaces, enumerations, constants, and time values

DRMAA IDL Spec (contd.)

- Improved, more consistent description text for all functions
  - More details regarding advanced OO-specific features (multiple session objects, exception hierarchies)
  - Consider languages with introspection functionalities
  - Some details about RPC-DRMAA scenarios (SOAP, RMI, ...)
- More parameter placeholders (e.g. for job ID)
- A lot more possible error codes for the operations

Backward Compatibility

- C- and Java bindings in their current state can be mostly derived also from the IDL spec
  - Demand for consistent name mapping might change one or two method names in the C-binding
  - Introduction of new job state / error codes does not break existing applications
  - DRMAA has already a notion of versioning
- .NET binding will be re-designed based on the IDL spec
- No official binding documents for Perl and Python so far

DRMAA Documents

- DRMAA GFD-P-R or GFD.22 document
  - Since June 2004
- C binding v1.0
  - Ready for submission to GFCS
- C binding experimental document v0.98
- Java binding 0.6.1
  - Fairly complete
- .NET binding v0.2
  - Needs a synch with IDL-spec
- IDL document
  - v0.3 and nearly feature complete
  - Need to augment with the DRMAA GFD-P-R text
  - Will be submitted as a standalone GFD-P-R doc

Next Steps

- Putting the DRMAA-IDL spec in the GGF document chain (GGF14 version is nearly final)
- Prerequisite for some announced activities
  - .NET-binding implementation (HPI)
    - On-top-of DRMAA C-library
  - Improved Condor C library
    - Join the efforts at http://sf.net/projects/condor-ext
  - Python binding specification
  - Maybe more implementations
- DRMAA collaborates with SAGA, JSDL, and OGSA-BES for identifying synergy effects

Conclusion

- Please take part in the discussion
  - Bi-weekly con calls
    - Toll Free: (866)545-5198 Code: 6898552
    - Regular: (865)521-8904
  - GridForge tracker
  - E-mail: *drmaa-wg@gridforum.org*
  - Archive: Use the link at *http://drmaa.org*
- Please implement DRMAA and tell us your experience
  - It‘s easy, Dan did it 4 times ;-) ...

Thank you !