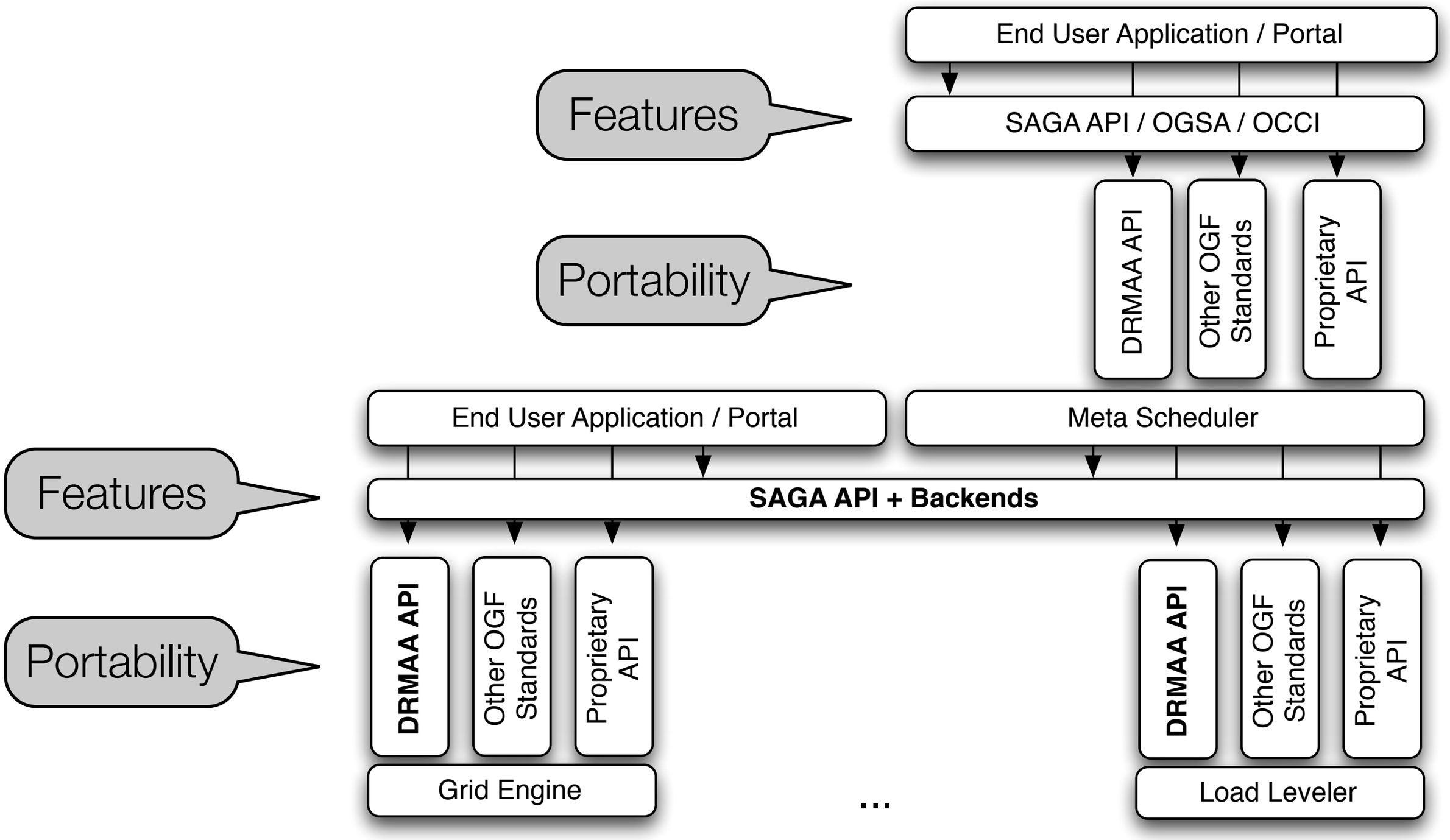
Distributed Resource Management Application API (DRMAA) Version 2

Dr. Peter Tröger
Hasso-Plattner-Institute, University of Potsdam
peter@troeger.eu

DRMAA-WG Co-Chair

http://www.drmaa.org/

Open Grid Forum (OGF) Application Area



DRMAA

DRMAA
Distributed Resource Management Application API — www.drmaa.org

- DRMAA group established in 2002
- Goal: Standardized API for distributed resource management systems (DRMS)
  - Low-level **portability** for cluster and grid infrastructure applications
  - Simple design, **bottom-up philosophy**
- Different DRMAA 1.0 documents
  - June 2004 - DRMAA 1.0 Proposed Recommendation (GFD.22)
  - April 2008 - Shift to IDL based root specification, some clarifications (GFD.130)
  - June 2008 - DRMAA 1.0 Grid Recommendation (GFD.133)
  - Official language binding documents for C, Java, Python (GFD.143)
  - Experience reports, tutorials, unofficial language bindings for Perl, Ruby and C#

Stakeholders

- Distributed resource management (DRM) system
  - Distributes computational tasks on execution resources
  - Central scheduling entity
- DRMAA implementation / library : Implementation of a language binding, semantics as in the root spec

- Submission host: Resource that runs the DRMAA-based application
- Execution host: Resource that can run a submitted computational task
- Job: One or more operating system processes on a execution host

DRMAA	Reference	GLUE 2.0	Reference [1]
DRM system	Section 1.1	Manager	Section 5.9
Execution host	Section 1.1	ExecutionEnvironment + ComputingManager	Section 6.4 / 6.6
Socket	Section 5.3.3	Physical CPU	Section 6
Core	Section 5.3.4	Logical CPU	Section 6
Job	Section 1.1	ComputingActivity	Section 6.9
Job category	Section 1.4	ApplicationEnvironment	Section 6.7
`UNSET` value	Section 1.3	Placeholder values for unknown data	Appendix A

DRMAA v1 Success Story

- Product-quality implementations of DRMAA C and Java, broad industrial up-take
  - *Grid Engine* since 2006
  - *Condor* since 2006
  - *PBS/Torque*, *LSF*, *LoadLeveler*, *Kerrighed*, *SLURM*, *Globus* (via *GridWay*), **...**
    (third-party implementations)
- Large and silent user community
  - Meta schedulers accessing DRM resources
    (Galaxy, QosCosGrid, MOAB, eXludus, OpenDSP, EGEE, Unicore, SAGA, **...**)
  - Applications or portals accessing DRM resources
    (BLAST, workflow processing, finance sector, seismic data, Mathematica, **...**)
  - Hundreds (?) of unknown Grid Engine users

DRMAA v1 C Example

```c
#include "drmaa.h"

int main(int argc, char **argv) {
     char error[DRMAA_ERROR_STRING_BUFFER];
     int errnum = 0;
     drmaa_job_template_t *jt = NULL;

     errnum = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER);
     if (errnum != DRMAA_ERRNO_SUCCESS) return 1;
     errnum = drmaa_allocate_job_template(&jt, error, DRMAA_ERROR_STRING_BUFFER);
     if (errnum != DRMAA_ERRNO_SUCCESS) return 1;
     drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, "sleeper.sh",
                         error, DRMAA_ERROR_STRING_BUFFER);
     const char *args[2] = {"5", NULL};
     drmaa_set_vector_attribute(jt, DRMAA_V_ARGV, args, error,
                            DRMAA_ERROR_STRING_BUFFER);
     char jobid[DRMAA_JOBNAME_BUFFER];
     errnum = drmaa_run_job(jobid, DRMAA_JOBNAME_BUFFER, jt, error,
                    DRMAA_ERROR_STRING_BUFFER);
     if (errnum != DRMAA_ERRNO_SUCCESS) return 1;
     errnum = drmaa_delete_job_template(jt, error, DRMAA_ERROR_STRING_BUFFER);
     errnum = drmaa_exit(error, DRMAA_ERROR_STRING_BUFFER);
     return 0;
}
```

DRMAA v1 Java Example (1/2)

```
try {
     session.init ("");

     System.out.println ("Version: " + session.getDrmaaImplementation ());

     JobTemplate jt = session.createJobTemplate ();
     jt.setRemoteCommand ("<SGE_ROOT>/examples/javaone/sleeper.sh");
     jt.setArgs (new String[]{"30"});
     jt.setWorkingDirectory ("<SGE_ROOT>/<SGE_CELL>/javaone");
     jt.setJobCategory ("sleeper");

     String jobId = session.runJob (jt);
     List jobIds = session.runBulkJobs (jt, 1, 4, 1);

     System.out.println ("Job " + jobId + " is running");

     for (Object id: List jobIds) {
        System.out.println ("Job " + id + " is running");
     }

     session.deleteJobTemplate (jt);
```

DRMAA v1 Java Example (2/2)

```
      JobInfo info = session.wait (jobId, Session.TIMEOUT_WAIT_FOREVER);
      Map usage = info.getResourceUsage ();

      for (Object name : usage.keySet ()) {
         System.out.println (name + "=" + usage.get (name));}

      if (info.hasExited ()) {
         System.out.println ("Job exited: " + info.getExitStatus ());}
      else if (info.hasSignaled ()) {
         System.out.println ("Job signaled: " + info.getTerminatingSignal ());

         if (info.hasCoreDump ()) {
            System.out.println ("A core dump is available.");}}
      else if (info.wasAborted ()) {
         System.out.println ("Job never ran.");}
      else {
         System.out.println ("Exit status is unknown.");}

      session.synchronize (jobIds, Session.TIMEOUT_WAIT_FOREVER, true);
   }
   catch (DrmaaException e) {
      e.printStackTrace ();
      System.exit (1);}
}
```

DRMAA v1 Got Old

- DRM systems got better
  - Concept of resources, session persistency, advance reservation, parallel jobs, many-core systems, queues, state models, WS-* stuff, ...
- Some obsolete / never implemented features
  - Date / time handling, host-to-host file staging, specialized job states, ...
- Awkward design decisions
  - Job synchronization, job monitoring, data reaping, native specification, ...
- DRMAAv2 work started in 2009
  - Public survey, Sun customer feedback, implementation experiences, ...
  - Close collaboration with SAGA, GAT, and OCCI WG; considered JSDL & BES
  - Performant investigation of current DRM systems

DRMAA v2 - Status Today

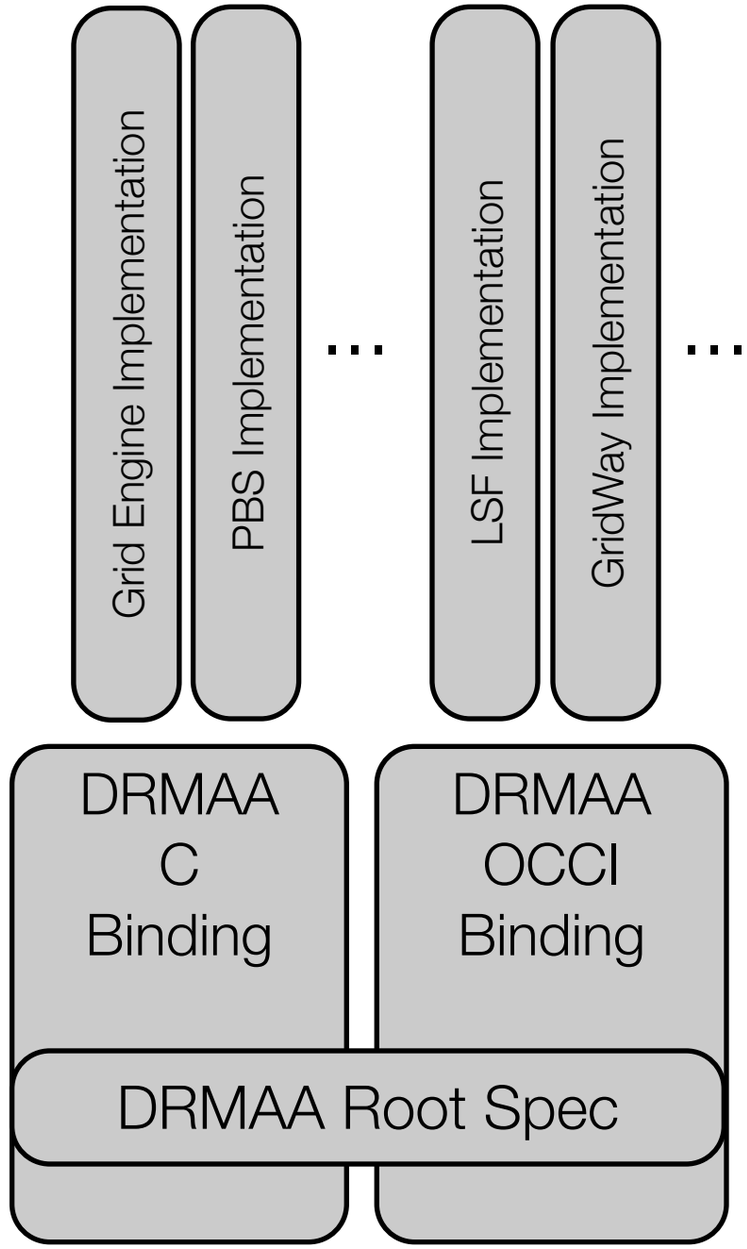
- DRMAAv2 root specification is an official OGF standard since January 2012
  - http://www.ogf.org/documents/GFD.194.pdf
- Final draft of DRMAAv2 C binding announced yesterday
  - No more major changes, header file available
  - OGF public comment period will start in May 2012
  - Official specification in late summer 2012
  - New project for fork-based reference implementation (drmaav2-mock)
- Language bindings for C++, Python, Java and OCCI under way

DRMAA v2 Implementations ?

- Implementations announced
  - Fork-based reference implementation (DRMAA Working Group)
  - Univa Grid Engine (Daniel Gruber)
  - Torque / PBS Pro (Mariusz Mamonski)
  - GridWay (Eduardo Huedo)
  - Condor (Todd Tannenbaum)
- Collection type handling in C should be re-usable
- Interfacing the DRM system is already part of the DRMAAv1 library
  - Old libraires can be extended with new features
    (job arrays, session persistency, reservation, monitoring)

DRMAA v2 Design Approach

- All behavioral aspects in the IDL-based root specification
  - Same model as W3C DOM, OGF SAGA, ...
  - What functions are offered ? How are they grouped ?
  - What are possible error conditions ?
  - For language binding designers and implementors
- Language binding just defines syntactical mapping
  - „DRMAA IDL construct A maps to programming language construct B“
  - Root spec mandates some design decisions in the language binding (e.g. „UNSET“)
- End users should get dedicated man pages



DRMAA v2 Layout

interface **SessionManager**

interface **DrmaaCallback**

struct **DrmaaNotification**

Establish session with the DRM system

interface **JobSession**

interface **Job**

interface **JobArray**

struct **JobTemplate**

struct **JobInfo**

interface **ReservationSession**

interface **Reservation**

struct **ReservationTemplate**

struct **ReservationInfo**

interface **MonitoringSession**

struct **MachineInfo**

struct **QueueInfo**

struct **SlotInfo**

Work with jobs / advance reservation / monitoring features of the DRM system

Exceptions

Enumerations

interface **DrmaaReflective**

Session Manager

- Create multiple connections to one (or more) DRM system(s) at a time
- `JobSession / ReservationSession`
  - Persistent storage of session state (at least) on submission machine
  - Can be reloaded by name, explicit reaping
- `MonitoringSession`
  - Read-only semantic, global view on all machines
- Maps nicely to SAGA and friends - management vs. monitoring
- Security remain out of scope for DRMAA
- Explicitly supports the portal / command-line tools use case
- Old concept of contact strings from DRMAAv1

Session Manager

```
interface SessionManager{
     readonly attribute string drmsName;
     readonly attribute Version drmsVersion;
     readonly attribute string drmaaName;
     readonly attribute Version drmaaVersion;
     boolean supports(in DrmaaCapability capability);
     JobSession createJobSession(in string sessionName, in string contact);
     ReservationSession createReservationSession(
                         in string sessionName, in string contact);
     JobSession openJobSession(in string sessionName);
     ReservationSession openReservationSession(in string sessionName);
     MonitoringSession openMonitoringSession (in string contact);
     void closeJobSession(in JobSession s);
     void closeReservationSession(in ReservationSession s);
     void closeMonitoringSession(in MonitoringSession s);
     void destroyJobSession(in string sessionName);
     void destroyReservationSession(in string sessionName);
     StringList getJobSessionNames();
     StringList getReservationSessionNames();
     void registerEventNotification(in DrmaaCallback callback);
};
```

Event Callback

- Optional support for event push notification
- Application implements `DrmaaCallback` interface
- Demands some additional rules from the language binding
- Push notification at least supported in Grid Engine
- Heavily demanded by SAGA and end users
- Might be realized by a polling DRMAA library
- Nested callbacks are forbidden
- Scalability is out of scope

```
interface DrmaaCallback {
     void notify(in DrmaaNotification notification);
};
struct DrmaaNotification {
     DrmaaEvent event;
     string jobId;
     string sessionName;
     JobState jobState;
};
enum DrmaaEvent {
     NEW_STATE, MIGRATED, ATTRIBUTE_CHANGE
};
```

Job Categories

- Consider deployment properties
  - Path settings, environment variables, software installed, application starters, ...
- Job category references a site-specific configuration
- Non-normative set of recommendations on DRMAA home page
- Idea is to perform updates without spec modification
- Site operators should be allowed to create their own job categories (configuration of the implementation)
- Initial set derived from JSDL SPMD Extension (GFD.115) + experience
- Proposals welcome ...

DRMAA Job Category Name	Description
"MPI"	Any MPI environment
"GridMPI"	A GridMPI environment
"LAM-MPI"	A LAM/MPI environment
"MPICH1"	A MPICH version 1 environment
"MPICH2"	A MPICH version 2 environment
"OpenMPI"	A OpenMPI environment
"PVM"	A PVM environment
"OpenMP"	A OpenMP environment
"OpenCL"	A OpenCL environment
"Java"	A Java environment

Job Session

- New: Fetch list of supported job categories (portal case)
- New: `Job` filtering
- New: `JobArray`
- New: `maxParallel` restriction for bulk jobs
- Old `synchronize()` was replaced by class-based state waiting approach
  - More responsive
  - Get rid of partial job failure problems

```
interface JobSession {
      readonly attribute string contact;
      readonly attribute string sessionName;
      readonly attribute StringList jobCategories;
      JobList getJobs(in JobInfo filter);
      JobArray getJobArray(in string jobArrayId);
      Job runJob(in JobTemplate jobTemplate);
      JobArray runBulkJobs( in JobTemplate jobTemplate,
                            in long beginIndex,
                            in long endIndex,
                            in long step,
                            in long maxParallel);
      Job waitAnyStarted(in JobList jobs,
                         in TimeAmount timeout);
      Job waitAnyTerminated(in JobList jobs,
                            in TimeAmount timeout);
};
native ZERO_TIME;
native INFINITE_TIME;
```

Job Template

- Basic rules
  - DRM systems manage machines, which have a CPU and physical memory
  - Machines can be booked in advance reservation
  - DRM systems manage **queues** and **slots** as resources, **opaque** to DRMAA
- Jobs can be submitted ...
  - ... to specified candidate machines or a queue
  - ... to machines matching an OS type, architecture type, or memory requirement
  - ... as ‚classified‘ job with special treatment by the DRMS (jobCategory)
    - Examples: MPICH2 job, OpenMPI job, OpenMP job

Job Template

- Most things remain as in DRMAAv1, but:
  - Relative start / end times are gone, switched to RFC822
  - Only copying between submission and exception machine, no host names
  - Standardized job category names (based on GFD.115, see web page)
  - Standardized resource limit types (*setrlimit*)
  - Several new resource specification attributes

```
struct JobTemplate {
      string remoteCommand;
      OrderedStringList args;
      boolean submitAsHold;
      boolean rerunnable;
      Dictionary jobEnvironment;
      string workingDirectory;
      string jobCategory;
      StringList email;
      boolean emailOnStarted;
      boolean emailOnTerminated;
      string jobName;
      string inputPath; string outputPath;
      string errorPath; boolean joinFiles;
      string reservationId;
      string queueName;
      long minSlots; long maxSlots;
      long priority;
      OrderedStringList candidateMachines;
      long minPhysMemory;
      OperatingSystem machineOS;
      CpuArchitecture machineArch;
      AbsoluteTime startTime;
      AbsoluteTime deadlineTime;
      Dictionary stageInFiles;
      Dictionary stageOutFiles;
      Dictionary resourceLimits;
      string accountingId;
};
```

Optional vs. Implementation-Specific

- Make non-mandatory things explicit
- `DrmaaCapability` enumeration
  - One enum per optional feature + `SessionManager::supports` method
- `DrmaaReflective` interface
  - Lists of implementation-specific attributes + generic getter / setter functions

```
enum DrmaaCapability {
     ADVANCE_RESERVATION, RESERVE_SLOTS, CALLBACK, BULK_JOBS_MAXPARALLEL, JT_EMAIL,
     JT_STAGING, JT_DEADLINE, JT_MAXSLOTS, JT_ACCOUNTINGID, RT_STARTNOW, RT_DURATION,
     RT_MACHINEOS, RT_MACHINEARCH };
interface DrmaaReflective {
     readonly attribute StringList jobTemplateImplSpec;
     readonly attribute StringList jobInfoImplSpec;
     ...
     string getInstanceValue(in any instance, in string name);
     void setInstanceValue(in any instance, in string name, in string value);
     string describeAttribute(in any instance, in string name); }
```

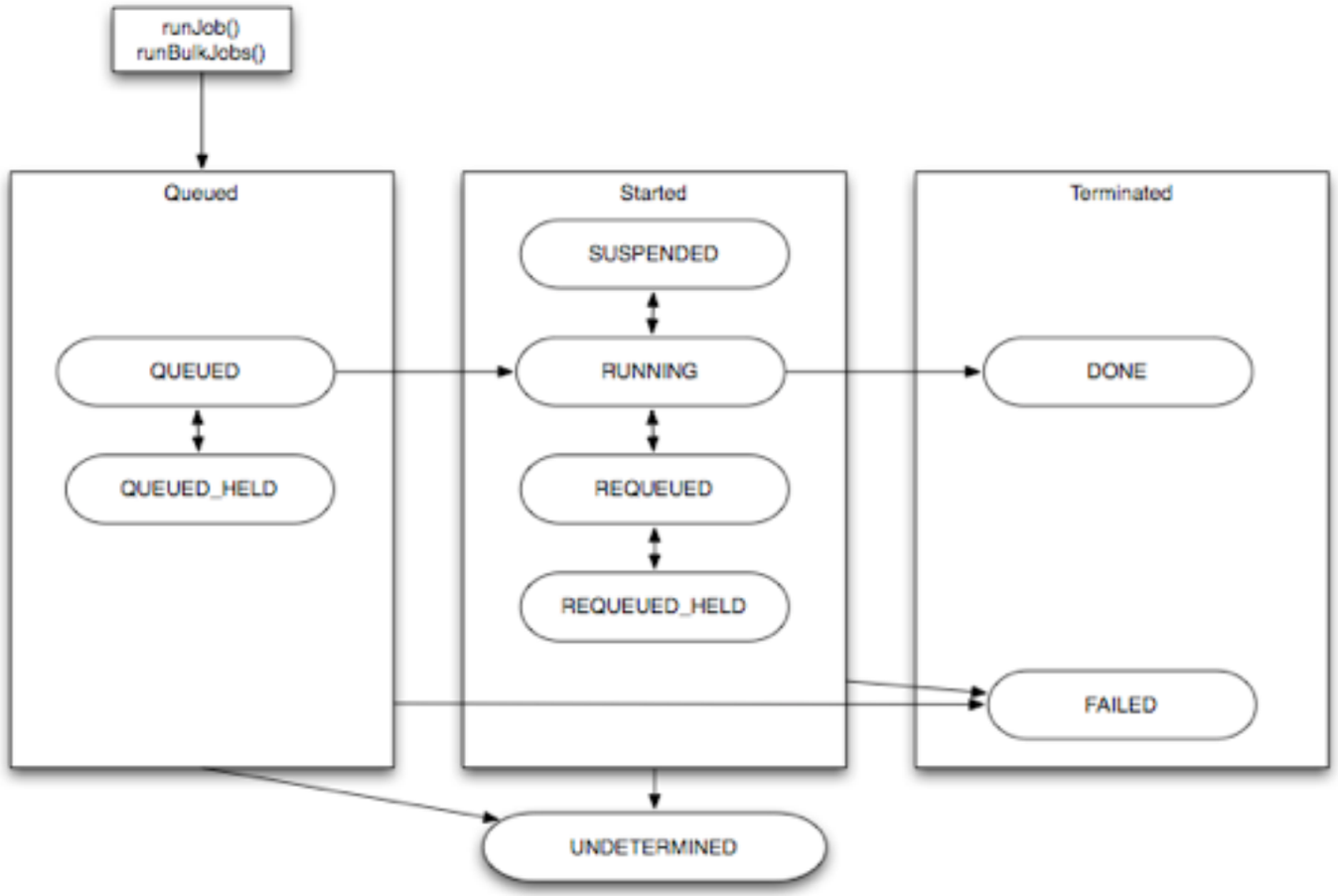
Job / JobArray

- Heavy cleanup
  - Dedicated control methods
  - Explicit job information fetching
- `waitStarted()` and `waitTerminated()` as on `JobSession` level
- Support for the new job sub-state model
- `JobArray` offers control operations on bulk jobs
- Rejected: Signaling, modifying running jobs, ...

```
interface Job {
      readonly attribute string jobId;
      readonly attribute string sessionName;
      readonly attribute JobTemplate jobTemplate;
      void suspend();
      void resume();
      void hold();
      void release();
      void terminate();
      JobState getState(out any jobSubState);
      JobInfo getInfo();
      Job waitStarted(in TimeAmount timeout);
      Job waitTerminated(in TimeAmount timeout);
}
interface JobArray {
   readonly attribute string jobArrayId;
   readonly attribute JobList jobs;
   readonly attribute string sessionName;
   readonly attribute JobTemplate jobTemplate;
   void suspend();
   void resume();
   void hold();
   void release();
   void terminate();
}
```

Everybody Loves State Models

- Number of states reduced in comparison to DRMAAv1
- New sub-state concept
  - Similar to OGSA-BES
  - Allows simple DRMAAv1 mapping
- Wait functions work only on class level (timing issues)
- No more different hold modes



DRMAA JobState	SAGA JobState [4]	OGSA-BES Job State [3]
UNDETERMINED	N/A	N/A
QUEUED	Running	Pending (Queued)
QUEUED_HELD	Running	Pending (Queued)
RUNNING	Running	Running (Executing)
SUSPENDED	Suspended	Running (Suspended)
REQUEUED	Running	Running (Queued)
REQUEUED_HELD	Running	Running (Queued)
DONE	Done	Finished
FAILED	Cancelled, Failed	Cancelled, Failed

JobInfo

- No surprises with `JobInfo`
  - Used for both filtering and job status representation
  - Snapshot semantics
  - No promises for terminated jobs
- Implementations can give a meaning to the reported list of machines being used

```
struct JobInfo {
   string jobId;
   long exitStatus;
   string terminatingSignal;
   string annotation;
   JobState jobState;
   any jobSubState;
   OrderedSlotInfoList allocatedMachines;
   string submissionMachine;
   string jobOwner;
   long slots;
   string queueName;
   TimeAmount wallclockTime;
   long cpuTime;
   AbsoluteTime submissionTime;
   AbsoluteTime dispatchTime;
   AbsoluteTime finishTime;
};
```

Advance Reservation

- Completely new
- Fits nicely to the rest of the spec

```
interface ReservationSession {
   readonly attribute string contact;
   readonly attribute string sessionName;
   Reservation getReservation(in string reservationId);
   Reservation requestReservation(in ReservationTemplate reservationTemplate);
   ReservationList  getReservations ();
};
interface Reservation {
   readonly attribute string reservationId;
   readonly attribute string sessionName;
   readonly attribute ReservationTemplate reservationTemplate;
   ReservationInfo getInfo();
   void terminate();
};
```

Advance Reservation

- Reservation template
  - Different combinations of `startTime`, `endTime`, and `duration` are allowed
  - Sliding window + time frame support
  - Support for authorization setup

```
struct SlotInfo {
   string machineName;
   long slots;
};
...
typedef sequence<SlotInfo> OrderedSlotInfoList;
...
```

```
struct ReservationTemplate {
   string reservationName;
   AbsoluteTime startTime;
   AbsoluteTime endTime;
   TimeAmount duration;
   long minSlots;
   long maxSlots;
   string jobCategory;
   StringList usersACL;
   OrderedStringList candidateMachines;
   long minPhysMemory;
   OperatingSystem machineOS;
   CpuArchitecture machineArch;
};
struct ReservationInfo {
   string reservationId;
   string reservationName;
   AbsoluteTime reservedStartTime;
   AbsoluteTime reservedEndTime;
   StringList usersACL;
   long reservedSlots;
   OrderedSlotInfoList reservedMachines;
};
```

DRMS Resource Monitoring

- Completely new
  - Global view on resources
  - Stateless session instance
  - Implementation has freedom to leave out information
- Resource information model matching to SAGA
- Virtual memory = physical memory + swap space
- Load = 1-min average load
- Snapshot semantics

```
typedef sequence <Reservation> ReservationList;
typedef sequence <Job> JobList;
typedef sequence <QueueInfo> QueueInfoList;
typedef sequence <MachineInfo> MachineInfoList;
...
interface MonitoringSession {
   ReservationList getAllReservations();
   JobList getAllJobs(in JobInfo filter);
   QueueInfoList getAllQueues(in StringList names);
   MachineInfoList getAllMachines(in StringList names);
};
struct QueueInfo {
   string name;
};
struct MachineInfo {
   string name;
   boolean available;
   long sockets;
   long coresPerSocket;
   long threadsPerCore;
   double load;
   long physMemory;
   long virtMemory;
   OperatingSystem machineOS;
   Version machineOSVersion;
   CpuArchitecture machineArch; };
```

More Stuff

- `DRMAA_INDEX_VAR`
  - Implementations should set the environment variable `DRMAA_INDEX_VAR`
  - Contains the name of the DRMS environment variable providing the job index
    - `TASK_ID` (Grid Engine), `PBS_ARRAYID` (Torque), `LSB_JOBINDEX` (LSF)
  - Jobs are enabled to get their own parametric index

DRMAAv2
C Binding Example

OCCI-DRMAA Example

Step 1 - Create a `jobtemplate` *resource:*

```
> POST /drmaa/jobtemplate/ HTTP/1.1
> [...]
> X-OCCI-Attribute: occi.drmaa.remoteCommand="/bin/date"
> X-OCCI-Attribute: occi.drmaa.machineOS="LINUX"
> X-OCCI-Attribute: occi.drmaa.email="peter@troeger.eu"
> X-OCCI-Attribute: occi.drmaa.email="tmetsch@platform.com"
> X-OCCI-Attribute: occi.drmaa.emailOnTerminated=true
> [...]

< HTTP/1.1 201 CREATED
< [...]
< Location: http://example.com/drmaa/jobtemplate/template1
< [...]
```

Step 2, Option 2 - Create a `job` *resource:*

```
> POST /drmaa/job/ HTTP/1.1
> [...]
> X-OCCI-Attribute: occi.drmaa.session="/drmaa/jobsession/session1"
> X-OCCI-Attribute: occi.drmaa.jobTemplate="/drmaa/jobtemplate/template1"
> [...]

< HTTP/1.1 201 CREATED
< [...]
< Location: http://example.com/drmaa/job/job43
< [...]
```

Note: Job Requirements are Hard to Unify

A	B	C	D	E	F	G	H
	JSDL		**SGE**				
	JSDL Name		**SGE Queue Properties**	**SGE Description**		**Condor Machine ClassAd**	**Condor submission file**
Resource requirement for job							
	CandidateHosts					MACHINE	
	TotalResourceCount		slots	Number of processes (allowed) to run			machine_count
	FileSystem					-- (no free choice of FS)	
	ExclusiveExecution		(This is accomplished through a special complex as of 6.2u3.)				
	OperatingSystem		(arch built-in complex)			OPSYS	
	CPUArchitecture		(arch built-in complex)			ARCH	
	IndividualCPUSpeed					KFLOPS	
	IndividualCPUTime						
	IndividualCPUCount		(num_proc built-in complex)			CPUS	
	IndividualNetworkBandwidth						
	IndividualPhysicalMemory		(mem_total built-in complex)			MEMORY	
	IndividualVirtualMemory		(virtual_total built-in complex)			VirtualMemory	
	IndividualDiskSpace					DISK	
	TotalCPUTime		s_cpu / h_cpu	Soft / hard limit for CPU time of all processes			
	TotalCPUCount						
	TotalPhysicalMemory						
	TotalVirtualMemory		s_vmem / h_vmem	Soft / hard limit for job virtual memory			
	TotalDiskSpace		s_fsize / h_fsize	Soft / hard limit for bytes on disk			
				Minimum time between			

Note: Job Submission Is Hard to Unify

	LSF	Torque	PBS Pro
Wildcard Support	no	not recommended	yes
Other than submission host	no	yes	yes
Appending file	yes	no	no
Directory Staging	no	not by default	yes

- Ensuring true application portability with a unified API is difficult
  - Interoperability (OGSA, JSDL) does not provide portability
  - Profiles (!) with „SHOULD“ and „UnsupportedFeatureFault“ are not helpful
- DRMAA tries to define **mandatory** job template attributes and API functions that are **implementable** in **most** DRM systems
  - This is why DRMAA will never by exhaustive -> use SAGA !

Note: Relation to Other OGF Standards

- JSDL is too exhaustive, GLUE is too conservative, SAGA / OGSA-BES use DRMAA

DRMAA `OperatingSystem`	JSDL `jsdl:OperatingSystemTypeEnumeration`	GLUE v2.0
HPUX	HPUX	
LINUX	LINUX	OSFamily_t:linux
IRIX	IRIX	
TRUE64	Tru64_UNIX, OSF	
MACOS	MACOS	OSFamily_t:macosx
SUNOS	SunOS, SOLARIS	OSFamily_t:solaris
WIN	WIN95, WIN98, Windows_R_Me	OSFamily_t:windows
WINNT	WINNT, Windows_2000, Windows_XP	OSFamily_t:windows
AIX	AIX	OSName_t:aix
UNIXWARE	SCO_UnixWare, SCO_OpenServer	
BSD	BSDUNIX, FreeBSD, NetBSD, OpenBSD	

DRMAA `CpuArchitecture`	JSDL `jsdl:ProcessorArchitectureEnumeration`	GLUE v2.0
ALPHA	other	
ARM	arm	
CELL	other	
PARISC	parisc	
X86	x86_32	Platform_t:i386
X64	x86_64	Platform_t:amd64
IA64	ia64	Platform_t:itanium
MIPS	mips	
PPC	powerpc	Platform_t:powerpc
PPC64	powerpc	Platform_t:powerpc
SPARC	sparc	Platform_t:sparc
SPARC64	sparc	Platform_t:sparc

Conclusion

- Just the right time for DRMAAv2 adoption in DRM systems
- Window of opportunity for tweaking the specs is still open
- Already broad acceptance on user side (SAGA / DRMAAv1 community)
- Promising new direction with OCCI-DRMAA mapping
- Standardization is collaborative work
  - Roger Brobst (Cadence Design Systems), Daniel Gruber (Univa GmbH), Mariusz Mamonski (Poznan Supercomputing and Networking Center), Morris Riedel (FZ Jülich), Daniel Templeton (Cloudera Inc.), Andre Merzky (LSU), Thijs Metsch (Platform Computing / IBM), OGF working groups, ...
- And if you want to visit Potsdam ...