GWD-R (in-process draft)

Distributed Resource Management
Application API (DRMAA) Working Group

Hrabri Rajic, Intel Americas Inc. (maintainer)
Roger Brobst, Cadence Design Systems
Waiman Chan, IBM
Fritz Ferstl, Sun Microsystems
Jeff Gardiner, John P. Robarts Res. Inst.
Andreas Haas, Sun Microsystems
Bill Nitzberg, Veridian Systems
Hrabri Rajic*, Intel Americas Inc.
John Tollefsrud*, Sun Microsystems
*co-chairs
December, 2002

Distributed Resource Management Application API Specification

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.

Copyright Notice

Abstract

This document provides description of Distributed Resource Management Application API (DRMAA) that hides the differences of Distributed Resource Management systems and provides an API intended for distributed application developers or ISVs. The scope of DRMAA is limited to job submission, job monitoring and control, and retrieving the finished job status. The net result is a programming model that enables distributed application development that is oriented towards developer needs rather than what is available from the underlying DRM system.

Table of Contents

1. Introduction

Distributed Resource Management Application API (DRMAA) hides the differences of the Distributed Resource Management Systems (DRMSs) and provides an API intended for distributed application developers or ISVs. It is one of the DRMAA working group goals to target a very broad audience by providing an easy to use programming model. The mandate of the DRMAA working group is to produce DRMAA specification 1.0.

1.1 DRMAA Charter

This is the Charter for the Global Grid Forum DRMAA Working Group:

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).

1.2 DRMAA Scope

The scope of DRMAA is limited to job submission, job monitoring and control, and retrieving the finished job status. Job reservation, security, etc. are the topics that are beyond DRMAA scope. These topics are handled by other GGF working or research groups.

1.3 Language issues

It is our position that the API should be implemented in multiple languages, C/C++ being the primary choice. The secondary choices are scripting languages, Perl and Python. Perl is especially heavily used in the biotech arena where there is great need for numerous parametric calculations.
It is possible to design an Interface Definition Language that will effectively resolve the issue of one interface serving multiple languages. While this is a viable approach, it was felt that it will slow the progress on the implementation side significantly. DRMAA interfaces are described by using an IDL like language.
Another viable approach would be to design a protocol instead of the API. Both of these alternatives should be considered in more detail when the time comes to address the long term comprehensive solutions.

1.4 Library Issues

An ideal library would have paths to handle all DRMSs and have versions to be linked statically or dynamically. This is not something that will be feasible or even desirable. A real possibility is a situation where one vendor implements multiple, but not all DRMSs. The packaging could come as one library, where a DRMS is selected at run time setting an environmental variable for the desired DRMS, or as one DRMS link per library. The authors advocate the latter approach. In this setup the shared library is selected at run time by the end users.

It is expected that the developers will be linking the library from serial and multithreaded codes. The library should be thread safe.
Debugging of distributed programs is more challenging than debugging of single machine versions. This document advocates providing production and debugging or tracing versions of the DRMAA library.
Library should provide the DRMAA API version number to the external programs (such as SCCS's "what" and RCS's "ident") and to the distributed applications programmatically.

1.5 User program and DRMAA interaction

All of the DRM systems are asynchronous in nature. They notify the end user of the status of a finished job via e-mail, which as an only option is not acceptable to the users of the DRMAA library. This document proposes to deal with the asynchrony similarly to Unix and Windows process interfaces, by blocking on the wait call for a specific job request or possibly to all of them in the same process. This is in contrast to Globus GRAM interface that is modeled on the reactive mode of execution. The support for reactive mode is to be addressed in the future DRMAA versions, because more and more programs come with graphic user interface these days.

The rest of the document is presented as follows. Chapter 2. deals with the API design issues. Chapter 3. contains the DRMAA API specification.

2. API Design Issues

Developers have been using Unix system, fork/exec, popen, and the wait interfaces for years to spawn additional processes and wait for the end of their execution to get their exit codes. Windows has equivalent utilities like CreateProcess and WaitForSingleObject. DRMAA provides its own set of interfaces that are OS neutral. It tries to be consistent with libc interfaces.

2.1 Basic Guidelines

Even though the API should be self-contained, it is not always possible to consolidate all variations of end user and DRMS interactions under the API. For this reason, this document advocates that the developers should provide a way for the end user to specify DRMS particular options. The primary mean to achieve this is through use of “job categories”.

Additionally, there might be a need for possibility that the DRMS specific options are specified at run time as command line options. The end user could loose portability this way, which is a small price to pay to be able to run the application in uncommon configurations. Besides that, DRMAA providers and ISVs are the ones that target multiple DRMSs; the end user does that at a much lesser degree.

The API centers around job_id attribute that is passed back by the DRMS upon job submission. Job_id is used for all the job control and monitoring purposes. An additional attribute, job_name, which is found in all DRMS implementations, is part of the job description. Job_name could be used by the developer and/or internally by the implementation to group the jobs for easier user classification and tracking. This attribute could be a key to achieve scalability for DRMAA implementations, especially since DRMS user jobs could be running concurrently with those of the other DRMS users.

There are few guidelines that were used in designing the uniform API:

- The API calling sequences should be simple and the API set small.
- The routine names should convey the semantics of the routine.
- The set should be as convenient as possible, even with the risk of being forced to emulate some functionality if missing from a DRMS.
- All job or job set manipulation is available without explicit job iterating.
- The server names are hidden, DRMS is a black box.
- The end user could specify native specification options attribute if he/she needs to interact with the DRMS, provided “job categories” is not sufficient.
- The API should be extensible in a sense that future implementation are backward compatible with earlier ones.
- It is expected there will be few implementation specific details. These will need to be documented by DRMAA vendors.

2.2 Distributed Application Environment

DRMAA implementation interfaces and DRMAA programming model are mainly concerned by providing sufficient mechanisms for job submitting, monitoring and controlling, and at the end result evaluating.

Ideally, DRMAA implementation and distributed application to a larger degree, should not be concerned by a particular DRM environment and DRM site policies. Unfortunately, this is not always possible to achieve, so solutions with minimal intrusiveness, such as “job categories” and “native specification” have been proposed. They abstract/aggregate the site-specific policies into simple strings that are interpreted by DRMAA implementations.

2.2.1 File Staging

DRMAA specification 1.0 does not have explicit file staging mechanisms. Setting implementation specific job template attributes could enable file staging, provided the DRMAA implementation and the DRM system supports it.

2.2.2 Job categories

The DRMAA interface specification should allow ISVs to write DRM-enabled applications even though the properties of a concrete DRM installation, in particular the configuration of the DRM system, cannot be known in advance.

Experiences made with integrations based on DRM CLI show that even when the same ISV application is run as a job with the same DRM system the site specific policies in effect differ widely. These policies are typically about questions like:

- what resources are to be used by the job
- preferences where to run the job
- how prior the job should be treated by the DRM scheduler compared to other jobs

For supporting the variety of policies, job specific requests expressed by DRM submit options are very common in the DRM product space.

Despite of these differences between two sites with the "same" job passed to the DRM system the application actually does not change when seen from the perspective of the ISV. Also for the end user who just wants a job to be started nothing changes due to different policies. This is an indication that there must be possibility for hiding these site-specific differences behind the DRMAA interface.

The job "categories concept" is the approach the DRMAA working group recommends for encapsulating site-specific details and completely hiding these details from applications making use of the DRMAA interface. The core of the idea is to have these applications only supplying a string attribute specifying a job category, i.e. a name specifying what kind of application that is to be dispatched by the DRMS. The category name can be used by the DRMAA library to determine site-specific resource and functional requirements of jobs in this category. Such requirements need to be configurable by the site operating a DRM system and deploying an ISV application on top of it.

An example can help to illustrate this idea:

At site A rendering application X is used in a heterogeneous clustered environment which is managed by a DRMS. Since application X is only available at a subset of these machines the administrator sets up the DRMS in a way requiring from the end-users to put a -l X=true into their submit command line.

At site B the same application is used in a homogenous clustered environment with rendering application X supported at all machines managed by the DRMS. However since X jobs do compete with applications Y sharing the same resources and X applications are to be treated with higher priority than Y jobs end-users need to put a -p 1023 into their submit command line for raising the dispatch priority.

An integration based on categories will allow submiting X jobs through the DRMAA interface in compliance with the policies of both sites A and B without the need to know about these policies. The ISV does this by specifying "X" as the category used for X rendering jobs submitted through the DRMAA interface and by mentioning this in the "DRM integration" section of the X rendering software documentation.

The administrators at the sites A and B site read the documentation or installation instructions about the "X" DRMAA category. The documentation of their DRMS contains directions about the category support of their DRMAA interface implementation. From this documentation they learn how to configure their DRMS in a way that "-l X=true" is used for "X" jobs at site A while "-p 1023" is used at site B for those jobs.

As far as the DRMAA interface specification is concerned only a standardized mechanism for specifying the category is required. The mechanism for associating the policy related portion of the submit command line to the job is to be delivered by each DRMAA implementation. A standardization of this mechanism is beyond the DRMAA standardization effort, because it is too much related to the administrative interface and it is anticipated that for different DRMS different mechanisms will be appropriate.

2.2.3 Native specification

The benefit of the categories concept from the last chapter is that it provides a means for completely hiding site-specific policy details to be considered with a DRMAA job submission for a whole class of jobs. The drawback however of this concept is that it requires one job category to be maintained for each policy to be used.

To allow the DRMAA interface to be used also for submission of jobs where job-individual policy specification is required "native specification" is supported. Native specification can be used without the requirement to maintain job categories. Instead of specifying a category name and having the DRMAA implementation associate the corresponding job submit options; the use of native specification will allow directly specifying these submit options.

An example can help to illustrate this idea:

In order to implement the example from section 2.2.2 via native specifications, the native option string "-l X=true" had to be passed directly to the DRMAA interface while "-p 1023" had to be used at site B.

As far as the DRMAA interface specification is concerned the native specification is an opaque string and interpreted by each DRMAA library. It is possible to use job categories and native specification with the same job submission for policy specification. It is assumed that in this case the DRMAA library is capable of joining the outcome of the two policy sources in a reasonable way.

2.2.4 Building Portals

The nature of the DRMAA implementation, as a shared library, makes it a good candidate for inclusion in a Web Server to support a Web Portal to a DRMS.

DRMAA library could be:

- Linked by a collection of CGI scripts that are referenced by resident Web Pages.
- Linked in a Web Server as a separate module.
- Built as a Perl (the same applies to a lesser degree to other scripting languages) module that is
  - included in mod_perl module
  - accessed from Perl CGI scripts

The questions about maintaining state, security, and authentication and authorization, require that DRMAA implementation is viewed as just one component of a DRM Web Portal. Clearly, evaluation of the Web Portal needs and their impact on DRMAA specification is beyond the scope of the current document and DRMAA Charter.

2.3 Interface Routines General Description

The routines are naturally grouped in five categories: init/exit, job template handling, job submission, job monitoring and control, and auxiliary or system routines like trace file specification and error message routines.

All of the routines should return an error code upon exit. A possible exception is an auxiliary error message routine that could be modeled after the standard libc strerror routine. DRMAA needs an equivalent of libc errno value for the internal failures. In libc, errno is a macro that expands to a modifiable lvalue, such as a de-referenced function pointer to address libc use in reentrant mode.

To prevent interface name collisions all the routines have a prefix “drmaa_”.

2.3.1 Init and exit routines

The calling sequence of the init routine should allow all of the considered DRM systems to be properly initialized, either by interfacing to the batch queue commands or to the DRMS API. Likewise, the exit routine should require parameters that will permit proper DRMS disengagement.

2.3.2 Job template routines

The remote jobs and their attributes are specified via job template opaque parameter. The job attributes could be scalar or vector values. They are listed below.

Required job attributes are:

- Remote command to execute
- Remote command input parameters. This is a vector parameter.
- Job state at submission.
- Job environment. This is a vector parameter.
- Job working directory.
- Job category.
- Native specification.
- Standard input, output, and error streams.
- E-mail to report the job completion and status. This is a vector parameter.
- E-mail suppression
- Job start time.
- Job name to be used for the job submission.

2.3.3 Job submission routines

The job submission routines come in two versions. There is one version for submitting individual jobs and one version for submitting bulk jobs.

2.3.4 Job monitoring and controlling routines

Job monitoring and controlling API group needs to handle:

- job releasing, stopping, resuming, and killing
- checking the exit code of the finished remote job
- checking the remote job status
- waiting for the remote job till the end of its execution
- waiting for all the jobs or are subset of the current session jobs to finish execution (this is a useful synchronization mechanism)

The Unix and Windows signals are replaced with the job control routines that have counterparts in DRM systems. The only nontraditional feature is the passing of DRMAA_JOB_IDS_SESSION_ALL string as job_id to indicate operations on all job_ids in the current process.

The remote job could be in following states:

- system hold
- user hold
- system and user hold simultaneously
- queued active
- system suspended
- user suspended
- system and user suspended simultaneously
- running
- finished (un)successfully

To this list we need to add a possibility of DRMAA implementation not being able to determine the status of the remote job. Note that a rejected job was not assigned a job_id and consequently could not have a state.

2.3.5 Auxiliary routines

The auxiliary routines are needed for execution tracing and error textual representation. The tracing is especially useful for the situations when there is multiple processes spawned few levels deep. For this version, DRMAA does not specify how tracing is done.

The next Chapter contains an API as specified here.

3. API Specification

For convenience, the API is divided in its five logical sections: init/exit, job template handling, job submission, job monitoring and control, and auxiliary routines. The interface is specified by using a IDL like language.

The plan is to fully specify C/C++ bindings to insure binary compatibility. It is unknown if the DRMAA API would need extra interfaces if OGSA bindings are to be used. Theoretically, all the DRM or site specific information could be passed to DRMAA implementation via direct side channels.

3.1 Introduction

IDL like language is used here to avoid questions about allocation/de-allocation issues, the questions that are specific to an implementation language. All of the interfaces, except a few that return scalar values and cannot fail, return an error code on exit. Successful return is indicated by return value DRMAA_ERRNO_SUCCESS. All internal errors are indicated with DRMAA_ERRNO_INTERNAL_ERROR error. Invalid argument is flagged as DRMAA_ERRNO_INVALID_ARGUMENT. The return codes are specified and listed in section 3.3.

The interface parameters could be IN, OUT, or INOUT parameters. For readers of this documents who know C they should be thought as parameters passed by value of by reference

respectively. Furthermore, the parameters could be scalar or vector values. The vector values are clearly documented.

The NOTES, text, and ISSUES that are numbered, yellow highlighted, and in blue [italic] font are scheduled for further discussion.

The green font sections need more background and understanding.

General Notes for the DRMAA API section:

C1
ISSUE 1: Extensibility issue with regard to backward compatibility. If DRMAA 2.0 allows multiple connections that means drmaa_session parameter is part of the interface. Visit this after all of the issues have been resolved This is slated for version 2 or higher.

C17
ISSUE 2: Names, interface, variable, etc., and global issues are to be determined when everything else has been completed.

Validity of job Ids across different DRMAA sessions
There is only one DRMAA session open at the time. Another session could be opened only after the current one is closed. This implies that nesting of sessions is not allowed. It is expected that the DRMAA library will free all the session resources, although this is not guaranteed, so old session resources are not to be used later. Job Ids are valid from one session to another. Job control routines should work correctly if a job Id came from a previous DRMAA session, provided the current DRMAA session knows how to resolve the job Id from the previous session. The burden is on the user to match previous job Id's with appropriate DRMAA sessions, i.e. DRM system servers. Re-startable applications have to make job Id's persistent in order to access the already submitted jobs.
DRMAA implementation collects remote run usage data after the remote job run. The user can reap this usage data only once. The implementation is free to "garbage collect" the reaped data at the convenient time. Only the usage data from the current session job Id's is guaranteed to be available. Reaping usage data from other session job Id's is implementation specific.

Precedence rules when the specified job attributes are in collision.
The attributes set by using API routines are set at the compile time. The attributes set via "job categories" are set at the installation time. The attributes set by the "native specification" are set at the run time. In principle that should determine the precedence rules, but these ideal precedence rules are not always achievable in practice due to complex interaction of attributes. Moreover, certain attributes in "job categories" would not be allowed to be overridden.
The precedence rules are therefore implementation specific.

Error handling.
In addition to return exit codes, DRMAA routines provide context specific information about the failures. The return exit code different from success code could be used in drmaa_strerror routine to retrieve textual representation of the error. The context specific error messages are returned as a last parameter.
The user is responsible for providing a buffer of size DRMAA_ERROR_STRING_BUFFER where the context specific error message will be put. The user is also responsible for thread safety of these buffers.

End user local environment specification thru native specification and job categories
Job categories and native specifications are two means for describing site specific requirements. Setting job categories, if they are supported by a DRMAA implementation is implementation specific. On the other hand, setting of the native specification while straightforward in the user code could be a challenge if the user needs to provide a complex set of options. Quotation marks are especially problematic if only one variable is used for a set of native specification options. Here are the recommendations for developers how to use this feature effectively:

- For each class of remote jobs give end user a chance to specify site specific environment, like a queue where to send remote jobs, architecture(s) where the remote applications are available, etc.
- Let users specify native specifications in a file, if the distributed application has several classes of jobs to submit or several DRMAA sessions.
- Applications with a graphical user interface could have a dedicated dialog for this purpose.

3.2 DRMAA API

/* ---------- Major Assumptions/Restrictions ---------- */

Callbacks (asynchronous notification) -- Polling only in v1.0
No explicit file staging.
JobID Uniqueness -- "As unique as the underlying DRM makes them"

/* Global constants */

DRMAA_ERROR_STRING_BUFFER = 1024
DRMAA_TIMEOUT_WAIT_FOREVER -1
DRMAA_TIMEOUT_NO_WAIT 0

/* ---------- Initialization & Exit Routines ---------- */

drmaa_init(contact, drmaa_errno_buf)
IN contact /* contact information for DRM system (string) */

Initialize DRMAA API library and create a new DRMAA Session. 'Contact'
is an implementation dependent string which may be used to specify
which DRM system to use. This routine must be called before any
other DRMAA calls, except for drmaa_version().
If 'contact' is NULL, the default DRM system will be used.

drmaa_init routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_INVALID_CONTACT_STRING or
DRMAA_ERRNO_DEFAULT_CONTACT_STRING_ERROR.

drmaa_exit(drmaa_errno_buf)

Disengage from DRMAA library and allow the DRMAA library to perform
any necessary internal clean up.
This routine ends this DRMAA Session, but does not effect any jobs (e.g.,
queued and running jobs remain queued and running).

drmaa_exit routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_DRMS_EXIT_ERROR or
DRMAA_ERRNO_ALREADY_ACTIVE_SESSION.

drmaa_version(major, minor)
OUT major /* major version number (non-negative integer) */
OUT minor /* minor version number (non-negative integer) */

Returns the major and minor version numbers of the DRMAA library;
for DRMAA 1.0, 'major' is 1 and 'minor' is 0.

DRM_engine drmaa_get_DRM_engine( )
Output (string) is implementation dependent and could contain the DRM engine and the
implementation vendor as its parts.

/* ---------- Job Template Routines ---------- */

drmaa_allocate_job_template(drmaa_errno_buf)
RETURNS /* new job template (opaque handle) */

Allocate a new job template.

drmaa_delete_job_template(jt, , drmaa_errno_buf)
INOUT jt /* job template (opaque handle) */

Deallocate a job template. This routine has no effect on jobs.

drmaa_set_attribute(jt, name, value, drmaa_errno_buf)
INOUT jt /* job template (opaque handle) */
IN name /* attribute name (string) */
IN value /* attribute value (string) */

Adds ('name', 'value') pair to list of attributes in job template 'jt'.
Only non-vector attributes may be passed.

drmaa_set_attribute routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT,
DRMAA_ERRNO_INVALID_ARGUMENT,
DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE, or
DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES.

drmaa_get_attribute(jt, name, value, drmaa_errno_buf)
IN jt /* job template (opaque handle) */
IN name /* attribute name (string) */
OUT value /* attribute value (string) */

If 'name' is an existing non-vector attribute name in the job template
'jt', then the value of 'name' is returned; otherwise, NULL is returned.

drmaa_get_attribute routine returns DRMAA_ERRNO_SUCCESS
on success, otherwise DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE.

drmaa_set_vector_attribute(jt, name, value, drmaa_errno_buf)
INOUT jt /* job template (opaque handle) */
IN name /* attribute name (string) */
IN values /* vector of attribute value (string vector) */

Adds ('name', 'values') pair to list of vector attributes in job template 'jt'.
Only vector attributes may be passed.

drmaa_set_vector_attribute routine returns DRMAA_ERRNO_SUCCESS
on success, otherwise
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT,
DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE,
DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES.

drmaa_get_vector_attribute(jt, name, value, drmaa_errno_buf)
IN jt /* job template (opaque handle) */
IN name /* attribute name (string) */
OUT values /* vector of attribute value (string vector) */

If 'name' is an existing vector attribute name in the job template 'jt',
then the values of 'name' are returned; otherwise, NULL is returned.

drmaa_get_vector_attribute routine returns DRMAA_ERRNO_SUCCESS
on success, otherwise DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE.

drmaa_get_attribute_names(names, drmaa_errno_buf)
OUT names /* vector of attribute name (string vector) */

Returns the set of supported attribute names whose associated value type is String.
This set will include supported DRMAA reserved attribute names and native attribute
names.

drmaa_get_vector_attribute_names(names, drmaa_errno_buf)
OUT names /* vector of attribute name (string vector) */

Returns the set of supported attribute names whose associated value type is String Vector. This set will include supported DRMAA reserved attribute names and native attribute names.

The following are reserved attribute names available in all implementations of DRMAA v1.0 (and their respective meanings). Vector attributes are marked with a 'V':

remote command to execute (string)
It is relative to the execution host.
It is evaluated on the execution host.
No binary file management is done.
The attribute name is drmaa_remote_command.

V input parameters (vector of strings)
These parameters are passed as arguments to the job.
The attribute name is drmaa_v_argv.

job state at submission (string value)
This might be useful for a rather rudimentary, but very general job dependent execution.
The states are drmaa_hold and drmaa_active:
drmaa_active means job has been queued, but it is eligible to run
drmaa_hold means job has been queued, but it is NOT eligible to run
The attribute name is drmaa_js_state.

V job environment (vector of strings)
The environment values that define the remote environment.
Each string will comply with the format <name>=<value>.
The values override the remote environment values if there is a collision.
If above is not possible, it is implementation dependent.
The attribute name is drmaa_v_env.

job working directory (string)
This attribute specifies the directory where the job is executed.
If not set, it is implementation dependent.
Evaluated relative to the execution host.

A $drmaa_hd_ph$ placeholder at the begin denotes the remaining portion of the directory_name as a relative directory name resolved relative to the job users home directory at the execution host.
The $drmaa_incr_ph$ placeholder can be used at any position within the directory_name of parametric job templates and will be substituted by the underlying DRM system with the parametric jobs' index.
The directory_name must be specified in a syntax that is common at the host where the job is executed.
If set and no placeholder is used an absolute directory specification is expected.
If set and the directory does not exist the job enters the state DRMAA_PS_FAILED.
The attribute name is drmaa_wd.

job category (string)
An opaque string specifying the way how to resolve site specific resources and/or policies.
The attribute name is drmaa_job_category.

native specification (string)
An opaque string that is passed by the end user to DRMAA to specify site specific resources and/or policies.
The attribute name is drmaa_native_specification.

e-mail address (vector of strings)
It is used to report the job completion and status.
The attribute name is drmaa_v_email.

e-mail suppression (string)
It is used to block sending e-mail by default.
Format is integer
1 block
0 unblock
The attribute name is drmaa_block_email

job start time (string)
This attribute specifies the earliest time when the job may be eligible to be run.
This is a required attribute named drmaa_start_time

The value of the attribute will be of the form
[[[[CC]YY/]MM/]DD] hh:mm[:ss] [{-|+}UU:uu]
where
CC is the first two digits of the year (century-1)
YY is the last two digits of the year
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
uu is the two digit minutes since (before) UTC
If the optional UTC-offset is not specified, the offset associated with the local timezone will be used.
If the day (DD) is not specified, the current day will be used unless the specified hour:mm:ss has already elapsed, in which case the next day will be used.
Similarly for month (MM), year (YY), and century-1 (CC).

Example:
The time: Sep 3 4:47:27 PM PDT 2002,
could be represented as: 2002/09/03 16:47:27 -07:00

job name
A job name will be comprised of alphanumeric and _ characters.
The drmaa-implementation will not provide the client with a job name longer than 1023 characters.
The drmaa-implementation may truncate any client-provided job name to an implementation-defined length which shall be at least 31 characters.
The attribute name is drmaa_job_name

input stream (string)
Specifies the jobs' standard input.
Unless set elsewhere, if not explicitly set in the job template, the job is started with an empty input stream.
If set specifies the network path of the jobs input stream file of the form
[hostname]:file_path
When the drmaa_transfer_files job template attribute is supported and contains the character 'i', the input file will be fetched by the underlying DRM system from the specified host or from the submit host if no hostname is specified.
When the drmaa_transfer_files job template attribute is not supported or does not contain the character 'i', the input file is always expected at the host where the job is executed irrespectively of a possibly hostname specified.
The $drmaa_incr_ph$ placeholder can be used at any position within the file_path of parametric job templates and will be substituted by the underlying DRM system with the parametric jobs' index.
A $drmaa_hd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the job users home directory at the host where the file is located.
A $drmaa_wd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the jobs working directory at the host where the file is located.
The file_path must be specified in a syntax that is common at the host where the file is located.
If set and the file can't be read the job enters the state DRMAA_PS_FAILED.
The attribute name is drmaa_input_path.

output stream (string)
Specifies how to direct the jobs' standard output.
If not explicitly set in the job template, the whereabouts of the jobs output stream is not defined.
If set specifies the network path of the jobs output stream file of the form
[hostname]:file_path
When the drmaa_transfer_files job template attribute is supported and contains the character 'o', the output file will be transferred by the underlying DRM system to the specified host or to the submit host if no hostname is specified.
When the drmaa_transfer_files job template attribute is not supported or does not contain the character 'o', the output file is always kept at the host where the job is executed irrespectively of a possibly hostname specified.
The $drmaa_incr_ph$ placeholder can be used at any position within the file_path of parametric job templates and will be substituted by the underlying DRM system with the parametric jobs' index.

A $drmaa_hd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the job users home directory at the host where the file is located.
A $drmaa_wd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the jobs working directory at the host where the file is located.
The file_path must be specified in a syntax that is common at the host where the file is located.
If set and the file can't be written before execution the job enters the state DRMAA_PS_FAILED.
The attribute name is drmaa_output_path.

error stream (string)
Specifies how to direct the jobs' standard error.
If not explicitly set in the job template, the whereabouts of the jobs error stream is not defined.
If set, specifies the network path of the jobs error stream file of the form [hostname]:file_path
When the drmaa_transfer_files job template attribute is supported and contains the character 'e', the output file will be transferred by the underlying DRM system to the specified host or to the submit host if no hostname is specified.
When the drmaa_transfer_files job template attribute is not supported or does not contain the character 'e', the error file is always kept at the host where the job is executed irrespectively of a possibly hostname specified.
The $drmaa_incr_ph$ placeholder can be used at any position within the file_path of parametric job templates and will be substituted by the underlying DRM system with the parametric jobs' index.
A $drmaa_hd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the job users home directory at the host where the file is located.
A $drmaa_wd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the jobs working directory at the host where the file is located.
The file_path must be specified in a syntax that is common at the host where the file is located.
If set and the file can't be written before execution the job enters the state DRMAA_PS_FAILED.
The attribute name is drmaa_error_path.

join files (string)
Specifies if the error stream should intermixed with the output stream.
If not explicitly set in the job template the attribute defaults to 'n'.
Either 'y' or 'n' can be specified.
If 'y' is specified the underlying DRM system will ignore the value of the drmaa_error_path attribute and intermix the standard error stream with the standard output stream as specified with drmaa_output_path.
The attribute name is drmaa_join_files.

The following are reserved attribute names available which are not required to be implemented by a conforming DRMAA v1.0 implementation. For attributes that are implemented, the meanings are required to be as follows:

transfer files (string)
Specifies how to transfer files between hosts.
If not explicitly set in the job template the attribute defaults to ''.
Any combination of 'e', 'i' and 'o' can be specified.
Whether the character 'e' is specified impacts the behaviour of the drmaa_error_path attribute.
Whether the character 'i' is specified impacts the behaviour of the drmaa_input_path attribute.
Whether the character 'o' is specified impacts the behaviour of the drmaa_output_path attribute.
The attribute name is drmaa_transfer_files.

absolute job termination time (string)

This attribute specifies a deadline after which the DRMS will terminate a job.
This is a reserved attribute named drmaa_deadline_time
The value of the attribute will be of the form

[[[[CC]YY/]MM/]DD] hh:mm[:ss] [{-|+}UU:uu]
where
CC is the first two digits of the year (century-1)
YY is the last two digits of the year
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
uu is the two digit minutes since (before) UTC
If the optional UTC-offset is not specified, the offset
associated with the local timezone will be used.
If the day (DD) is not specified, the current day will
be used unless the specified hour:mm:ss has already
elapsed, in which case the next day will be used.
Similarly for month (MM), year (YY), and century-1 (CC).

Example:
The time: Sep 3 4:47:27 PM PDT 2002,
could be represented as: 2002/09/03 16:47:27 -07:00

wall clock time limit (string)
This attribute specifies when the job's wall clock time limit has been exceeded. The DRMS will terminate a job which has exceeded its wall clock time limit. Note that the suspended time is also accumulated here.
This is a reserved attribute named drmaa_wct_hlimit
The value of the attribute will be of the form
[[h:]m:]s
where
h is one or more digits representing hours
m is one or more digits representing minutes
s is one or more digits representing seconds

Example:
To terminate a job after 2 hours and 30 minutes,
any of the following can be passed:
2:30:0, 1:90:0, 150:0

soft wall clock time limit (string)
This attribute specifies an estimate as to how long the job will need wall clock time to complete. Note that the suspended time is also accumulated here.
This attribute is intended to assist the scheduler.
If the time specified in insufficient, the
drmaa-implementation may impose a scheduling penalty.
This is a reserved attribute named drmaa_wct_slimit
The value of the attribute will be of the form
[[h:]m:]s
where
h is one or more digits representing hours
m is one or more digits representing minutes
s is one or more digits representing seconds

job run duration hlimit (string)
This attribute specifies how long the job may be in a running state before its limit has been exceeded, and therefore is terminated by the DRMS.
This is a reserved attribute named drmaa_run_duration_hlimit
The value of the attribute will be of the form
[[h:]m:]s
where
h is one or more digits representing hours
m is one or more digits representing minutes
s is one or more digits representing seconds

job run duration slimit (string)
This attribute specifies an estimate as to how long the job will need to remain in a running state to complete.

This attribute is intended to assist the scheduler. If the time specified in insufficient, the drmaa-implementation may impose a scheduling penalty.
This is a reserved attribute named drmaa_run_duration_slimit
The value of the attribute will be of the form
[[h:]m:]s
where
h is one or more digits representing hours
m is one or more digits representing minutes
s is one or more digits representing seconds

NOTE:
There are two ways of using reserved attributes, programmatically or by passing information directly to DRMAA implementation at run time via "native specification" or "job categories."

One method of using reserved attributes programmatically is to get a list of DRMAA attributes and then use the attributes that are available. The other way is to try setting the attribute and checking the routine return value.

```
/* ----------  Job Submission Routines ---------- */

drmaa_run_job(job_id, jt, drmaa_errno_buf)
   OUT job_id      /* job identifier (string) */
   IN  jt          /* job template (opaque handle) */

  Submit a job with attributes defined in the job template 'jt'.
  The job identifier 'job_id' is a printable, NULL terminated string,
  identical to that returned by the underlying DRM system.

drmaa_run_job routine returns DRMAA_ERRNO_SUCCESS on success,
otherwise DRMAA_ERRNO_TRY_LATER,
DRMAA_ERRNO_DENIED_BY_DRM,
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, or
DRMAA_ERRNO_AUTH_FAILURE.

drmaa_run_bulk_jobs(job_ids, jt, start, end, incr, drmaa_errno_buf)
   OUT job_ids     /* job identifiers (array of strings) */
   IN  jt          /* job template (opaque handle) */
   IN  start       /* beginning index ( unsigned integer?)*/
   IN  end         /* ending index ( unsigned integer?) */
   IN  incr        /* loop increment (integer)*/

  Submit a set of parametric jobs, dependent on the implied loop index, each
  with attributes defined in the job template 'jt'.
  The job identifiers 'job_ids' are all printable,
  NULL terminated strings, identical to those returned by the underlying
  DRM system.  Nonnegative loop bounds are mandated to avoid file names
  that start with minus sign like command line options.

  The special index placeholder is a DRMAA defined string
     drmaa_incr_ph /* == $incr_pl$ */
  that is used to construct parametric job templates.

  For example:
     drmaa_set_attribute(pjt, "stderr", drmaa_incr_ph + ".err" ); /*
  C++/java string syntax used */

drmaa_run_bulk_jobs routine returns DRMAA_ERRNO_SUCCESS on success,
otherwise DRMAA_ERRNO_TRY_LATER,
DRMAA_ERRNO_DENIED_BY_DRM,
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, or
DRMAA_ERRNO_AUTH_FAILURE.



/* ---------- Job Control Routines ---------- */


drmaa_control(job_id, action, drmaa_errno_buf)
   IN job_id       /* job identifier (string) */
   IN action       /* control action (const) */
```

Start, stop, restart, or kill the job identified by 'job_id'.
If 'job_id' is DRMAA_JOB_IDS_SESSION_ALL, then this routine
acts on all jobs *submitted* during this DRMAA session.
The legal values for 'action' and their meanings are:
DRMAA_CONTROL_SUSPEND: stop the job,
DRMAA_CONTROL_RESUME: (re)start the job,
DRMAA_CONTROL_HOLD: put the job on-hold,
DRMAA_CONTROL_RELEASE: release the hold on the job, and
DRMAA_CONTROL_TERMINATE: kill the job.

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

drmaa_control routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE,
DRMAA_ERRNO_AUTH_FAILURE,
DRMAA_ERRNO_RESUME_INCONSISTENT_STATE
DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE
DRMAA_ERRNO_HOLD_INCONSISTENT_STATE
DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE
DRMAA_ERRNO_INVALID_JOB.

drmaa_synchronize(job_ids, timeout, dispose, drmaa_errno_buf)
IN job_ids /* job identifiers (array of strings) */
IN timeout /* how long we block in this call (signed long) */
IN dispose /* dispose reaping information (boolean)*/

Wait until all jobs specified by 'job_ids' have finished
execution. If 'job_ids' is DRMAA_JOB_IDS_SESSION_ALL, then this routine
waits for all jobs *submitted* during this DRMAA session. To prevent
blocking indefinitely in this call, the caller could use timeout specifying
after how many seconds to time out in this call.
If the call exits before timeout, all the jobs have been waited on or there was an interrupt.

If the invocation exits on timeout, the return code is DRMAA_ERRNO_EXIT_TIMEOUT.
The caller should check system time before and after this call
in order to check how much time has passed.

Dispose parameter specifies how to treat reaping information:
True=1 "fake reap", i.e. dispose of the rusage data
False=0 do not reap

drmaa_ synchronize routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE,
DRMAA_ERRNO_AUTH_FAILURE,
DRMAA_ERRNO_EXIT_TIMEOUT, or
DRMAA_ERRNO_INVALID_JOB.

drmaa_wait(job_id, stat, timeout, rusage, drmaa_errno_buf)
IN job_id /* job identifier (string) or DRMAA_JOB_IDS_SESSION_ANY (string) */
OUT job_id /* job identifier of ended job (string) or NULL */
OUT stat /* status code of job (integer) */
IN timeout / * how long we block in this call (signed long) */
OUT rusage /* resource usage */

This routine waits for a job with job_id to fail or finish execution. If the special string
DRMAA_JOB_IDS_SESSION_ANY is provided as the job_id, 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 DRMAA_TIMEOUT_WAIT_FOREVER (-1) can be specified to wait
indefinitely for a result. The value DRMAA_TIMEOUT_NO_WAIT (0) 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, the return code is DRMAA_ERRNO_EXIT_TIMEOUT.
The caller should check system time before and after this call

in order to check how much time has passed.

The routine reaps jobs on a successful call, so any subsequent calls to drmaa_wait should fail returning an error DRMAA_ERRNO_INVALID_JOB meaning that the job has been already reaped. This error is the same as if the job was unknown. Failing due to an elapsed timeout has an effect that it is possible to issue drmaa_wait multiple times for the same job_id. When successful, the rusage information will be provided as an array of strings, where each string complies with the format <name>=<value>.
The string portion <value> contains the amount of resources consumed by the job and is opaque.

drmaa_ wait routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE,
DRMAA_ERRNO_AUTH_FAILURE,
DRMAA_ERRNO_EXIT_TIMEOUT, or
DRMAA_ERRNO_INVALID_JOB.

drmaa_wifexited(OUT exited, IN stat, drmaa_errno_buf)
Can optionally evaluate into 'exited' a zero value if status was returned for a job that terminated not normally. A non-zero 'exited' value indicates more detailed diagnosis can be provided by means of drmaa_wifsignaled(), drmaa_wtermsig() and drmaa_wcoredump().

drmaa_wexitstatus(OUT exit_code, IN stat, drmaa_errno_buf)
If the OUT parameter 'exited' of drmaa_wifexited() is non-zero, this function evaluates into 'exit_code' 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.

drmaa_wifsignaled(OUT signaled, IN stat, drmaa_errno_buf)
Evaluates into 'signaled' a non-zero value if status was returned for a job that terminated due to the receipt of a signal.

drmaa_wtermsig(OUT signal, IN stat, drmaa_errno_buf)
If the OUT parameter 'signaled' of drmaa_wifsignaled(stat) is non-zero, this function evaluates into signal a string representation of the signal that caused the termination of the job. For signals declared by POSIX, the symbolic names are returned (e.g., “SIGABRT”, “SIGALRM”).
For signals not declared by POSIX, any other string may be returned.

drmaa_wcoredump(OUT core_dumped, IN stat, drmaa_errno_buf)
If the OUT parameter 'signaled' of drmaa_wifsignaled(stat) is non-zero, this function evaluates into 'core_dumped' a non-zero value if a core image of the terminated job was created.

drmaa_wifaborted(OUT aborted, IN stat, drmaa_errno_buf)
Evaluates into 'aborted' a non-zero value if 'stat' was returned for a job that ended before entering the running state.

The 'stat' drmaa_wait parameter is used in a series of functions, defined above, for providing more detailed information about job termination if available. An analogous set of macros is defined in POSIX for analyzing wait3(2) OUT parameter 'stat'. The misleading upper-case function names reminding to macros are changed to lower-case names.

drmaa_job_ps(IN job_id, OUT remote_ps, drmaa_errno_buf);
IN job_id /* job identifier (string) */
OUT remote_ps /* program status (constant) */

Get the program status of the job identified by 'job_id'.
The possible values returned in 'remote_ps' and their meanings are:
DRMAA_PS_UNDETERMINED = 00H : process status cannot be determined,

DRMAA_PS_QUEUED_ACTIVE = 10H : job is queued and active,
DRMAA_PS_SYSTEM_ON_HOLD = 11H : job is queued and in system hold,
DRMAA_PS_USER_ON_HOLD = 12H : job is queued and in user hold,
DRMAA_PS_USER_SYSTEM_ON_HOLD = 13H : job is queued and in user and system hold,

DRMAA_PS_RUNNING = 20H : job is running,
DRMAA_PS_SYSTEM_SUSPENDED = 21H : job is system suspended,
DRMAA_PS_USER_SUSPENDED = 22H : job is user suspended,
DRMAA_PS_DONE = 30H : job finished normally, and
DRMAA_PS_FAILED = 40H : job finished, but failed.

DRMAA should always get the status of job_id from DRM system,
unless the previous status has been DRMAA_PS_FAILED or DRMAA_PS_DONE and the status has been successfully cached. Terminated jobs get DRMAA_PS_FAILED status.

drmaa_ synchronize routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE,
DRMAA_ERRNO_AUTH_FAILURE, or
DRMAA_ERRNO_INVALID_JOB.

/* ---------- Auxiliary Routines ---------- */

error_string drmaa_strerror (errno, drmaa_errno_buf);
IN errno /* Errno number (integer) */
RETURNS /* Readable text version of errno (constant string) */

Get the error message text associated with the errno number.

contact drmaa_get_contact();
OUT contact /* Current contact information for DRM system (string) */

3.3 List of DRMAA Errors

---------- these are relevant to all sections ----------

DRMAA_ERRNO_SUCCESS
Routine returned normally with success.

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

DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
Could not contact DRM system for this request.

DRMAA_ERRNO_AUTH_FAILURE
The specified request is not processed successfully due to authorization failure.

DRMAA_ERRNO_INVALID_ARGUMENT
The input value for an argument is invalid.

---------- init and exit specific ----------

DRMAA_ERRNO_INVALID_CONTACT_STRING
Initialization failed due to invalid contact string.

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

DRMAA_ERRNO_DRMS_INIT_FAILED
Initialization failed due to failure to init DRM system.

DRMAA_ERRNO_ALREADY_ACTIVE_SESSION
Initialization failed due to existing DRMAA session.

DRMAA_ERRNO_DRMS_EXIT_ERROR
DRM system disengagement failed.

---------- job attributes specific ----------
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT
The format for the job attribute value is invalid.

DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE
The value for the job attribute is invalid.

DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES
The value of this attribute is conflicting with a previously set attributes.

---------- job submission specific ----------

DRMAA_ERRNO_TRY_LATER
Could not pass job now to DRM system. A retry may succeed however (saturation).

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

---------- job control specific ----------

DRMAA_ERRNO_INVALID_JOB
The job specified by the 'jobid' does not exist.

DRMAA_ERRNO_RESUME_INCONSISTENT_STATE
The job has not been suspended. The RESUME request will not be processed.

DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE
The job has not been running, and it cannot be suspended.

DRMAA_ERRNO_HOLD_INCONSISTENT_STATE
The job cannot be moved to a HOLD state.

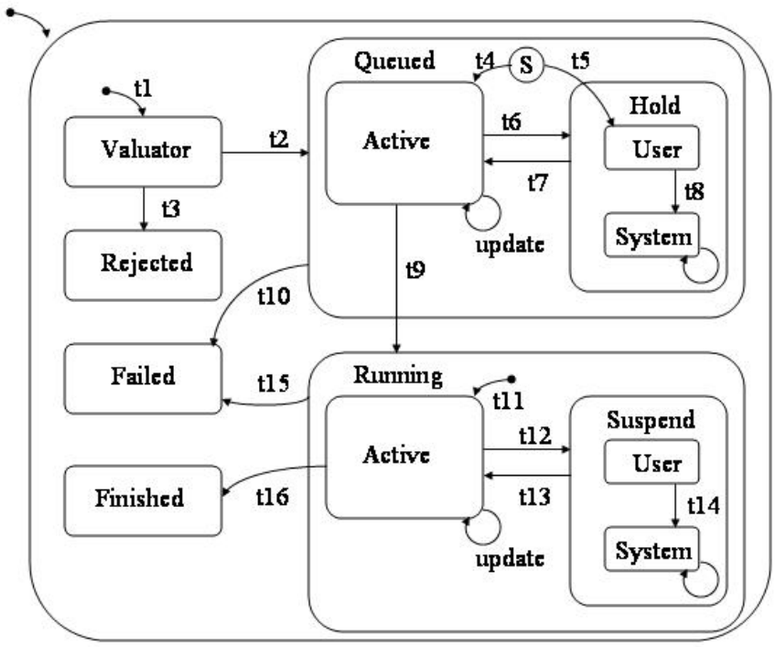
DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE
The job is not in a HOLD state.

DRMAA_ERRNO_EXIT_TIMEOUT
We have encountered a time-out condition for drmaa_synchronize or drmaa_wait.

3.4 DRMAA job state transition diagram

DRMAA job state transition diagram that matches the specification in section 3.2 is provided in Harel notation here.

DRMAA Job State Transition



4. 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.

Author Information

Roger Brobst
rbrobst@cadence.com
Cadence Design Systems, Inc
555 River Oaks Parkway
San Jose, CA 95134

Waiman Chan
waimanc@us.ibm.com
International Business Machines Corporation
2455 South Road
Poughkeepsie, New York 12601

Fritz Ferstl
fritz.ferstl@sun.com
Sun Microsystems GmbH
Dr.-Leo-Ritter-Str. 7
D-93049 Regensburg
Germany

Jeffrey T. Gardiner
gardiner@imaging.robarts.ca
Robarts Research Institute
PO Box 5015, 100 Perth Drive
London ON, N6A 5K8
Canada

Andreas Haas
andreas.haas@sun.com
Sun Microsystems GmbH
Dr.-Leo-Ritter-Str. 7
D-93049 Regensburg

Germany

Bill Nitzberg
bill@computer.org
Veridian, PBS Products
2672 Bayshore Pkwy, Suite 810
Mountain View, CA 94043

Hrabri L. Rajic
hrabri.rajic@intel.com
Intel Americas Inc.
1906 Fox Drive
Champaign, IL 61820

John Tollefsrud
j.t@sun.com
Sun Microsystems
200 Jefferson Drive UMPK29-302
Menlo Park, CA 94025

Intellectual Property Statement

Full Copyright Notice