GWD-R
Distributed Resource Management
Application API (DRMAA) Working Group

Daniel Templeton, Sun Microsystems (maintainer)
Peter Tröger, Universität Potsdam

January, 2005

Distributed Resource Management Application API Object-Oriented Bindings 0.1

Status of This Memo

This memo is a Global Grid Forum Grid Working Draft - `Recommendations` (GWD-R) in process, in general accordance with the provisions of Global Grid Forum Document GFD-C.1, the Global Grid Forum Documents and Recommendations: Process and Requirements, revised April 2002.

Table of Contents

Index of Tables

1 Abstract

This document describes the Distributed Resource Management Application API (DRMAA) object-oriented bindings. The document is based on the implementations work of the DRMAA GWD-R document.

2 Overview

This document describes the object-oriented binding for the DRMAA interface. It arises from the results of a collaborative effort to bring the Java™ language binding and .Net binding into agreement.

3 Design Decisions

An effort has been made to choose design patterns that are not unique to a specific language. However, in some cases, various language disagree over some points. In those cases, the most meritous approach was taken, irrespective of language.

3.1 Service Provider Interface

The object-oriented binding borrows from Java the notion of a service provider interface. This idea means that the core API need only be implemented once, whereas each vendor can provide his own service provider implementation. The service provider interface is accomplished by factoring common functionality out into concrete and abstract classes. The service provider then extends the abstract classes and uses the concrete classes to build his own implementation. The advantage is that the common core functionality need only be implemented once in any given language, and that developers can become familiar with a single package which may hide beneath it several different vendor implementations.
In object-oriented languages which do not support the notion of abstract classes or class inheritance, the service provider interface can be ignored, and all abstract classes can be considered concrete.

4 The Object-Oriented Binding API

The DRMAA Interface Specification was written originally with a proceedural slant. As such, several aspects of the DRMAA interface needed to be altered slightly to better fit with an object-oriented language like the Java language or .Net. Among the aspects that changed are variable and method naming and the error structure.

The object-oriented binding uses the concept of properties instead of attributes for languages which support introspection, such as the Java language and .Net. A property is defined as a named entity which has a corresponding setter and getter. For example, a property named “prop” of type “Type” would have a getter with the signature “Type getProp()” and a setter with the signature “void setProp(Type t)”. Below is a table of the DRMAA attributes, their corresponding object-oriented property names, and their suggested types. Since type is anything but universal, the types listed in the table are only a suggestion, and the most appropriate type must be chosen individually for each language binding.

DRMAA Attribute	*OO Property*	*Type*
drmaa_remote_command	remoteCommand	String
drmaa_v_argv	args	String[]
drmaa_js_state	jobSubmissionState	int
drmaa_v_env	jobEnvironment	Dictionary
drmaa_wd	workingDirectory	String
drmaa_job_category	jobCategory	String
drmaa_native_specification	nativeSpecification	String
drmaa_v_email	email	String[]
drmaa_block_email	BlockEmail	boolean
drmaa_start_time	startTime	org.ggf.drmaa.PartialTimestamp
drmaa_job_name	jobName	String
drmaa_input_path	inputPath	String
drmaa_output_path	outputPath	String
drmaa_error_path	errorPath	String
drmaa_join_files	joinFiles	boolean
drmaa_transfer_files	transferFiles	org.ggf.drmaa.FileTransferMode
drmaa_deadline_time	deadlineTime	org.ggf.drmaa.PartialTimestamp
drmaa_wct_hlimit	hardWallclockTimeLimit	long
drmaa_wct_slimit	softWallclockTimeLimit	long
drmaa_run_duration_hlimit	hardRunDurationLimit	long
drmaa_run_duration_slimit	softRunDurationLimit	long

Table 1: DRMAA Attributes

In languages which support introspection, some methods from the DRMAA specification are unnecessary. The drmaa_get_attribute(), drmaa_set_attribute(), drmaa_get_vector_attribute(), drmaa_set_vector_attribute(), and drmaa_get_vector_attribute_names() methods are not needed because introspective languages are able to obtain a list of property names from the JobTemplate.getAttributeNames() method and use introspection to locate the appropriate getters and getters. The getters and setters can also be directly accessed.

All property setters and getters operate in a pass-by-value mode. For data types which are not natively pass-by-value, such as org.ggf.drmaa.FileTransferMode, the data is copied so that the data structure stored by the implemention is uncoupled from the data structure in the calling application.

Optional attributes are also represented by setters and getters. An object-oriented binding implementation must provide implementations for setters and getters for all DRMAA properties, both required and optional. The setter and getter implementations for optional attributes must throw org.ggf.drmaa.UnsupportedAttributeException in languages which supports exceptions. In languages which do not support exceptions, the optional attribute setters and getters should

return some form of error. The service provider implementation should then override the setters and getters for supported optional attributes with methods that operate normally.

The JobTemplate.getAttributeNames() method returns the names of all properties supported by the service provider implementation, including required, optional, and implementation specific attributes. In order to get the values for supported attributes, such as in a property sheet, one should use introspection to call the appropriate setter and getter for each attribute.

For languages which do not support introspection, the setAttribute() and getAttribute() methods should be used.

4.1 The Session Interface

The main class in an object-oriented language binding is the Session class. It represents the majority of the functionality defined by the DRMAA specification. It has the following structure:

```
public abstract class Session {
    public static final String JOB_IDS_SESSION_ALL,
    public static final String JOB_IDS_SESSION_ANY,
    public static final long TIMEOUT_WAIT_FOREVER,
    public static final long TIMEOUT_NO_WAIT,
    public abstract void init(String contactString)
    public abstract void exit()
    public abstract org.ggf.drmaa.JobTemplate createJobTemplate()
    public abstract void
       deleteJobTemplate(org.ggf.drmaa.JobTemplate jobTemplate)
    public abstract String runJob(org.ggf.drmaa.JobTemplate jobTemplate)
    public abstract List
       runBulkJobs(org.ggf.drmaa.JobTemplate jobTemplate,
       int beginIndex, int endIndex, int step)
    public abstract void control(java.lang.String jobName,
       JobControlAction operation)
    public abstract void synchronize(List jobList, long timeout,
       boolean dispose)
    public abstract org.ggf.drmaa.JobInfo wait(String jobName,
       long timeout)
    public abstract JobProgramState getJobProgramStatus(String jobName)
    public abstract String getContact()
    public abstract org.ggf.drmaa.Version getVersion()
    public abstract String getDRMSInfo()
    public abstract String getDRMAAImplementation()
}
```

The Session class makes use of the following two enumerations:

```
public enum JobControlAction {
    SUSPEND,
    RESUME,
    HOLD,
    RELEASE,
    TERMINATE
}
```

```
public enum JobProgramState {
    UNDETERMINED,
    QUEUED_ACTIVE,
    SYSTEM_ON_HOLD,
    USER_ON_HOLD,
    USER_SYSTEM_ON_HOLD,
    RUNNING,
    SYSTEM_SUSPENDED,
    USER_SUSPENDED,
    USER_SYSTEM_SUSPENDED,
    DONE,
    FAILED}
}
```

If a language does not support enumerations, the above enumerations should have their members treated as integer contants of the Session class, e.g. “int Session.UNDETERMINED.”

4.1.1 SUSPEND

The **SUSPEND** value is used by the **control()** method to indicate that the given job should be suspended.

4.1.2 RESUME

The **RESUME** value is used by the **control()** method to indicate that the given job should be resumed.

4.1.3 HOLD

The **HOLD** value is used by the **control()** method to indicate that the given job should be placed into a hold state.

4.1.4 RELEASE

The **RELEASE** value is used by the **control()** method to indicate that the given job should be released from its hold state.

4.1.5 TERMINATE

The **TERMINATE** value is used by the **control()** method to indicate that the given job should be terminated.

4.1.6 JOB_IDS_SESSION_ALL

The **JOB_IDS_SESSION_ALL** constant is used by the **control()** and **synchronize()** methods to indicate that the methods should operate on all jobs currently in the **RUNNING** state in the session.

4.1.7 JOB_IDS_SESSION_ANY

The **JOB_IDS_SESSION_ANY** constant is used by the **wait()** method to indicate that the method may operate on any job currently in the **RUNNING** state in the session.

4.1.8 TIMEOUT_WAIT_FOREVER

The **TIMEOUT_WAIT_FOREVER** constant is used by the **wait()** and **synchronize()** methods to indicate that the methods should not return until the given job or jobs have entered the **DONE** or **FAILED** state.

4.1.9 TIMEOUT_NO_WAIT

The **TIMEOUT_NO_WAIT** constant is used by the **wait()** and **synchronize()** methods to indicate that the methods should return immediately if the given job or jobs have not yet entered the **DONE** or **FAILED** state.

4.1.10 UNDETERMINED

The **UNDETERMINED** value is used by the **getJobProgramStatus()** method to indicate that the job's current state cannot be determined.

4.1.11 QUEUED_ACTIVE

The **QUEUED_ACTIVE** value is used by the **getJobProgramStatus()** method to indicate that the job is queued, waiting to be scheduled.

4.1.12 SYSTEM_ON_HOLD

The **SYSTEM_ON_HOLD** value is used by the **getJobProgramStatus()** method to indicate that the job has been placed on hold by the system or administrator.

4.1.13 USER_ON_HOLD

The **USER_ON_HOLD** value is used by the **getJobProgramStatus()** method to indicate that the job has been placed on hold by a user.

4.1.14 USER_SYSTEM_ON_HOLD

The **USER_SYSTEM_ON_HOLD** value is used by the **getJobProgramStatus()** method to indicate that the job has been placed on hold by both the system or administrator and a user.

4.1.15 RUNNING

The **RUNNING** value is used by the **getJobProgramStatus()** method to indicate that the job has been scheduled and is running.

4.1.16 SYSTEM_SUSPENDED

The **SYSTEM_SUSPENDED** value is used by the **getJobProgramStatus()** method to indicate that the job has been suspended by the system or administrator.

4.1.17 USER_SUSPENDED

The **USER_SUSPENDED** value is used by the **getJobProgramStatus()** method to indicate that the job has been suspended by a user.

4.1.18 USER_SYSTEM_SUSPENDED

The **USER_SYSTEM_SUSPENDED** value is used by the **getJobProgramStatus()** method to indicate that the job has been suspended by both the system or administrator and a user.

4.1.19 DONE

The **DONE** value is used by the **getJobProgramStatus()** method to indicate that the job has finished normally.

4.1.20 FAILED

The **FAILED** value is used by the **getJobProgramStatus()** method to indicate that the job exited abnormally before finishing.

4.1.21 init

The **init()** method is used to initialize a DRMAA session for use. The *contact* parameter is an implementation-dependent string that may be used to specify which DRM system to use. This method must be called before any other DRMAA calls, except for **getDrmSystem()**, **getDrmaaImplementation()**, and **getContact()**.
If contact is `null`, the default DRM system is used, provided there is only one DRMAA implementation available. If there is more than one DRMAA implementation available, **init()** throws a NoDefaultContactStringSelectedException or returns a corresponding error code is exceptions aren't supported. **init()** should be called only once, by only one of the threads. The main thread is recommended. A call to **init()** by another thread or additional calls to **init()** by the same thread with throw a SessionAlreadyActiveException or return a corresponding error code if exceptions are not supported.

4.1.21.1 Parameters

`contact` - implementation-dependent string that may be used to specify which DRM system to use. If `null`, will select the default DRM if there is only one DRMAA implementation available.

4.1.21.2 Exceptions

`DrmaaException` - May be be one of the following:

- DrmsInitException – failed while initializing the session.
- InvalidContactStringException – the *contact* parameter is invalid.
- AlreadyActiveSessionException – the session has already been initialized.

- DefaultContactStringException – the *contact* parameter is `null` and the default contact string could not be used to connect to the DRMS.
- NoDefaultContactStringSelectedException – the *contact* parameter is `null` and more than one DRMAA implementation is available.

4.1.22 exit

The **exit()** is used to disengage from DRM and allow the DRMAA implementation to perform any necessary internal cleanup. This routine ends the current DRMAA session but doesn't affect any jobs (e.g., queued and running jobs remain queued and running). **exit()** should be called only once, by only one of the threads. Additional calls to **exit()** beyond the first will throw a NoActiveSessionException or return a corresponding error code if exceptions aren't supported.

4.1.22.1 Exceptions

`DrmaaException` - May be one of the following:
- DrmsExitException – failed while exiting the session.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called

4.1.23 createJobTemplate

The **createJobTemplate()** method is used to get a new job template. The job template is used to set the environment for jobs to be submitted. Once the job template has been created, it should also be deleted (via **deleteJobTemplate()**) when no longer needed. Failure to do so may result in a memory leak.

4.1.23.1 Returns

The **createJobTemplate()** method returns a blank JobTemplate object.

4.1.23.2 Exceptions

`DrmaaException` - May be one of the following:
- DrmCommunicationException – unable to communication with the DRMS

4.1.24 deleteJobTemplate

The **deleteJobTemplate()** method is used to deallocate a job template. This routine has no effect on running jobs.

4.1.24.1 Parameters

`jt` - the JobTemplate to delete.

4.1.24.2 Exceptions

`DrmaaException` - May be one of the following:
- DrmCommunicationException – unable to communication with the DRMS

4.1.25 runJob

The **runJob()** method is used to submit a job with attributes defined in the job template, *jt*. The returned job identifier is a String identical to that returned from the underlying DRM system.

4.1.25.1 Parameters

`jt` - the job template to be used to create the job.

4.1.25.2 Returns

The **runJob()** method returns a job identifier String identical to that returned from the underlying DRM system.

4.1.25.3 Exceptions

`DrmaaException` - May be one of the following:

- TryLaterException – the request could not be processed due to excessive system load.
- DeniedByDrmException – the DRMS rejected the job. The job will never be accepted due to job template or DRMS configuration settings.
- DrmCommunicationException – unable to communication with the DRMS
- AuthorizationException – the user does not have permission to submit jobs

4.1.26 runBulkJobs

The **runBulkJobs()** method is used to submit a set of parametric jobs, dependent on the implied loop index, each with attributes defined in the job template, *jt*. The returned job identifiers are Strings identical to those returned from the underlying DRM system.

The JobTemplate class defines a **PARAMETRIC_INDEX** placeholder for use in specifying paths. This placeholder is used to represent the individual identifiers of the tasks submitted through this method.

4.1.26.1 Parameters

`start` - the starting value for the loop index.
`end` - the terminating value for the loop index.
`incr` - the value by which to increment the loop index each iteration.
`jt` - the job template to be used to create the job.

4.1.26.2 Returns

The **runJob()** method returns a list of job identifier Strings identical to that returned by the underlying DRM system

4.1.26.3 Exceptions

`DrmaaException` - May be one of the following:

- TryLaterException – the request could not be processed due to excessive system load.
- DeniedByDrmException – the DRMS rejected the job. The job will never be accepted due to job template or DRMS configuration settings.

- DrmCommunicationException – unable to communication with the DRMS
- AuthorizationException – the user does not have permission to submit jobs

4.1.27 control

The **control()** method is used to hold, release, suspend, resume, or kill the job identified by *jobId*. If *jobId* is **JOB_IDS_SESSION_ALL**, then this routine acts on all jobs submitted during this DRMAA session up to the moment **control()** is called. To avoid thread races in multithreaded applications, the DRMAA implementation user should explicitly synchronize this call with any other job submission calls or control calls that may change the number of remote jobs.

The legal values for *action* and their meanings are:

- JobControlAction.**SUSPEND**: stop the job,
- JobControlAction.**RESUME**: (re)start the job,
- JobControlAction.**HOLD**: put the job on-hold,
- JobControlAction.**RELEASE**: release the hold on the job, and
- JobControlAction.**TERMINATE**: kill the job.

This method returns once the action has been acknowledged by the DRM system, but does not necessarily wait until the action has been completed.

Some DRMAA implementations may allow this method to be used to control jobs submitted external to the DRMAA session, such as jobs submitted by other DRMAA session in other DRMAA implementations or jobs submitted via native utilities.

4.1.27.1 Parameters

`jobId` - The id of the job to control.
`action` - the control action to be taken.

4.1.27.2 Exceptions

`DrmaaException` - May be one of the following:
- DrmCommunicationException – unable to communication with the DRMS.
- AuthorizationException – the user does not have permission to modify jobs.
- ResumeInconsistentStateException – the job is not in a state from which is can be resumed.
- SuspendInconsistentStateException – the job is not in a state from which is can be suspended.
- HoldInconsistentStateException – the job is not in a state from which is can be held.
- ReleaseInconsistentStateException – the job is not in a state from which is can be released.
- InvalidJobException – the job id does not represent a valid job.

4.1.28 synchronize

This method waits until all jobs specified by *jobIds* have finished execution. If *jobIds* contains **JOB_IDS_SESSION_ALL**, then this method waits for all jobs submitted during this DRMAA session up to the moment **synchronize()** is called. To avoid thread race conditions in

multithreaded applications, the DRMAA implementation user should explicitly synchronize this call with any other job submission calls or control calls that may change the number of remote jobs.

To prevent blocking indefinitely in this call, the caller may use a timeout specifying after how many seconds to block in this call. The value **TIMEOUT_WAIT_FOREVER** may be specified to wait indefinitely for a result. The value **TIMEOUT_NO_WAIT** may be specified to return immediately if no result is available. If the call exits before the timeout has elapsed, all the jobs have been waited on or there was an interrupt. If the invocation exits on timeout, an ExitTimeoutException is thrown or a corresponding error code is return if exceptions aren't supported. The caller should check system time before and after this call in order to be sure of how much time has passed.

The *dispose* parameter specifies how to treat the reaping of the remote job's internal data record, which includes a record of the job's consumption of system resources during its execution and other statistical information. If set to `true`, the DRM will dispose of the job's data record at the end of the **synchroniize()** call. If set to `false`, the data record will be left for future access via the **wait()** method.

4.1.28.1 Parameters

`jobIds` - the ids of the jobs to synchronize.
`timeout` - the maximum number of seconds to wait.
`dispose` - specifies how to treat reaping information.

4.1.28.2 Exceptions

`DrmaaException` - May be one of the following:

- DrmCommunicationException – unable to communication with the DRMS.
- AuthorizationException – the user does not have permission to sychonize against jobs.
- ExitTimeoutException – the call was interrupted before all given jobs finished.
- InvalidJobException – the job id does not represent a valid job.

4.1.29 wait

This method will wait for a job with *jobId* to finish execution or fail. If the special string, **JOB_IDS_SESSION_ANY**, is provided as the *jobId*, this routine will wait for any job from the session. This routine is modeled on the wait3 POSIX routine.

The *timeout* value is used to specify the desired behavior when a result is not immediately available. The value, **TIMEOUT_WAIT_FOREVER**, may be specified to wait indefinitely for a result. The value, **TIMEOUT_NO_WAIT**, may be specified to return immediately if no result is available. Alternatively, a number of seconds may be specified to indicate how long to wait for a result to become available.

If the call exits before timeout, either the job has been waited on successfully or there was an interrupt. If the invocation exits on timeout, an ExitTimeoutException is thrown or a corresponding error code is returned if exceptions aren't supported. The caller should check system time before and after this call in order to be sure how much time has passed.

The routine reaps job data records on a successful call, so any subsequent calls to **wait()** will fail, throwing an InvalidJobException, meaning that the job's data record has been already reaped. This exception is the same as if the job were unknown. (The only case where **wait()** can be

successfully called on a single job more than once is when the previous call to **wait()** timed out before the job finished.)

When successful, the resource usage information for the job is provided as a Map of usage parameter names and their values. The values contain the amount of resources consumed by the job and are implementation defined.

4.1.29.1 Parameters

`jobId` - the id of the job for which to wait.
`timeout` - the maximum number of seconds to wait.

4.1.29.2 Returns

This method returns the resource usage and status information as a JobInfo object.

4.1.29.3 Exceptions

`DrmaaException` - May be one of the following:

- DrmCommunicationException – unable to communication with the DRMS.
- AuthorizationException – the user does not have permission to wait for a job.
- NoResourceUsageDataException – the resource usage information for the given job is unavailable.
- ExitTimeoutException – the call was interrupted before the given job finished.
- InvalidJobException – the job id does not represent a valid job.

4.1.30 getJobProgramStatus

The **getJobProgramStatus()** method returns the program status of the job identified by *jobId*. The possible values returned from this method are:

- **UNDETERMINED**: process status cannot be determined,
- **QUEUED_ACTIVE**: job is queued and active,
- **SYSTEM_ON_HOLD**: job is queued and in system hold,
- **USER_ON_HOLD**: job is queued and in user hold,
- **USER_SYSTEM_ON_HOLD**: job is queued and in user and system hold,
- **RUNNING**: job is running,
- **SYSTEM_SUSPENDED**: job is system suspended,
- **USER_SUSPENDED**: job is user suspended,
- **DONE**: job finished normally, and
- **FAILED**: job finished, but failed.

The DRMAA implementation should always get the status of the job from the DRM system unless the status has already been determined to be **FAILED** or **DONE** and the status has been successfully cached. Terminated jobs return a **FAILED** status.

4.1.30.1 Parameters

`jobId` - the id of the job whose status is to be retrieved.

4.1.30.2 Returns

The **getJobProgramStatus()** method returns the program status.

4.1.30.3 Exceptions

`DrmaaException` - May be one of the following:

- DrmCommunicationException – unable to communication with the DRMS.
- AuthorizationException – the user does not have permission to query for a job's status.
- InvalidJobException – the job id does not represent a valid job.

4.1.31 getContact

If called before **init()**, this method returns a comma delimited String containing the contact Strings available from the default DRMAA implementation, one element per DRM system available. If called after **init()**, this method returns the contact String for the DRM system to which the session is attached. The returned String is implementation dependent.

4.1.31.1 Returns

This method returns the current contact information for the DRM system or a comma delimited list of possible contact Strings.

4.1.32 getVersion

This method returns a Version object containing the major and minor version numbers of the DRMAA library. For DRMAA 1.0, major is 1 and minor is 0.

4.1.32.1 Returns

This method returns the version number as a Version object.

4.1.33 getDRMSInfo

If called before **init()**, this method returns a comma delimited list of DRM systems, one element per DRM system implementation provided. If called after **init()**, this method returns the selected DRM system. The returned String is implementation dependent.

4.1.33.1 Returns

This method returns the DRM system information.

4.1.34 getDRMAAImplementation

If called before **init()**, this method returns a comma delimited list of DRMAA implementations, one element for each DRMAA implementation provided. If called after **init()**, this method returns the selected DRMAA implementation. The returned String is implementation dependent and may contain the DRM system as a component.

4.1.34.1 Returns

This method returns the DRMAA implementation information.

4.2 The SessionFactory Class

In order to enable an object-oriented language binding implementation to be supported by multiple different vendors, a factory class is needed to allow a DRMAA application to retrieve a vendor specific implementation of the Session interface. The SessionFactory class serves this purpose and additionally allows the vendor the freedom to return different Session implementations depending on the need. The structure of the SessionFactory class is as follows:

```
public abstract class com.sun.grid.drmaa.SessionFactory {
    public static com.sun.grid.drmaa.SessionFactory getFactory()
    public abstract com.sun.grid.drmaa.Session getSession()
}
```

It is likely that with a future version of this specification, the SessionFactory class will be expanded to included a method to explicitly request a specific service provider implementation, or it will be replaced with a form of driver architecture.

4.2.1 getFactory

This method returns a Session object appropriate for the DRM in use.

4.2.1.1 Returns

This method returns a Session object appropriate for the DRM in use.

4.2.2 getSession

This method returns a SessionFactory object appropriate for the DRM in use.

4.2.2.1 Returns

This method returns a SessionFactory object appropriate for the DRM in use.

4.3 The JobTemplate Class

In order to define the attributes associated with a job, a DRMAA application uses the JobTemplate class. JobTemplates are created via the active Session implementation. A DRMAA application gets a JobTemplate from the active Session, specifies in the JobTemplate any required job parameters, and the passes the JobTemplate back to the Session when requesting that a job be executed. When finished, the DRMAA application should call the `Session.deleteJobTemplate()` method to allow the underlying implementation to free any resources bound to the JobTemplate object. The structure of the JobTemplate class is as follows:

```
public class org.ggf.drmaa.JobTemplate {
    public static final int HOME_DIRECTORY
    public static final int WORKING_DIRECTORY
    public static final int PARAMETRIC_INDEX
    public JobTemplate()
    public String remoteCommand {get, set}
```

```
    public String[] args {get, set}
    public JobSubmissionState jobSubmissionState {get, set}
    public Dictionary jobEnvironment {get, set}
    public String workingDirectory {get, set}
    public String jobCategory {get, set}
    public String nativeSpecification {get, set}
    public String[] email {get, set}
    public boolean blockEmail {get, set}
    public org.ggf.drmaa.PartialTimestamp startTime {get, set}
    public String jobName {get, set}
    public String inputPath {get, set}
    public String outputPath {get, set}
    public String errorPath {get, set}
    public boolean joinFiles {get, set}
    public org.ggf.drmaa.FileTransferMode transferFiles {get, set}
    public org.ggf.drmaa.PartialTimestamp deadlineTime {get, set}
    public long hardWallclockTimeLimit {get, set}
    public long softWallClockTimeLimit {get, set}
    public long hardRunDurationLimit {get, set}
    public long softRunDurationLimit {get, set}
    public abstract List getAttributeNames()
}
```

The JobTemplate class uses the following enumeration:

```
public enum JobSubmissionState {
    public static final int HOLD
    public static final int ACTIVE
}
```

If a language does not support enumerations, the above enumeration should have it members treated as integer contants of the Session class, e.g. “int Session.HOLD.”

4.3.1 HOLD

The **HOLD** enumeration represents a value for the jobSubmissionState property which means the job may be queued but it is not eligible to run.

4.3.2 ACTIVE

The **ACTIVE** enumeration represents a value for the jobSubmissionState property which means the job may be queued but it is not eligible to run.

4.3.3 HOME_DIRECTORY

The **HOME_DIRECTORY** constant is a place holder uesd to represent the user's home directory when building paths for the workingDirectory, inputPath, outputPath, and errorPath properties.

4.3.4 WORKING_DIRECTORY

The **WORKING_DIRECTORY** constant is a place holder uesd to represent the current working directory when building paths for the inputPath, outputPath, and errorPath properties.

4.3.5 PARAMETRIC_INDEX

The **PARAMETRIC_INDEX** constant is a place holder uesd to represent the id of the current parametric job subtask when building paths for the workingDirectory, inputPath, outputPath, and errorPath properties.

4.3.6 JobTemplate

The **JobTemplate()** constructor creates a new instance of a JobTemplate. In most cases, a DRMAA implementation will require that JobTemplates be created through the Session.**createJobTemplate()** method, however. In those cases, passing a JobTemplate created through the **JobTemplate()** constructor to the Session.**deleteJobTemplate()**, Session.**runJob()**, or Session.**runBulkJobs()** methods will result in an InvalidJobTemplateException being thrown or a corresponding error code being returned if exceptions are not supported.

4.3.7 getAttributeNames

This method returns the list of supported property names. This list includes supported DRMAA reserved property names (both required and optional) and DRM-specific property names.

4.3.7.1 Returns

This method returns the list of supported property names.

4.3.8 Comparable

A JobTemplate object must be comparable to another JobTemplate object for equality. This comparability should be accomplished through whatever mechasnism is most natural for the implementation language.

4.3.9 Printable

A JobTemplate object must be convertible to a String for printing. This convertability should be accomplished through whatever mechasnism is most natural for the implementation language. The resulting String should contain the values of all set properties.

4.3.10 Getters

For each property listed in Table 1: DRMAA Attributes, the JobTemplate class has a corresponding getter. Each getter is of the form “public <propertyType> get<propertyName>().”

4.3.10.1 Returns

The getter methods all return the current value of the corresponding property in the job template. All non-primitive, mutable return values are copies of the originals.

4.3.11 Setters

For each property listed in Table 1: DRMAA Attributes, the JobTemplate class has a corresponding setter. Each setter is of the form "public void set<propertyName>(<propertyType> value)." Setters for non-primitve, mutable properties store a copy of the new value instead of storing the original object.

4.3.11.1 Parameters

value – the value to which the property should be set in the job template.

4.3.11.2 Exceptions

`DrmaaException` - May be one of the following:

- InvalidAttributeValueException – the property value is invalid for the property, e.g. a startTime that is in the past.
- ConflictingAttributeValuesException – the property value conflicts with a previously set property value.

4.4 The JobInfo Class

The information regarding a job's execution is encapsulated in the JobInfo class. Via the JobInfo class a DRMAA application can discover information about the resource usage and exit status of a job. The structure of the JobInfo class is as follows:

```
public abstract class org.ggf.drmaa.JobInfo {
    public String jobId {get}
    public Dictionary resourceUsage {get}
    public abstract boolean ifExited()
    public abstract int getExitStatus()
    public abstract boolean ifSignaled()
    public abstract java.lang.String getTerminatingSignal()
    public abstract boolean coreDump()
    public abstract boolean ifAborted()
}
```

4.4.1 getJobId

This method returns the completed job's id.

4.4.1.1 Returns

This method returns the completed job's id.

4.4.2 getResourceUsage

This method returns the completed job's resource usage data.

4.4.2.1 Returns

This method returns the completed job's resource usage data.

4.4.3 ifExited

This method returns `true` if the job terminated normally. `False` can also indicate that although the job has terminated normally an exit status is not available or that it is not known whether the job terminated normally. In both cases **getExitStatus()** doesn't provide exit status information. `True` indicates more detailed diagnosis can be provided by means of **getExitStatus()**.

4.4.3.1 Returns

This method returns a boolean indicating whether the job has exited.

4.4.4 getExitStatus

If **ifExited()** returns `true`, this function returns the exit code that the job passed to _exit() (see exit(2)) or exit(3C)), or the value that the child process returned from main.

4.4.4.1 Returns

This method returns the exit code for the job.

This method should throw an exception if ifExited() is false, but which one???

4.4.5 ifSignaled

The **ifSignaled()** method returns `true` if the job terminated due to the receipt of a signal. `False` can also indicate that although the job has terminated due to the receipt of a signal, the signal is not available or that it is not known whether the job terminated due to the receipt of a signal. In both cases **getTerminatingSignal()** does not provide signal information.

4.4.5.1 Returns

This method returns a boolean indicating whether the job terminated due to a signal.

4.4.6 getTerminatingSignal

If **ifSignaled()** returns `true`, this method returns a representation of the signal that caused the termination of the job. For signals declared by POSIX, the symbolic names are be returned (e.g., SIGABRT, SIGALRM).
For signals not declared by POSIX, a DRM dependent string is returned.

4.4.6.1 Returns

This method returns the name of the terminating signal.

This method should throw an exception if ifSignaled() is false, but which one???

4.4.7 coreDump

If **ifSignaled()** returns `true`, this function returns `true` if a core image of the terminated job was created.

This method should throw an exception if ifSignaled() is false, but which one???

4.4.7.1 Returns

This method returns a boolean indicating whether a core image of the terminated job was created.

4.4.8 ifAborted

This method returns `true` if the job ended before entering the running state.

4.4.8.1 Returns

This method returns a boolean indicating whether the job ended before entering the running state.

4.4.9 Serializable

It must be possible to serialize a JobInfo object for storage and retrieval. This serializability should be accomplished through whatever mechasnism is most natural for the implementation language.

4.5 The PartialTimestamp Class

The PartialTimestamp class is used by the JobTemplate class to represent partially specified time stamps, as required by the Distributed Resource Management Application API Specification 1.0. The PartialTimestamp should be an extension of the native date/time class if possible and reasonable.
The PartialTimestamp must support the following fields: century (>=19), year (0-99), month (1-12), date (1-31), hour (0-23), minute (0-59), second (0-61), zone offset hour (-11-12), and zone offset minute (0-59). It must also support six essential operations: “get field value,” “set field value,” “get time as milliseconds,” “get time as native date/time object,” “covert to string,” and “parse from string.” If possible, these operations should leverage structure already present in the native date/time class. The two field operations may be represented as properties.
The “get field value” operation should return the current value for the given field. The “set field value” operation should return the current value for the given field. The “get time as milliseconds” operation should resolve the partial time to a concrete time that is the soonest possible that is not in the past, and should return that concrete time as the number of milliseconds since the epoch. The “get time as native date/time object” operation must resolve the partial time to a concrete time that is the soonest possible that is not in the past, and should return that concrete time as a native date/time object. The “convert to string” operation must return the partial time as a String which adheres to the following format: `[[[[CC]YY/]MM/]DD] hh:mm[:ss] [{-|+}UU:uu]`, where:

- CC is the first two digits of the year [19,]
- YY is the last two digits of the year [0,99]
- MM is the two digits of the month [01,12]
- DD is the two-digit day of the month [01,31]

- hh is the two-digit hour of the day [00,23]
- mm is the two-digit minute of the day [00,59]
- ss is the two-digit second of the minute [00,61]
- UU is the two-digit hours since (before) UTC [-11,12]
- uu is the two-digit minutes since (before) UTC [0,59]

In order for this operation to be performed, the PartialTimestamp object must have no unset field of a lower order than the highest order set field, with the exception of second and the zone offsets. For example, if the year is set, the month, date, hour, and minute must also be set for this operation to be performed. Failure to meet this criterion must result in an InvalidArgumentException being thrown, or the corresponding error code being returned in languages which do not support exceptions. The "parse from string" operation must parse a string in the above format to generate a PartialTimestamp object, which must be returned. If the string is not in the above format, an InvalidArgumentException must be thrown or the corresponding error code must be returned in languages which do not support exceptions.

The PartialTimestamp may also support the following four operations: "get field modifier," "set field modifier," "add to field," and "roll field." If possible, these operations should leverage structure already present in the native date/time class. The two field operations may be represented as properties. The "get field modifier" operation must return any additional modifiers set for the given field. An additional modifier is added to the field's value after it has been resolved to a concrete time. The "set field modifier" operation must set the additional modifiers for the given field. The "add to field" operation must add a given value to the given field. If supported by the native date/time class, this operation should attempt to resolve out of range field values that may result from the operation. For example, adding "1" to the date of a PartialTimestamp object which is set to January 31st should result in the PartialTimestamp object being set to February 1st. If this operation is supported, the "get field modifier" and "set field modifier" operations must also be supported. The "roll field" operation is similar to the "add to field" operation, except that the operation cannot modify a field of a higher order than the given field. Such modifications are simply lost. For example, adding "1" to the date of a PartialTimestamp object which is set to January 31st should result in the PartialTimestamp object being set to January 1st.

The PartialTimestamp class must also support the **UNSET** constant. The **UNSET** constant is the value assigned to all fields which have not been explicitly set. This constant should be of the same type as the date/time properties and should be the maximum value for that data type.

4.6 The FileTransferMode Class

The FileTransferMode class is used by the JobTemplate class to indicate the value for the transferFiles property. The class has three properties which determine which streams will be staged in or out. See the transferFiles property in the Distributed Resource Management Application API Specification 1.0 for more information. The structure of the FileTransferMode class is as follows:

```
public class org.ggf.drmaa.FileTransferMode {
    public org.ggf.drmaa.FileTransferMode()
    public org.ggf.drmaa.FileTransferMode(boolean inputStream,
       boolean outputStream, boolean errorStream)
    public void setInputStream(boolean inputStream)
    public boolean getInputStream()
    public void setOutputStream(boolean outputStream)
```

```
    public boolean getOutputStream()
    public void setErrorStream(boolean errorStream)
    public boolean getErrorStream()
}
```

4.6.1 setInputStream

This method sets whether to transfer input stream files.

4.6.1.1 Parameters

`inputStream` - whether to transfer input stream files

4.6.2 getInputStream

This method returns a boolean representing whether to transfer input stream files.

4.6.2.1 Returns

This method returns a boolean representing whether to transfer input stream files.

4.6.3 setOutputStream

This method sets whether to transfer output stream files.

4.6.3.1 Parameters

`outputStream` - whether to transfer output stream files

4.6.4 getOutputStream

This method returns a boolean representing whether to transfer output stream files.

4.6.4.1 Returns

This method returns a boolean representing whether to transfer output stream files.

4.6.5 setErrorStream

This method sets whether to transfer error stream files.

4.6.5.1 Parameters

`errorStream` - whether to transfer error stream files

4.6.6 getErrorStream

This method returns a boolean representing whether to transfer error stream files.

4.6.6.1 Returns

This method returns a boolean representing whether to transfer error stream files.

4.6.7 Clonable

It must be possible to make a copy of a FileTransferMode object. This clonability should be accomplished through whatever mechasnism is most natural for the implementation language.

4.6.8 Printable

It must be possible to convert a FileTransferMode object into a string for printing. This convertability should be accomplished through whatever mechasnism is most natural for the implementation language. The resulting String should contain the values of the three stream properties.

4.6.9 Comparable

A FileTransferMode object must be comparable to another FileTransferMode object for equality. This comparability should be accomplished through whatever mechasnism is most natural for the implementation language.

4.6.10 Serializable

It must be possible to serialize a FileTransferMode object for storage and retrieval. This serializability should be accomplished through whatever mechasnism is most natural for the implementation language.

4.7 The Version Class

The Version class is a holding class for the major and minor version numbers of the DRMAA implementation as returned by the DrmaaSession.**getVersion()** method. The class structure follows:

```
public class org.ggf.drmaa.Version {
    public org.ggf.drmaa.Version(int major, int minor)
    public int getMajor()
    public int getMinor()
}
```

4.7.1 getMajor

This method the major version number.

4.7.1.1 Returns

This method the major version number.

4.7.2 getMinor

This method the minor version number.

4.7.2.1 Returns

This method the minor version number.

4.7.3 Clonable

It must be possible to make a copy of a Version object. This clonability should be accomplished through whatever mechasnism is most natural for the implementation language.

4.7.4 Printable

It must be possible to convert a Version object into a string for printing. This convertability should be accomplished through whatever mechasnism is most natural for the implementation language. The resulting String should be of the form "<major>.<minor>".

4.7.5 Comparable

A Version object must be comparable to another Version object for equality. This comparability should be accomplished through whatever mechasnism is most natural for the implementation language.

4.7.6 Serializable

It must be possible to serialize a Version object for storage and retrieval. This serializability should be accomplished through whatever mechasnism is most natural for the implementation language.

4.8 Exceptions

All exceptions in the object-oriented binding inherit from the DrmaaException class. The structure of DrmaaException as as follows:

```
public class com.sun.grid.drmaa.DrmaaException
    extends java.lang.Exception{
    public com.sun.grid.drmaa.DrmaaException()
    public com.sun.grid.drmaa.DrmaaException(java.lang.String message)
}
```

All exceptions under the DrmaaException class have the following structure:

```
public class com.sun.grid.drmaa.<NAME>Exception
    extends DrmaaException{
    public com.sun.grid.drmaa.<NAME>Exception()
    public com.sun.grid.drmaa.<NAME>Exception(java.lang.String message)
}
```

where <NAME> is the name of the exception.

4.8.1 The Exception Hierarchy

The DRMAA exception hierarchy is as follows:

- *org.ggf.drmaa.DrmaaException*
  - org.ggf.drmaa.AuthorizationException
  - org.ggf.drmaa.InvalidContactStringException
    - org.ggf.drmaa.DefaultContactStringException
    - org.ggf.drmaa.NoDefaultContactStringSelectedException
  - org.ggf.drmaa.DeniedByDrmException
  - org.ggf.drmaa.DrmCommunicationException
  - org.ggf.drmaa.DrmsExitException
  - *org.ggf.drmaa.InconsistentStateException*
    - org.ggf.drmaa.HoldInconsistentStateException
    - org.ggf.drmaa.ReleaseInconsistentStateException
    - org.ggf.drmaa.ResumeInconsistentStateException
    - org.ggf.drmaa.SuspendInconsistentStateException
  - org.ggf.drmaa.DrmsInitException
  - org.ggf.drmaa.InvalidArgumentException
    - org.ggf.drmaa.InvalidJobException
  - *org.ggf.drmaa.InvalidAttributeException*
    - org.ggf.drmaa.ConflictingAttributeValuesException
    - org.ggf.drmaa.InvalidAttributeFormatException*
    - org.ggf.drmaa.InvalidAttributeValueException
  - org.ggf.drmaa.NoResourceUsageException
  - org.ggf.drmaa.ExitTimeoutException
  - *org.ggf.drmaa.SessionException*
    - org.ggf.drmaa.NoActiveSessionException
    - org.ggf.drmaa.AlreadyActiveSessionException
  - org.ggf.drmaa.TryLaterException
  - org.ggf.drmaa.InternalException**
    - org.ggf.drmaa.OutOfMemoryException**
  - org.ggf.drmaa.UnsupportedAttributeException**
  - org.ggf.drmaa.InvalidJobTemplateException**

Exceptions listed in itallics exist only for behavior aggregation and must be declared as abstract. Exceptions marked with an asterisk are only useful for languages which do not support introspection. Exceptions marked with two asterisks should be treated as unchecked exceptions in languages where a distinction between checked and unchecked exceptions is made.

Whereever possible, native exceptions should be used in favor of the DRMAA-specific exceptions. For example, most languages which support exceptions, support some native form of the OutOfMemoryException.

4.8.2 AlreadyActiveSessionException

Initialization failed due to existing DRMAA session.

4.8.3 AuthorizationException

The user is not authorized to perform the given operation.

4.8.4 ConflictingAttributeValuesException

The value of this attribute conflicts with one or more previously set properties.

4.8.5 DefaultContactStringException

The DRMAA implementation could not use the default contact string to connect to DRM system.

4.8.6 DeniedByDrmException

The DRM system rejected the job. The job will never be accepted due to DRM configuration or job template settings.

4.8.7 DrmCommunicationException

Could not contact DRM system.

4.8.8 DrmsExitException

A problem was encountered while trying to exit the session.

4.8.9 DrmsInitException

A problem was encountered while trying to initialize the session.

4.8.10 ExitTimeoutException

The Session.**wait()** or Session.**synchronize()** call returned before all selected jobs entered the **DONE** or **FAILED** state.

4.8.11 HoldInconsistentStateException

The job cannot be moved to a **HOLD** state.

4.8.12 InternalException

Unexpected or internal DRMAA error like system call failure, etc.

4.8.13 InvalidArgumentException

A parameter value is fundamentally invalid, such as being of the wrong type or being `null`.

4.8.14 InvalidAttributeFormatException

The value for the job template property is improperly formatted, such as a badly formatted time stamp.

4.8.15 InvalidAttributeValueException

The value for the job template property is invalid.

4.8.16 InvalidJobException

The job specified by the given job id does not exist.

4.8.17 InvalidJobTemplateException

The job template is not valid. It was either created incorrectly, i.e. not via Session.**createJobTemplate()**, or it has been deleted via Session.**deleteJobTemplate()**.

4.8.18 NoActiveSessionException

Method call failed because there is no active session.

4.8.19 NoDefaultContactStringSelectedException

No defaults contact string was provided or selected. DRMAA requires that the default contact string is selected when there is more than one default contact string due to multiple DRMAA implementations being present and available.

4.8.20 NoResourceUsageException

This exception is thrown by Session.**wait()** when a job has finished but no resource usage or exit status data could be provided.

4.8.21 OutOfMemoryException

This exception can be throw by any method at any time when the DRMAA implementation has run out of free memory.

4.8.22 ReleaseInconsistentStateException

The job is not in a **HOLD** state, and hence cannot be released.

4.8.23 ResumeInconsistentStateException

The job is not in a ***_SUSPENDED** state, and hence cannot be resumed.

4.8.24 SuspendInconsistentStateException

The job is not is a state from which it can be suspended.

4.8.25 TryLaterException

The DRMS rejected the operation due to excessive load. A retry attempt may succeed, however.

4.8.26 UnsupportedAttributeException

The given job template proprety is not supported by the current DRMAA implementation.

4.8.27 Correlation to Error Codes

The following table shows how the error codes defined in the Distributed Resource Management Application API Specification 1.0 correlate to execeptions in the Distributed Resource Management Application API Object-Oriented Bindings 0.1.

Error Code Name (DRMAA_ERRNO_...)	*Exception Name (org.ggf.drmaa....)*
SUCCESS	none
INTERNAL_ERROR	InternalException
DRM_COMMUNICATION_FAILURE	DrmCommunicationException
AUTH_FAILURE	AuthorizationException
INVALID_ARGUMENT	java.lang.IllegalArgumentException
NO_ACTIVE_SESSION	NoActiveSessionException
NO_MEMORY	java.lang.OutOfMemoryError
INVALID_CONTACT_STRING	InvalidContactStringException
DEFAULT_CONTACT_STRING_ERROR	DefaultContactStringException
DRMS_INIT_FAILED	DrmsInitException
ALREADY_ACTIVE_SESSION	AlreadyActiveSessionException
DRMS_EXIT_ERROR	DrmsExitException
INVALID_ATTRIBUTE_FORMAT	InvalidAttributeFormatException
INVALID_ATTRIBUTE_VALUE	InvalidAttributeValueException
CONFLICTING_ATTRIBUTE_VALUES	ConflictingAttributeValuesException
TRY_LATER	TryLaterException
DENIED_BY_DRM	DeniedByDrmException
INVALID_JOB	InvalidJobException
RESUME_INCONSISTENT_STATE	ResumeInconsistentStateException
SUSPEND_INCONSISTENT_STATE	SuspendInconsistentStateException
HOLD_INCONSISTENT_STATE	HoldInconsistentStateException
RELEASE_INCONSISTENT_STATE	ReleaseInconsistentStateException
EXIT_TIMEOUT	ExitTimeoutException

Error Code Name (DRMAA_ERRNO_...)	*Exception Name (org.ggf.drmaa....)*
NO_RUSAGE	NoResourceUsageException
none	InvalidJobTemplateException
none	UnsupportedAttributeException

Table 2: Correlating Error Codes to Exceptions

The DRMAA_ERRNO_SUCCESS code clearly does not need to be represented as an exception. The object-oriented binding specification introduces two new exceptions which have no error code correlaries. The InvalidJobTemplateException is used to indicate that the job template object currently being used is not valid. This may be, for example, because it has already been deleted via Session.**deleteJobTemplate()**. The UnsupportedAttributeException is used to indicated that for the current DRMAA implementation the given property is unsupported.

5 Security Considerations

Security issues are not discussed in this document. The scheduling scenario described here assumes that security is handled at the point of job authorization/execution on a particular resource.

6 Author Information

Daniel Templeton
dan.templeton@sun.com
Sun Microsystems GmbH
Dr.-Leo-Ritter-Str. 7
D-93049 Regensburg
Germany

Peter Tröger
peter.troeger@hpi.uni-potsdam.de
Universität Potsdam
Am Neuen Palais 10
D-14469 Potsdam
Germany

7 Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

8 Full Copyright Notice