NORDUGRID

NORDUGRID-TECH-14

5/8/2008

The ARC Computational Job Management Component - A-Rex

Description and Administrator's Manual

A.Konstantinov*

*aleks@fys.uio.no

Contents

1 Introduction

A-Rex is one of ARC middleware componets implementing functions of so called *Computing Element* (CE).

The aim of the A-Rex is to take care of job pre- and post-processing. It provides functionality to stage-in files containing input data and program modules from wide range of sources and transfer or store output results.

The A-Rex implements Web Service (WS) interface which provides a way to submit and control computational tasks (jobs) to be executed by the A-Rex and undelying Local Resource Management System.

You should use this document for advanced configuration purposes and understanding of internals of the aforementioned tools. For general instalaltion and configuration refer to other documents avaialble at `http://www.nordugrid.org/papers.html`.

2 Main concepts

A job is a set of input files (which may or may not include executables), a main executable and a set of output files. The process of gathering input files, executing a job, and transferring/storing output files is called a *session*.

Each job gets a directory on the CE called the *session directory* (SD). Input files are gathered in the SD. The job is supposed to produce new data files in the SD as well. The A-Rex does not guarantee the availability of any other places accessible by the job other than SD (unless such a place is part of a requested Runtime Environement). The SD is also the only place which is controlled by the A-Rex. It is accessible by the user from outside through HTTP(S) protocol. Any file created outside the SD is not controlled by the A-Rex. Any exchange of data between client and A-Rex (including also program modules) is done through HTTP(S) protocol . A URL for accessing input/output files is obtained through the WS Local Information Description Interface (LIDI) of A-Rex.

Each job gets an identifier (*jobid*). This is a handle which identifies the job in the A-Rex and the Information Interface.

Jobs are initiated and controlled through WS interface. Complete job descriptions (JD) are passed to the A-Rex through WS in JSDL [1] coded description . Input data files and job executables are transferred separately through the same interface, as described in the next section.

3 Input/output data

One of the most important tasks of the A-Rex is to take care of processing input and output data (files) of the job. Input files are gathered in the SD. There are two ways to put a file into the SD:

- Download is initiated by the A-Rex – This is the case for files defined in the JD (with name and source). The A-Rex alone is responsible to make sure that all required files will be available in the SD.
  The supported protocols for sources at the moment are: GridFTP, FTP, HTTP, HTTPS (HTTP over SSLv3) and HTTPg (HTTP over GSI). Also some nonstandard sources are supported. Those are described below. The A-Rex fully relies on HED framework [?] for data transfering capabilities. Hence actual set of supported protocols depends on installed Data Management Components of the HED.

- Upload is initiated by the user directly or through the User Interface (UI). Because the SD becomes available immediately at the time of submission of JD, UI can (and should) use that to upload data files which are not otherwise accessible by the A-Rex. Examples of such files are the main executable of the job, the job's input files, etc. These files can (and should) also be specified in the JD.

There is no other reliable way for a job to obtain input data on the CE based on the A-Rex . Access to AFS, NFS, FTP, HTTP and any other remote data transport during execution of a job is not guaranteed (at least not by A-Rex).

Jobs should store output files in their SD. Like input files, output files belong into two groups:

- Files which are supposed to be moved to a *Storage Element* (SE) and optionally registered in some *Indexing Service* like the Globus *Replica Location Service* (RLS) – The A-Rex takes care of these files. They have to be specified in the JD. If the job fails during any stage of processing, no attept is made to transfer those files to their final destination, unless the option *preserve=yes* is specified in their URLs.
- Files which are supposed to be fetched by the user – The user has to use a tool like the UI to obtain these files. They **must** also be specified in the JD.

4 Job flow

From the point of view of the A-Rex a job passes through various states. Picture 1 presents a diagram of the possible states of a job. A user can examine the state of a job by querying the dedicated Local Information Description Interface

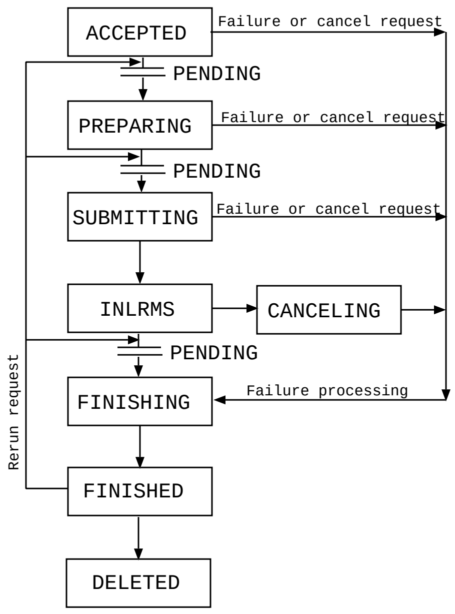

Figure 1: Job states

of A-Rex using the UI or any other suitable tool

or through query method of WS interface.

Configuration can put limits on the amout of simultaneous jobs in some states. If such a limit is reached, a job ready to enter into the state in question will stay in it's current state waiting for a free slot. This situation is presented by prepending the prefix **PENDING:** to the current state name in the job's status mark.

Below is the description of all actions taken by the A-Rex at every state:

- **Accepted** - In this state the job has been submitted to a CE but is not processed yet. The A-Rex will analyze the JD and move to the next stage. If the JD can not be processed the job will be canceled and moved to the state **Finishing**.

- **Preparing** - The input data is being gathered in the SD (stage-in). The A-Rex is downloading the files specified in the JD and is waiting for files which are supposed to be uploaded by the UI. If all files are successfully gathered the job moves to the next state. If **any** file can't be downloaded or it takes the UI too long to upload a file, the job moves to **Finishing** state. It is possible to put a limit on the number of simultaneous **Preparing** jobs. If this limit is exceeded, jobs ready to enter the **Preparing** state will stay in the **Accepted** state, but prefixed with the PENDING: mark. Exceptions are jobs which have no files to be downloaded. These are processed out of limits.
- **Submitting** - The job is being passed for execution to the *Local Resource Management System* (LRMS). The correspoding backends for many LRMSs are provided with the default installation. If the local job submission is successful the job moves to the **InLRMS** state. Otherwise it moves to **Finishing**. It is possible to limit the agregate number of jobs in **Submitting** and **InLRMS** states.
- **InLRMS** - The job is queued or being executed in the LRMS. The A-Rex takes no actions except waiting until the job finishes.
- **Cancelling** - Necessary action to cancel the job in the LRMS is being taken.
- **Finishing** - The output data is being processed (stage-out). Specified data files are moved to the specified SEs and are optionally registered at an Indexing Service. The user can download data files from the SD by using the UI or other adequate tool. All the files not specified as output files are removed from the SD at very beginning of this state. It is possible to limit number of simultaneous jobs in this state.
- **Finished** - No more processing is performed by the A-Rex. The user can continue to download data files from the SD. The SD is kept available for some time (default is 1 week). After that the job is moved to the state **Deleted**. The 'deletion' time can be obtained by querying the Informatrion Interface of the A-Rex . If a job was moved to **Finished** because of failure, it may be restarted on request of a client. When restarted, a job is moved to the state previous to the one in which it failed and is assigned mark PENDING. This is needed in order to not break the configuration limits. Exception is a job failed in **InLRMS** state and lacking input files specified in JD. Such a job is treated like failed in **Preparing** state.
- **Deleted** - The job is moved to this state if the user have not requested job to be cleaned before the SD's lifetime expires. Only minimal subset of information about such job is kept. The SD is not available anymore.

In case of a failure special processing is applied to output files. All specified output files are treated as **downloadable by the user**. No files will be moved to their destination SE.

5 URLs

In full installation the A-Rex and it's components support the following data transfer protocols and corresponding URLs: *ftp, gsiftp, http, httpg, https, se, rc* and *rls.* For more information please see "The Hosting Environment of the Advanced Resource Connector middleware " document [**?**].

6 Internals

6.1 Internal Files of the A-Rex

For each local UNIX user listed in the A-Rex configuration - including a generic one which covers all local user identities - a *control directory* exists. In this directory the A-Rex stores information about jobs belonging to that user.

Multiple users can share the same *control directory*. To make it easier to recover in case of failure, the A-Rex stores most information in files rather than in memory. All files belonging to the same job have names starting with **job.ID.,** where ID is the job identifier.

The files in the control directory and their formats are described below:

- *job.ID.status* - current state of the job. This is a plain text file containing a single word representing the current state of the job. Possible values are :
  - ACCEPTED
  - PREPARING
  - SUBMITTING
  - INLRMS
  - FINISHING
  - FINISHED
  - CANCELING
  - DELETED

See section 4 for a description of the various states. Additionally each value can be prepended the prefix "PENDING:" (like PENDING:ACCEPTED, see section 4). This is used to show that a job is *ready* to be moved to the next state but it has to stay in it's current state *only* because otherwise some limits set in the configuration would be exceeded.

- *job.ID.description* - contains the description of the job (JD).
- *job.ID.local* - information about the job used by the A-Rex. It consists of lines of format *"name = value"* . Not all of them are always available. The following names are defined:
  - *subject* - user certificate's subject, also known as the distinguished name (DN)
  - *starttime* - GMT time when the job was accepted represented in the Generalized Time format of LDAP
  - *lifetime* - time period to preserve the SD after the job has finished in seconds
  - *cleanuptime* - GMT time when the job should be removed from the cluster and it's SD deleted in Generalized Time format
  - *notify* - email addresses and flags to send mail to about the job specified status changes
  - *processtime* - GMT time when to start processing the job in Generalized Time format
  - *exectime* - GMT time when to start job execution in Generalized Time format
  - *expiretime* - GMT time when the credentials delegated to the job expire in Generalized Time format
  - *rerun* - number of retries left to rerun the job
  - *jobname* - name of the job as supplied by the user
  - *lrms* - name of the LRMS backend to be used for local submission
  - *queue* - name of the queue to run the job at
  - *localid* - job id in LRMS (appears only after the job has reached state **InLRMS**)
  - *args* - executable name followed by a list of command-line arguments
  - *downloads* - number of files to download into the SD before execution

  - *uploads* - number of files to upload from the SD after execution
  - *gmlog* - directory name which holds files containing information about the job when accessed through GridFTP interface
  - *clientname* - name (as provided by the user interface) and ip address:port of the submitting client machine
  - *clientsoftware* - version of software used to submit the job
  - *sessiondir* - the job's SD
  - *failedstate* - state in which job failed (available only if it is possible to restart the job)
  - *jobreport* - URL of a user requested *logger service*. The GM will also send job records to this service in addition to the default logger service configured in the configuration (This feature does not work in current implementation).

This file is filled partially during job submission and fully when the job moves from the **Accepted** to the **Preparing** state.

- *job.ID.input* - list of input files. Each line contains 2 values separated by a space. First value contains name of the file relative to the SD. Second value is an URL or a file description. Example:

  input.dat gsiftp://grid.domain.org/dir/input_12378.dat

  url - ordinary URL for gsiftp, ftp, http, https or httpg protocols with the addition of '**replica catalog url**' (RC URL) and '**replica location service url**' (RLS URL).
  Each URL can contain additional options.

  file description - [size][.checksum].

  size - size of the file in bytes.

  checksum - checksum of the file identical to the one produced by ***cksum*** (1).

  Both size and checksum can be left out. Special kind of file description *.* is used to specify files which are **not** required to exist.

This file is used by the '***downloader***' utility. Files with 'url' will be downloaded to the SD and files with 'file description' will simply be checked to exist. Each time a new **valid** file appears in the SD it is removed from the list and *job.ID.input* is updated. Any external tool can thus track the process of collecting input files by checking *job.ID.input.*

- *job.ID.output* - list of output files. Each line contains 1 or 2 values separated by a space. First value is the name of the file relative to the SD. The second value, if present, is a URL. Supported URLs are the same as those supported by job.ID.input.

This file is used by the '***uploader***' utility. Files with *url* will be uploaded to SE and remaining files will be left in the SD. Each time a file is uploaded it is removed from the list and *job.ID.output* is updated. Files not mentioned as output files are removed from the SD at the the beginning of the **Finishing** state.

- *job.ID.failed* - the existence of this file marks the failure of the job. It can also contain one or more lines of text describing the reason of failure. Failure includes the return code different from zero of the job itself.

- *job.ID.errors* - this file contains the output produced by external utilities like ***downloader***, ***uploader***, script for job submission to LRMS, etc on their stderr handle. Those are not necessarily errors, but can be just useful information about actions taken during the job processing. In case of problem include content of that file while asking for help.

- *job.ID.diag* - information about resources used during execution of job and other information suitable for diagnostics and statistics. It's format is similar to that of *job.ID.local*. The following names are at least defined:
  - *nodename* - name of computing node which was used to execute job,
  - *runtimeenvironments* - used runtime environments separated by ';',
  - *exitcode* - numerical exit code of job,
  - *frontend_distribution* - name and version of operating system distribution on frontend computer,
  - *frontend_system* - name of operating on frontend computer,
  - *frontend_subject* - subject (DN) of certificate representing frontend computer,
  - *frontend_ca* - subject (DN) of issuer of certificate representing frontend computer,

  and other information provided by GNU *time* utility. Note that some implementation of *time* insert unrequested information in their output. Hence some lines can have broken format.
- *job.ID.proxy* -delegated X509 credentials .
- *job.ID.proxy.tmp* - temporary X509 credentials with different UNIX ownership used by processes run with effective *user id* different form job owner's *id*.

There are other files with names like job.ID.* which are created and used by different parts of the A-Rex. Their presence in the *control directory* can not be guaranteed and can change depending on changes in the A-Rex code.

6.2

6.3 Web Service Interface

A-REX Web Service Interface provides means to submit a description of a computational job to a computing resource, to stage-in additional data, to monitor and control processing of jobs, and obtain data produced during the execution of a job. The WS Interface is built and deployed inside Hosting Envirnoment Daemon (HED) infrastructure[**?**].

6.3.1 Basic Execution Service Interface

The job submission and control interface is based on document produced by the OGF OGSA Basic Execution Services (BES) Working Group[2]. The exchange of SOAP messages is performed over the HTTP(S) protocol. The BES interface is represented by two port-types - BES-Management and BES-Factory. The former is made to control the A-REX service itself and thus defines operations to start and stop the functionality of the BES service. The A-Rex does not implement remote control of service functionality. Hence the BES-Management port-type is not functional. The BES-Factory port-type provides operations to submit new jobs (to create an activity in terms of BES) and to monitor its state. It also has an ability to provide information about the service. A-Rex fully implements the functionality of this port-type.

For job descriptions A-Rex accepts the Job Submission Description Language (JSDL)[1] documents as defined by the OGF Job Submission Description Language Working Group. Supported elements and extensions are described below.

Table 1: Job states definitions and mappings

Applicable BES State	ARC Sub-state	Description
Pending	Accepting	Job is in the process of being submitted
	Accepted	Job was submitted
Running	Preparing	Stage-in process is going on
	Prepared	Stage-in process has finished
	Submitting	Communication with local batch system is in process
	Executing	Job is being executed in local batch system
	Killing	Communication with local batch system to terminate execution is in process
	Executed	Job execution in local batch system has finished
	Finishing	Stage-out process is going on
Cancelled	Failed	There was a failure during execution
Failed		
All	Pending	Job is prevented from going to the next state due to some internal limits; this sub-state appears in parallel with other sub-states
All	Held	Job processing is suspended on client request; this sub-state appears in parallel with other sub-states

6.3.2 Extensions to OGSA BES interface

A-R introduces a new operation in addition to those provided by BES. It does that by defining its own port-type with the single operation *ChangeActivityStatus* (see Appendix D). This operation provides a way to request simple transfers between states of jobs and corresponding actions.

- *ChangeActivityStatus*
  - Input
    - *ActivityStatusType OldStatus*: Description of the state the job is supposed to be in during execution of this request. If the current state of the job is different from the one having been given, the operation is aborted and a fault is returned. This parameter is optional.
    - *ActivityStatusType NewStatus*: Description of the state the job is to be put into.
  - Output
    - *ActivityStatusType NewStatus*: Description of the current state of the job.
  - Fault(s)
    - *NotAuthorizedFault*: Indicates that the client is not allowed to do this operation.
    - *InvalidActivityIdentifierFault*: There is no such job/activity.
    - *CantApplyOperationToCurrentStateFault*: The requested transition is not possible.

On result of this command, the job should be put into the requested state. If such a procedure cannot be performed immediately then the corresponding sequence is initiated and fault OperationWillBeAppliedEventuallyFault will be returned.

Since BES allows implementators to extend their initial activity states with additional sub-states, A-Rex defines a set of sub-states of activity processing in addition to those defined by the BES, as listed in Table 1.

6.3.3 Delegation Interface

The A-Rex also supportd the Delegation Interface (see Appendix E). This is a common purpose interface to be used by ARC services which accepts delegated credentials from clients. The Delegation Interface implements two operations: initiatialization of credentials delegation (DelegateCredentialsInit) and update/renewal of credentials (UpdateCredentials).

- *DelegateCredentialsInit* operation - this operation performs the first half of the credentials delegation sequence.
  - Input
    - None. On this request the service generates a pair of *public* and private keys. The public key is then sent to the client in response.
  - Output(s)
    - *TokenRequestType TokenRequest*: Contains the public key generated by the service as a Value element. It also provides an identifier in the Id element which should be used to refer to the corresponding private key.
  - Fault(s)
    - *UnsupportedFault*: Indicates that the service does not support this operation despite supporting the port-type.
    - *ProcessingFault*: Internal problems during generation of the token.
- *UpdateCredentials* operation - this operation makes it possible to update the content of delegated credentials (like in the case of credentials being renewed) unrelated to other operations of the service.
  - Input
    - *DelegatedTokenType DelegatedToken*: Contains an X509 proxy certificate based on the public key from the DelegateCredentialsInit signed by the user's proxy certificate. Also includes the Id element which identifies the private key stored at the service side associated with these credentials. The reference element refers to the object to which these credentials should be applied in a way specific to the service. The same element must also be used for delegating credentials as part of other operations on service.
  - Output(s)
    - None.
  - Fault(s)
    - *UnsupportedFault*: Indicates that service does not support this operation despite supporting the port-type.
    - *ProcessingFault*: Internal problems during generation of the token.

Additionally, A-Rex Web Service Interface allows delegation to be performed as part of the *CreateActivity* operation of the BES-Factory port-type. For this it accepts the element *DelegatedCredentials* inside the *CreateActivity* element. The *Id* element of *DelegatedCredentials* must contain an identifier obtained in response to the previous *DelegateCredentialsInit* operation.

6.3.4 Local Information Description Interface

The A-REX implements the Local Information Description Interface (LIDI) interface common for all ARC services. This interface is based on OASIS Web Services Resource Properties specification[3]. Information about resources and maintained activities/jobs are represented in a *WS-Resource Properties* informational XML document. The document type is defined in the A-Rex WSDL as a *ResourceInformationDocumentType*. It contains the following elements/resources:

BESFactory - collection of BES Factory attributes as defined in the BES specifications.

Glue2Resource - description of a computation resource that uses Glue2 schema. This one is going to be specified after the Glue2 XML bindings will be available.

Activities - list of maintained activities. Each activity contains an identifier (*ActivityIdentifier*), BES (*ActivityDocument*) and Glue2 (*Glue2Job*) descriptions of the activity.

All information can be accessed either through requests on particular resources or through XPath queries using WS-Resource Properties operations.

6.3.5 Supported JSDL elements

A-REX supports the following elements from the JSDL version 1.0 specification[1] including POSIX Applications extension:

JobName - name of the job as assigned by the user.

Executable (POSIX) - name of the executable file.

Argument (POSIX) - arguments the executable will be launched with.

DataStaging

Filename	- name of the data file on the executing node.
Source	- source where the file will be taken from before execution.
Target	- destination the file will be delivered to after execution.

Input (POSIX) - file to be used as standard input for the executable.

Output (POSIX) - file to be used as standard output for the executable.

Error (POSIX) - file to be used as standard error for the executable.

MemoryLimit (POSIX) - amount of physical memory needed for execution.

TotalPhysicalMemory - same as MemoryLimit.

IndividualPhysicalMemory - same as MemoryLimit.

CPUTimeLimit (POSIX) - maximal amount of CPU time needed for execution.

TotalCPUTime - same as CPUTimeLimit.

IndividualCPUTime - same as CPUTimeLimit.

WallTimeLimit (POSIX) - amount of clock time needed for execution.

TotalCPUCount - number of CPUs needed for execution.

IndividualCPUCount - same as TotalCPUCount.

6.3.6 ARC-specific JSDL Extensions

A-REX accepts JSDL documents having the following additional elements (see Appendix F):

IsExecutable - marks file to become executable after being delivered to the computing resource.

RunTimeEnvironment - specifies the name of the Runtime Environment needed for job execution.

Middleware - request for specific middleware on the computing resource frontend.

RemoteLogging - destination for the usage record report of the executed job.

LocalLogging - name for the virtual directory available through job interface and containing various debug information about job execution.

AccessControl - ACL expression which describes the identities of those clients who are allowed to perform operations on this job.

Notify - Email destination for notification of job state changes.

SessionLifeTime - duration for the directory containing job-related files to exist after the job finished executing.

JoinOutputs - specifies if standard output and standard error channels must be merged.

Reruns - defines how many times a job is allowed to rerun in case of failure.

CredentialServer - URL of MyProxy service which may be used for renewing the expired delegated job credentials.

CandidateTarget - specifies host name and queue of a computing resource.

7 Cache

The GM can cache input files. Caching is enabled if corresponding command is present in configuration file. The GM does not cache files marked as executable in job. Caching can also be explicitly turned off by user for each file by using *cache=no* option in URL (for URL options read “Protocols, Uniform Resource Locators (URL) and extensions supported in ARC” [4]). The disc space occupied by cache is controlled by removing unused files. For more information look in section 9.1.

7.1 Structure

Cache directory contains plain files. Those are

- *list* - stores names of the files (8 digit numbers) and corresponding URLs delimited by blank space. Each pair is delimited by some amount of \0 codes. Also creation and expiration times are stored if available
- *old* - stores URLs which have been removed from cache. Records are delimited by some amount of \0 codes and are meant to be removed by some external routine.

- *new* - stores URLs which have been added to cache. Records are delimited by some amount of \0 codes and are removed when corresponding files are removed from cache. They can also be handled by some external routines. Every time record is added to *old* it is removed from *new*.
- *statistics* - consists of strings containing *name=value* pairs. Following names are defined:
  - *hardsize* -size of file system for storing cached data
  - *hardfree* - amount of disc space available on that file system
  - *softsize* - if cache exceeds this size files are started being removed
  - *softfree* - space left till softsize (can be negative)
  - *claimed* - space used by files claimed by running jobs
  - *unclaimed* - space used by files not being currently used by any job
- *########.info* - stores state of file (*########* stands for 8 digits). State is represented by one character:
  c - just created, content is empty.
  f - failed to download (treated same as 'c').
  r - ready to be used, content is valid.
  d - being downloaded. 'd' is followed by identifier of application/job downloading that file. During content's download this file has write lock set.
- *########.claim* - stores list of identifiers of applications/jobs using this file. Identifiers are stored one per line.
- *########* - files storing content of corresponding URL. These can be stored in separate directory.

Files *list, old , new* and *########.info* has to be stored on filesystem which has support for files' locking.

7.2 How it works

If a job requests an input file which is subject for cacheing, it is stored in the cache directory instead of the SD. The file is made available to the job by either soft-linking it in the SD or copying it to the SD. The latter option is more secure and hence advised.

Before downloading a file the GM tries to determine it's size and then tries to preallocate space in the cache directory by writing a file of the same size. If this fails (possibly because the file system has no more space), it tries to the remove oldest cached files which are not being used by any job. This means that **the hard limit of cache size is the space available in the file-system**. In case cache gets full and it is impossible to free enough space, the download fails and then is retried without using cache.

Before giving access to an already cached file, the GM contacts the original file source to check if the user has suficient access rights to the file. Not all protocols make this check possible.

Also, file creation or validity times are checked to make sure the cached file is fresh . If it is impossible to obtain creation and invalidation times , the file is invalidated 24 hours after it was downloaded.

The GM checks the cache periodically. If the space used by cache exceeds the high water-mark given in the configuration file (*softsize*), it tries to remove the oldest unused files until cache size drops below the low water-mark. This sets the soft limit of cache size.

There are two kinds of caches supported. Files in *private* cache are owned by the Unix user to which a grid user is mapped. Those files are readable only by that particular Unix user. Another kind of cache is *shared*. Files in the shared cache are owned by the Unix user who started GM and are readable by everyone.

8 Files and directories

8.1 Modules

The GM consists of several separate executable modules. These are:

- *grid-manager* - The main module. It is responsible for processing jobs, moving them through states and running other modules.
- *downloader* - This is a module responsible for gathering input files in the SD. It processes the *job.ID.input* file and updates it.
- *uploader* - This module is responsible for delivering output files to the specified SEs and registration at an Indexding Service (like RLS) as needed. It processes and updates the *job.ID.output* file.
- *cache-register* - Utility to register cached data into an Indexing Service . It reads and modifies cache informational files *old* and *new* (as described in section 7). Configuration is read directly from the GM's configuration file (see section 9.1). It is run by the GM every 5 minutes.
- *frontend-info-collector* - Utility to gather information about the frontend . It puts collected information into the *job.ID.diag* file.
- *gm-kick* - Sends a signal to the GM though a FIFO file to wake it up. It's used to increase responsiveness of GM.

The following modules are always run under the Unix account to which a grid user is mapped.

- *smtp-send.sh* and *smtp-send* - These are the modules responsible for sending e-mail notifications to the user. The format of the mail messages can be easily changed by editing the simple shell script *smtp-send.sh*.
- *submit-*-job* - Here * stands for the name of the LRMS. Curently supported LRMS are PBS/Torque, Condor and SGE. Also *fork* pseudo-LRMS is supported for testing purposes. This module is responsible for job submission to the LRMS.
- *cancel-*-job* - This script is for canceling jobs which have been already submitted to the LRMS.
- *scan-*-job* -This shell script is responsible for notifying the GM about completion of jobs. It's implementation for PBS uses server logs to extract information about jobs. If logs are not available it uses the less reliable *qstat* command for that. Other backends use different techniques.

In addition, there is also an administration utility:

- *gm-jobs* - prints a list of jobs available on the cluster and the number of jobs in each state.
  gm-jobs [-h] [-l] [-u uid] [-U name]
  -l – print more information about each job,
  -u – pretend utility is run by user with id *uid*,
  -U – pretend utility is run by user with name *name*.

GM comes with plugins useable for various authorization purposes (see for example the description of *authplugin* command below):

- *inputcheck* - checks if all input files specified in job description are downloadable.
  inputcheck [-h] [-d debug_level] RSL_fle [proxy_file]
  - RSL_file – file with job description,
  - proxy_file – credentials proxy.

- *lcas* - executes LCAS plugins on credentials and returns 0 if authorization passed.
  lcas credentials description [library [db [directory]]]
  - credentials – path to file with credentials to authorize,
  - description – path to file with job description,
  - library – path to LCAS library (full or relative to LCAS directory),
  - db – path to LCAS DB file (full or relative to LCAS directory),
  - directory – LCAS directory.

8.2 Directories

The GM is installed into a single installation point referred as $NORDUGRID_LOCATION and the following subdirectories are used:

$NORDUGRID_LOCATION/bin – tools
$NORDUGRID_LOCATION/libexec – program modules used by GM
$NORDUGRID_LOCATION/etc – configuration files, deprecated, central configuration file is used by default
$NORDUGRID_LOCATION/sbin – daemons
$NORDUGRID_LOCATION/lib – gridftp server's plugins and API libraries

The GM also uses following directories:

- *session root directory* - Tthis is the directory under which a user's SDs are created. It's location is configurable per UNIX user. Several (or even all) users may share the same session root directory. .
  There are 2 processes which need to have permissions to create new files and directories in it : GM and GFS.
  If any of these processes are run under a dedicated user account, that account needs full permissions in the *session root directory*.
  If these processes are run under the *root* account, make sure *session root directory* resides on a filesystem which does not limit the capabilities of the *root* user (as does for example NFS with *root_squash* option).
  If there is a need to run processes under the *root* account (to be able to run jobs in LRMS under different users' accounts, for example) but there is no way to provide a suitable *session root directory,* use the *norootpower* command in GM's configuration file. In that case GM and GFS will use the identity of the local user to which a Grid identity is mapped to access the *session root directory*. Hence those users will need full access there.
  The GM creates SDs with proper ownership and permissions for the local identity used to run a job. Some filesystems require users to have *execute* permission on the *session root directory* in order to access any file or subdirectory there.
  In order for jobs to access their input files, session root directories should be shared across cluster nodes. Otherwise, LRMS-specific methods must be used to transfer files to execution nodes. For more information see section 9.4.

- *control directory* - In this directory GM and GFS store an information about accepted jobs. Both processes must have full permissions there.
  A subdirectory called *log* is created there. It is used to accumulate information about started and finished jobs. This information is periodically sent to the desired *logger service*(s). For each job start and stop event, and

for each logger service where that event must be sent, a separate file is written. Once an event is sent, the corresponding file is deleted.

9 Configuration

9.1 Configuration of the Grid Manager

The GM configuration is done through a single configuration file. Historically GM supports 2 kinds of configuration files. For old one (deprecated) it looks at following places:

- *$NORDUGRID_LOCATION/etc/grid-manager.conf*
- */etc/grid-manager.conf*

And for new one in

- */etc/arc.conf*

The old configuration file consists of empty lines, lines containing comment (line starts from #) or configuration commands. Blank spaces in arguments must be escaped using '\' or arguments must be enclosed in '"'. Command line starts from command followed by arguments separated from command and between them by spaces.

The new configuration file can also contain empty lines and comments starting from #. It is separated into sections. Each sections starts from string containing

- *[section name/subsection name/subsubsection name].*

Each section continues till next section of end of file. One configuration file can have commands for multiple services/modules/programs. Each service get it's own section named after it. The GM uses section *[grid-manager]*. Some services can make use of multiple subsections to reflect their internal modular structure. Commands in section *[common]* apply to all services. Command lines have format

- *name="arguments string".*

Names are same as in old configuration file. The *argument string* consists of same arguments as in old format. And they must obey same rules.

Both files support almost same commands. Following commands are defined (examples are given for new format):

Global commads (those which affect global parameters of the GM and affect all serviced users, also described in [5]):

- ***daemon****=yes|no* - specifies whether the GM should go to background after started. Defaults to *yes*.
- ***logfile****=[path]* - specifies name of file for logging debug/informational output. Defaults to */dev/null* for daemon mode and *stderr* for foreground mode.
- ***user****=[uid[:gid]]* - specifies user id (and optionally group id) to which the GM must switch after reading configuration. Defaults to *not switch.*
- ***pidfile****=[path]* - specifies file where id if GM process will be stored. Defaults to *not write*.

- ***debug***=*number* - specifies level of debug information. More information is printed for higher levels. Currently highest effective number is 3 and lowest 0. Defaults to 2.

All commands above are generic for every daemon-enabled server in ARC NorduGrid toolkit (like GFS and HTTPSD).

- ***joblog***=*[path]* - specifies where to store log file containing information about started and finished jobs.

- ***jobreport***=*[URL ... number]* - specifies that GM has to report information about jobs being processed (started, finished) to centralized service running at given *URL*. Multiple entries and multiple URLs are allowed. *number* specifies how long old records have to be kept if failed to be reported. That time is in days. Last specified value becomes effective.

- ***securetransfer***=*yes|no* - specifies whether to use encryption while transferring data. Currently works for GridFTP only. Default is no. It is overridden by value specified in URL options.

- ***localtransfer***=*yes|no* - specifies whether to pass file downloading/uploading task to computing node. If set to yes the GM won't download/upload files. Instead it composes script submitted to LRMS in way to make it do that. This requires installation of GM and Globus to be accessible from computing nodes and environment variables GLOBUS_LOCATION and NORDUGRID_LOCATION to be set accordingly. Default is no.

- ***maxjobs***=*[max_processed_jobs [max_running_jobs]]* - specifies maximum number of jobs being processed by the GM at different stages:
  max_processed_jobs - maximal amount of jobs being processed by GM. This does not limit amount of jobs, which can be suNOTE:bmitted to cluster
  max_running jobs - maximal amount of jobs passed to Local Resource Management System
  Missing value or -1 means no limit.

- ***maxlod***=*[max_frontend_jobs [emergency_frontend_jobs [max_transferred_files]]]* - specifies maximum load caused by jobs being processed on frontend:
  max_frontend_jobs - maximal amount of jobs heavily using resources of frontend (applied before moving job to PREPARING and FINISHING states)
  emergency_frontend_jobs - if limit of *max_frontend_jobs* is used only by PREPARING or by FINISHING jobs aforementioned number of jobs can be moved to another state .This is used to avoid case then jobs can't finish due to big amount of recently submitted jobs.
  max_transfered_files - maximal number of files being transfered in parallel by every job. Used to decrease load on not so powerful frontends.
  Missing value or -1 means no limit.

- ***wakeupperiod***=*time* - specifies how often for external changes are performed (like new arrived job, job finished in LRMS, etc.). *time* is a minimal time period specified in seconds. Default is 3 minutes.

- ***cacheregistration***=*yes|no* - enables or disables registration of cache data into Indexing Services like RC or RLS. The default is *no*. Only files dowmloaded through *meta-url* are registred. Registration is done to same service used for obtaining information about file. For this opeartion credentials of the GM (host key and certificate) are used. If required new files storage location is registered at Indexing Service with quasi-url *cache://hostname/* and name *hostname:cache* .

- ***authplugin***=*state options plugin* - specifies *plugin* (external executable) to be run every time job is going to switch to *state*. Following states are allowed: ACCEPTED, PREPARING, SUBMIT, FINISHING, FINISHED and DELETED. If exit code is not 0 job is canceled by default. *Options* consist of *name*=*value* pairs separated by a comma. Following *name*s are supported:

timeout - specifies how long in seconds execution of the plugin allowed to last (mandatory, “*timeout*=“ can be skipped for backward compatibility).
onsuccess, *onfailure* and *ontimeout* - defines action taken in each case (*onsuccess* happens if exit code is 0). Possible actions are:
pass - continue execution,
log - write information about result into logfile and continue execution,
fail - write information about result into logfile and cancel job.

- ***localcred***=*timeout plugin* - specifies *plugin* (external executable or function in shared library) to be run every time job has to do something on behalf of local user. Execution of *plugin* may not last longer than *timeout* seconds. If *plugin* looks like *function@path* then function *int function(char*,char*,char*,...)* from shared library *path* is called (*timeout* is not functional in that case). If exit code is not 0 current operation will fail.

- ***norootpower***=*yes/no* - if set to yes all processes involved in job management will use local identity of a user to which Grid identity is mapped in order to access filesystem at path specified in ***session*** command (see below). Sometimes this may involve running temporary external process.

- ***allowsubmit***=*[group ...]* - list of authorization groups of users allowed to submit new jobs while "allownew=no" is active in *jobplugin.so* configuration (see below in section **??**). Multiple commands are allowed.

- ***speedcontrol***=*min_speed min_time min_average_speed max_inactivity* - specifies how long/slow data fransfer is allowed to take place. Transfer is canceled if transfer rate (bytes per second) is lower than *min_speed* for at least *min_time* seconds, or if average rate is lower than *min_average_speed*, or no data is receved for longer than *max_inactivity* seconds.

- ***copyurl***=*template replacement* - specifies that URLs, starting from template should be accessed in a different way (most probably Unix open). The *template* part of the URL will be replaced with *replacement. replacement* can be either URL or local path starting from '/'. It is advisable to end template with '/'.

- ***linkurl***=*template replacement [node_path]* - mostly identical to *copyurl* but file won't be copied. Instead soft-link will be created. *replacement* specifies the way to access the file from the frontend, and is used to check permissions. The *node_path* specifies how the file can be accessed from computing nodes, and will be used for soft-link creation. If *node_path* is missing - *local_path* will be used instead. Both *node_path* and *replacement* should not be URLs.

  NOTE: URLs which fit into *copyurl* or *linkurl* are treated as more easily accessible than other URLs. That means if GM has to choose between few URLs from which should it download input file, these will be tried first.

Per UNIX user commands:

- ***mail***=*e-mail_address* - specifies an email address **from** which the notification mails are sent.

- ***defaultttl***=*ttl [ttr]* - specifies the time in seconds for the SD to be available after job finished (*ttl*) and after job was deleted (*ttr*) due to *ttl*. Defaults are 7days for *ttl* and 30 days for *ttr*.

- ***defaultlrms***=*default_lrms_name default_queue_name* - specifies names for the LRMS and queue. Queue name can also be specified in the JD (currently it is not allowed to override used LRMS by using JD). In new configuration file this command is called ***lrms***.

- ***session***=*path* - specifies path to the directory in which the SD is created. If the path is * the default one is used - *$HOME/.jobs* . In new configuration file this command is called ***sessiondir***.

- ***cachedir***=*path [link_path]* - specifies the directory to store cached data. Empty *path* disables caching. Default is not to cache data. Optional *link_path* specifies the path at which cache is accessible at computing nodes. If *link_path* is set to '.' files are not soft-linked, but copied to session directory. In old configuration file this command is called ***cache***.
- ***privatecache***=*path [link_path]* - same as *cache* command, but cache belongs (owned) to user. For shared caches use 'cache'.
- ***cachedata***=*path* - allows to specify separate place to store cache files containing data itself. This can be useful in case of big data storage available only on NSF server which does not support file locking. If command or *path* is missing - default is to store data at place specified in *cache* or *privatecache* command, together with control files.
- ***cachesize***=*high_mark [low_mark]* - specifies high and low water-mark for space used by cache. Values are specified in bytes. Both *high_mark* and *low_mark* can be negative values. In that case corresponding positive value means space left on filesystem. If *low_mark* is omitted it becomes equal to *high_mark*. By default this feature is turned off. To turn it off explicitly *cachesize* without parameters should be specified. If turned off cache will grow up till it fills whole file system.
- ***maxrerun***=*number* - specifies maximal number of times job will be allowed to rerun after it failed in LRMS. Default value is 2. This only specifies a upper limit. Actual number is provided in job description and defaults to 0.

All per-user commands should be put before *control* command which initiates serviced user.

- ***control***=*path username [username [...]]* - This option initiates UNIX user as being serviced by the GM. *path* refers to the control directory (see section 6 for the description of control directory). If the path is * the default one is used - $HOME/.jobstatus . *username* stands for UNIX name of the local user. Multiple names can be specified. If the name is * it is substituted by all names found in file /etc/grid-security/grid-mapfile (for the format of this file one should study the Globus project [6]).
  Also the special name '.'(dot) can be used. Corresponding control directory will be used for **any** user. This option should be the last one in the configuration file. In new configuration file command ***controldir***=*path* is also available. It uses special username '.' and is always executed last independent of placement in file.
- ***helper***=*username command [argument [argument [...]]]* - associates external program with the local UNIX user. This program will be kept running under account of the specified user. *username* stands for the name of the user. Special names can be used: '*' - all names from /etc/grid-security/grid-mapfile, '.' - root user. The user should be already configured with *control* option (except root, who is always configured). *command* is an executable and *argument*s are passed as arguments to it.

Following are global commands supported only in new configuration file. Most of them are specific to underlying LRMS (PBS in this case) and are passed in environment variables if old configuration file is used.

- ***pbs_bin_path***=*path* - path to directory which contains PBS commands.
- ***pbs_log_path***=*path* - path to directory with PBS server's log files.
- ***gnu_time***=*path* - path to *time* utility.
- ***tmpdir***=*path* - path to directory for temporary files.
- ***runtimedir***=*path* - path to directory which contains *runtimenvironment* scripts.

- ***shared_filesystem***=*yes|no* - if compiting nodes have an access to session directory through a shared filesystem like NFS. Corresponds to an environement variable RUNTIME_NODE_SEES_FRONTEND.
- ***nodename***=*command* - command to obtain hostname of computing node.
- ***scratchdir***=*path* - path on computing node where to move session directory before execution.
- ***shared_scratch***=*path* - path on frontend where ***scratchdir*** can be found.
- ***nodename***=*command* - command to obtain hostname of computing node.

In the command arguments (paths, executables, ...) following substitutions can be used:

%R	- session root - see command *session*
%C	- control dir - see command *control*
%U	- username
%u	- userid - numerical
%g	- groupid - numerical
%H	- home dir - home specified in /etc/passwd
%Q	- default queue - look command 'defaultlrms'
%L	- default lrms - look command 'defaultlrms'
%W	- installation path - ${NORDUGRID_LOCATION}
%G	- globus path - ${GLOBUS_LOCATION}
%c	- list of all control directories
%I	- job's ID (for plugins only, substituted in runtime)
%S	- job's state (for *authplugin* plugins only, substituted in runtime)
%O	- reason (for *localcred* plugins only, substituted in runtime). Possible reasons are:

new	- new job, new credentials
renew	- old job, new credentials
write	- write/delete file, create/delete directory (through gridftp)
read	- read file, directory, etc. (through gridftp)
extern	- call external program (grid-manager)

Some configuration parameters can be specified from command line while starting the GM:

grid-manager [-h] [-C level] [-d level] [-c path] [-F] [-U uid[:gid]] [-L path] [-P path]
-h - short help,
-d - debug level,
-L - name log file (overwrites value in configuration file),
-P - name for file containing process id (overwrites value in configuration file),

-U - user and gropu id to use for running daemon,

-F - do not make process daemon,

-c - name od configuration file,

-C - remove old information before starting: 1- remove finished jobs, 2 - remove active jobs too, 3- also remove everything that looks like junk.

9.2

-
-
-
-
-
-
-
  -
  -
  -
  -
    -
    -
-
-
-
-

•

  –

    ∗

    ∗

    ∗

  –

    ∗

    ∗

      ·

      ·

      ·

      ·

      ·

      ·

      ·

      ·

      ·

      ·

      ·

      ·

  –

9.3 Authorization

Authorization is performed at GFS by applying set of rules. Each rule takes one line in the *group* section. For information about supported rules please read "Configuration and authorisation of ARC (NorduGrid) Services" [5].

9.4 LRMS support

The GM comes with support for several LRMS. And this number is slowly growing. Features explained below are for **PBS** backend. This support is provided through *submit-pbs-job*, *cancel-pbs-job*, *scan-pbs-job* scripts. *submit-pbs-job* creates job's script and submits it to PBS. Created job's script is responsible for moving data between frontend machine and cluster node (if required) and execution of actual job. Alternatively it can download input files and upload output if *"localtransfer no"* is specified in the configuration file.

Behavior of submission script is mostly controlled using environment variables. Most of them can be specified on frontend in GM's environment and overwritten on cluster's node through PBS configuration. Some of them may be set in configuration file too.

PBS_BIN_PATH - path to PBS executables. Like */usr/local/bin* for example. *pbs_bin_path* configuration command.

PBS_LOG_PATH - path to PBS server logs. *pbs_log_path* configuration command.

TMP_DIR - path to directory to store temporary files. Default value is */tmp*. *tmpdir* configuration command.

RUNTIME_CONFIG_DIR - path where runtime setup scripts can be found. *runtimedir* configuration command.

GNU_TIME - path to GNU time utility. It is important to path to utility compatible with GNU time. If such utility is not available, modify *submit-pbs-job* to either reset this variable or change usage of available utility. *gnu_time* configuration command.

NODENAME - command to obtain name of cluster's node. Default is */bin/hostname -f*. *nodename* configuration command.

RUNTIME_LOCAL_SCRATCH_DIR - if defined should contain path to the directory on computing node, which can be used to store job's files during execution. *scratchdir* configuration command.

RUNTIME_FRONTEND_SEES_NODE - if defined should contain path corresponding to *RUNTIME_LOCAL_SCRATCH_DIR* as seen on **frontend** machine. *shared_scratch* configuration command.

RUNTIME_NODE_SEES_FRONTEND - if set to "*no*" means computing node does not share filesystem with frontend. In that case content of the SD is moved to computing node by using means provided by the LRMS. Results are moved back after job's execution in a same way. *shared_filesystem* configuration command.

Figures 2,3,4 present some possible combinations for RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NOD
and explain how data movement is performed. Pictures a) corresponds to the situation right after all input files have been gathered in the session directory and show the actions taken right after the job's script starts. Pictures b) shows the situation while the job is running and the actions which are taken right after it has finished. Pictures c) illustrates the final situation, when the job's output files are ready to be uploaded to an external storage element or be downloaded by the user.

9.5 Runtime environment

The GM can run specially prepared *BASH* scripts prior creation of job's script, before and after executing job's main executable. Those scripts are requested by user through *runtimeenvironment* attribute in RSL and are run with only argument set equal to '0', '1' or '2' during creation of job's script, before execution of main executable and after main executable finished accordingly. They all are run through BASH's 'source' command, and hence can manipulate with shell variables. With argument '0' scripts are run by the GM on frontend. Some environment variables are defined in that case and can be changed to influence job's execution later:

- joboption_directory - session directory.

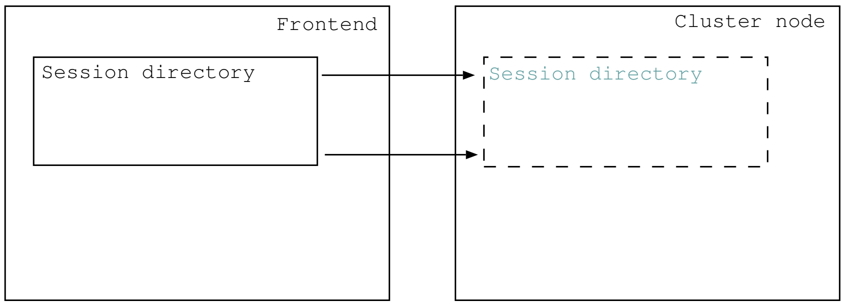

Figure 2: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE are undefined. It is assumed that session directories are visible from computing nodes. The job is executed directly in the session directory prepared by GM on the frontend.

- joboption_args - command to be executed as specified in RSL.
- joboption_env_# - array of 'NAME=VALUE' environment variables (**not** bash array).
- joboption_runtime_# - array of requested *runtimeenvironment* names (**not** bash array).
- joboption_num - *runtimeenvironment* currently beeing processed (number starting from 0).
- joboption_stdin - name of file to be attached to stdin handle.
- joboption_stdout - same for stdout.
- joboption_stderr - same for stderr.
- joboption_maxcputime - amout of CPU time requested (minutes).
- joboption_maxmemory - amout of memory requested (megabytes).
- joboption_count - number of processors requested.
- joboption_lrms - LRMS to be used to run job.
- joboption_queue - name of a queue of LRMS to put job into.
- joboption_nodeproperty_# - array of properties of computing nodes (LRMS specific, **not** bash array).
- joboption_jobname - name of the job as given by user.
- joboption_rsl - whole RSL for very clever submission scripts.
- joboption_rsl_*name* - RSL attributes and values (like joboption_rsl_executable="/bin/echo")

For example *joboption_args* could be changed to wrap main executable. Or *joboption_runtime* could be expanded if current one depends on others.

With argument '1' scripts are run just before main executable is run. They are executed on computing node. Such script can prepare environment for some third-party software package. A current directory in that case is one which would be used for execution of job. Variable HOME also points to that directory.

With argument '2' scripts are executed after main executable finished. Main purpose is to clean possible changes done by scripts run with '1' (like removing temporary files). Execution of scripts at that stage also happens on computing node and is not reliable. If the job is killed by LRMS they most probably won't be executed.

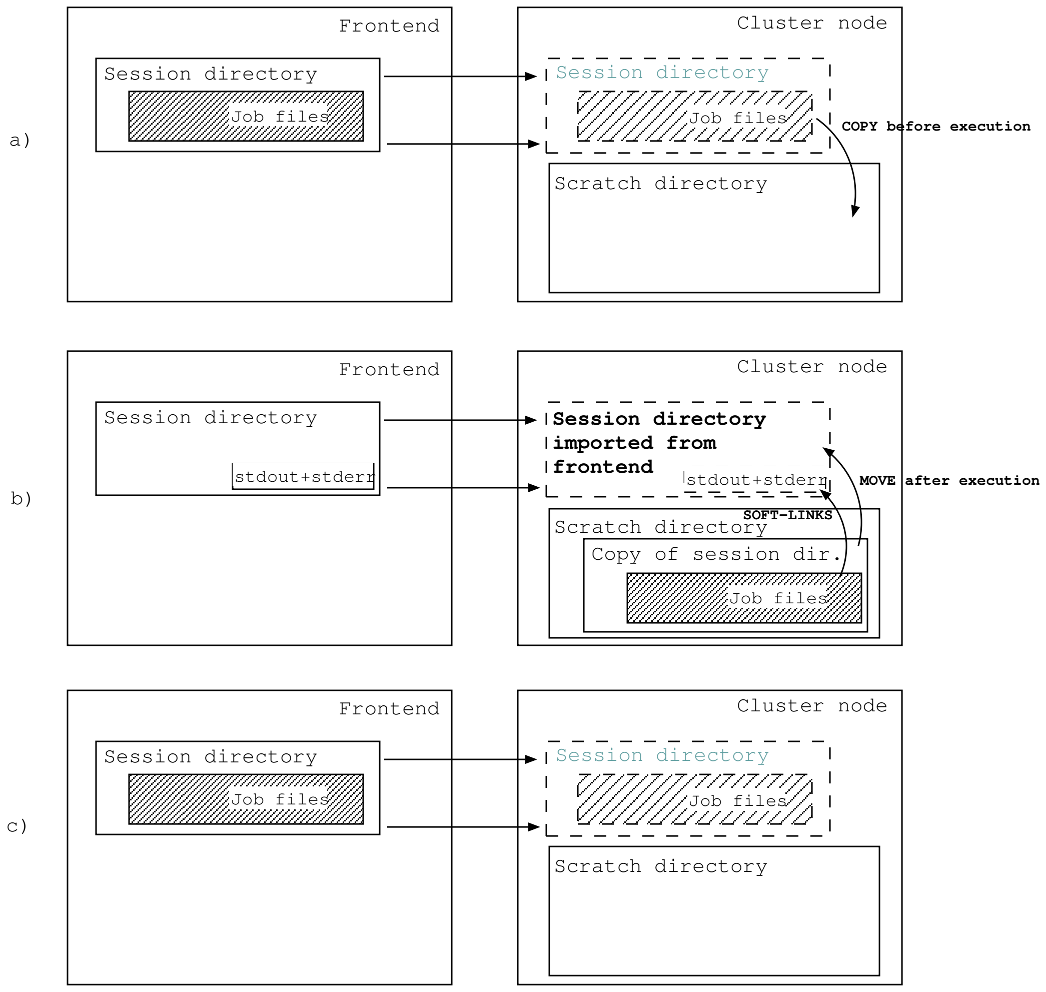

Figure 3: RUNTIME_LOCAL_SCRATCH_DIR is set to the location of the scratch directory on the computing node, RUNTIME_FRONTEND_SEES_NODE is undefined.

a) After the job script starts all input files are moved to 'scratch directory' on the computing node.

b) The job runs in a separate directory in 'scratch directory'. Only the files representing the job's *stdout* and *stderr* are placed in the original 'session directory' and soft-linked in 'scratch'. After execution all files from 'scratch' are moved back to the original 'session directory'.

c) All output files are in 'session directory' and are ready to be uploaded/downloaded.

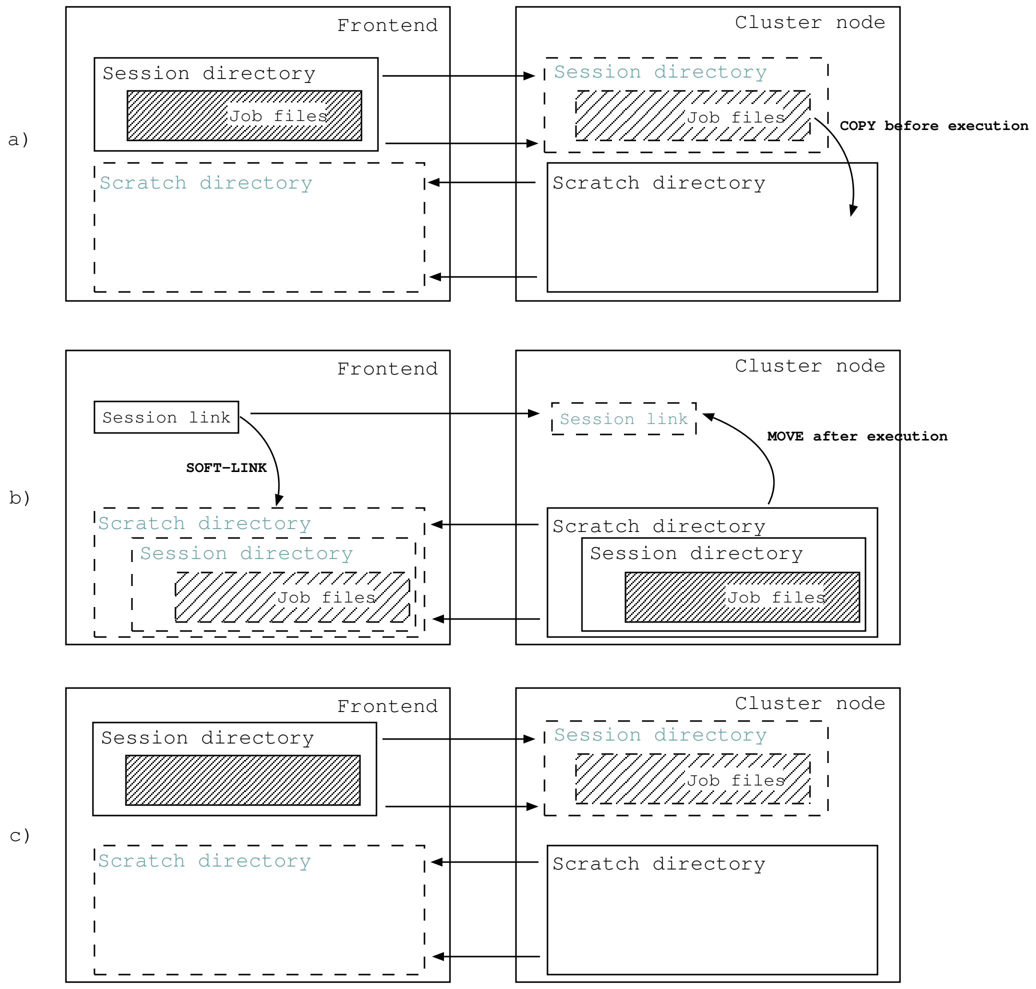

Figure 4: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE are set to the location of the scratch directory on the computing node and the location where this scratch directory is accessible from the frontend, respectively .

a) After the job script starts all input files are moved to 'scratch directory' on the computing node. The original 'session directory' is removed and replaced with a soft-link to copy of session directory in 'scratch directory' as seen on the frontend.

b) The job runs in a separate directory in 'scratch directory'. All files are also available on frontend through the soft-link. After execution the soft-link is replaced with a directory and all files from 'scratch' are moved back to the original 'session directory'.

c) All output files are in 'session directory' and are ready to be uploaded/downloaded.

10 Installation

To install GM as part of ARC-enabled site please read "NorduGrid ARC server installation instructions" at http://www.nordugrid.org/documents/ng-server-install.html.

10.1 Requirements

The GM is mostly written using C++. It was tested and should compile on recent enough *Linux* systems using *gcc* compiler and *GNU make* (gcc versions 2.95, 2.96, 3.2, 3.4 were tested). You will also need *Globus Toolkit*^{TM} of version higher than 2.2 installed http://www-unix.globus.org/toolkit/.

10.2 Setup of the Grid Manager

For in-depth information about how to properly setup the GM and related software please read "NorduGrid ARC server installation instructions" at http://www.nordugrid.org/documents/ng-server-install.html. Follow that manual to install GM, configure and run it. Additional tips are described here.

The GM is designed to be able to run both as root and as ordinary user. You can chose the name of the user by using corresponding command in configuration file. It is better run GM as root if You want to serve few users.

The GM writes debug information into a file /var/log/grid-manager.log by default. . Also file /var/log/gm-jobs.log (default path in configuration template, turned off by default) contains information about all started and finished jobs, 2 lines per job (1 when job is started and 1 after it finished).

10.3 Setup of the GridFTP Server

For in-depth information about how to properly setup the GM and related software please read "NorduGrid ARC server installation instructions" at http://www.nordugrid.org/documents/ng-server-install.html. Follow that manual to install GM, configure and run it. Additional tips are described here.

To make GFS to interoperate with other parts of the ARC only one *jobplugin.so* needs to be configured. It is advisable to use the template configuration file. You can leave only part which configures *jobplugin.so* plugin.

10.4 Usage

Refer to the description of the *User Interface* part [7] and extensions to RSL [8] for using the GM.

10.5 Running as non-root

Bot GM and GFS are primarily designed to be run by the *root* UNIX account and serve multiple global Grid identities mapped to several UNIX accounts. Nevertheless it is possible to use *non-root* accounts to run these services at the cost of some functionality loss as described below.

There are no drawbacks of running GFS with *gaclplugin* or *fileplugin* under a *non-root* account as long as the only Grid identity used is that of the user who runs the services and all served files and directories are owned by the server's account.

In order for GM and GFS with *jobplugin* to be able to cooperate, both services must be run either by the same non-root account or one of the services must be run under the *root* account. This is needed because the two services communicate over a local filesystem, hence they must have *full* access to the same set of files.

As long as GFS with *jobplugin* is run under a non-root account , no mapping from Grid identity to local UNIX account takes place. All alowed Grid users are assigned the server's account and are then processes by GM using same account. The only way to overcome this limitation is to run one GFS per local account with proper access control configured.

Because GM has to impersonate a user's local account while communicating with the LRMS, it can serve only the account it is run under (unless it is run under the *root* account, of course). As in the case of GFS, multiple instances of GM may be run, one per local account. This solution has some drawbacks. The GM looses possibility to share cached files among serviced users. In addition, it is not possible to control the load on a frontend by limiting the number of simultatenuosly running *downloader* and *uploader* modules.

One also has to take into account that the private part of the GSI infrastructure (private key of a host at least) has to be duplicated for every account used to run GFS.

Appendix A. Job control over jobplugin.so

Virtual tree

Under the mount point of the jobplugin, a user connecting with a gridftp client will see virtual directories representing the jobs belonging to him/her. Directory names are job identifiers, each representing one job . These directories are directly connected to session directories of jobs and contain the files and subdirectories that are visible on the frontend.

If a job's xRSL has *gmlog* specified, then the job's directory also contains a virtual subdirectory of that name holding files with information about the job as created by GM. The 'errors' and 'status' files are especially usefull for troubleshooting. 'errors' contains the stderr output of the various modules run by GM during job processing stages (downloader, uploader, job's submission to LRMS). 'status' contains one word representing the job's state .

Also under the jobplugin's mount point there is an additional directory named "new" – used to submit new jobs – and another directory named ”info”. The latter has subdirectories named after job ids , each of which contains files with information about a job. These are the same files that can be accessed in the subdirectory specified through *gmlog*.

Submission

Each xRSL put into the “new” directory by a gridftp client is treated as a new job's description. GFS's jobplugin parses the xRSL and returns the client a positive response if there were no errors in the request.

The new job gets an identifier and a directory with the corresponding name appears. If the job's description contains input files which should be transferred from the client's machine, the client must upload them to that directory under specified names.

The job identifier reserved by GFS to a new job must be somehow communicated back to the client. Within the bounds of the FTP protocol, this is achieved in the following way. Prior to uploading the xRSL, the client issues a CWD command to change the current directory to “new”. In turn, the server responds with a redirect to the new session directory named after the reserved identifier. The client has now found out the job's id, and proceeds with uploading the xRLS.

Actions

Various actions to affect processing of an existing job are requested by uploading special xRSL files into directory “new”. Such an xRSL may consist of only 2 parameters - *action* for action to be performed, and *jobid* to identify the job to be affected. All other parameters are ignored.

The currently supported actions are:

cancel	to cancel a job
clean	to remove a job from computing resource
renew	to renew credentials delegated to a job
restart	to restart a job after failure at some phases

Alternatively, it is also possible to perform some of these actions by using the shortcut FTP operations described below.

Cancel

A job is canceled by performing a DELE (delete file) command on the directory representing the job. It can take some time (a few minutes) before the job is actually canceled. Nevertheless, the client gets a response immediately.

Clean

A job’s content is cleaned by performing a RMD (remove directory) command on the directory representing the job. If the job is in "FINISHED" state it will be cleaned immediately. Otherwise it will be cleaned after it reaches state "FINISHED".

Renew

If a client requests CWD to a job’s session directory, credentials passed during authentication are compared to the current credentials of the job. If the validity time of the new credentials is longer, the job’s current credentials are replaced with the new ones.

Appendix B. Library *libarcdata*

libarcdata is now part of *libngui* library. It’s functions are declared in a header file *arcdata.h*. They correspond to ng* utilities meant for data handling - *arcacl*, *arccp*, *arcls*, *arcrm*, *arctransfer*. It consists of following functions:

```
void arcacl(const std::string& file_url, const std::string& command, int timeout = 0);
void arcregister (const std::string& source_url, const std::string& destination_url, bool secure = fa
void arccp (const std::string& source_url, const std::string& destination_url, bool secure = false, b
void arcls(const std::string& dir_url, bool show_details = false, bool show_urls = false, int recursi
void arcrm(const std::string& file_url, bool errcont = false, int timeout = 0);
void arctransfer(const std::string& destination, std::list<std::string>& sources, int timeout = 0);
```

Additionally this library contains C++ classes used by *ng** data management utilities. Those are described in “ARC::DataMove Reference Manual”.

Appendix C. Error messages of GM

If job has not finished successfully the GM put one or more lines into *job.ID.failed*. Possible valuesinclude those generated by the GM itself:

Error string	*Reason/description*
Internal error	Error in internal algorithm
Internal error: can't read local file	Error manipulating files in the control directory
Failed reading local job information	-//-
Failed reading status of the job	-//-
Failed writing job status	-//-
Failed during processing failure	-//-
Serious troubles (problems during processing problems)	-//-
Failed initiating job submission to LRMS	Could not run backend executable to pass job to LRMS
Job submission to LRMS failed	Backend executable supposed to pass job to LRMs returned non-zero exit code
Failed extracting LRMS ID due to some internal error	Output of Backend executable supposed to contain local ID of passed job could not be parsed
Failed in files upload (post-processing)	Failed to upload some or all output files
Failed in files upload due to expired credentials - try to renew	Failed to upload some or all output files most probably due to expired credentials (proxy certificate)
Failed to run uploader (post-processing)	Could not run *uploader* executable
uploader failed (postprocessing)	Generic error related to *uploader* component
Failed in files download (pre-processing)	Failed to upload some or all input files
Failed in files download due to expired credentials - try to renew	Failed to download some or all input files most probably due to expired credentials (proxy certificate)
Failed to run downloader (pre-processing)	Could not run *downloader* executable
downloader failed (preprocessing)	Generic error related to *downloader* component
User requested to cancel the job	GM detected external request to cancel this job, most probably issued by user
Could not process RSL	Job description could not be processed to syntax errors or missing elements
User requested dryrun. Job skiped.	Job description contains request not to process this job
LRMS error: (CODE) DESCRIPTION	LRMS returned error. CODE is replaced with numeric code of LRMS, and DESCRIPTION with textual description
Plugin at state STATE failed: OUTPUT	External plugin specified in GM's configuration returned non-zero exit code. STATE is replcaced by name of state to which job was going to be passed, OUTPUT by textual output generated by plugin.
Failed running plugin at state STATE	External plugin specified in GM's configuration could not be executed.

Provided by downloader component (URL is replcaced by source of input file, FILE by name of file):

Error string	*Reason/description*
Internal error in downloader	Generic error
Input file: URL - unknown error	Generic error
Input file: URL - unexpected error	Generic error
Input file: URL - bad source URL	Source URL is either malformed or not supported
Input file: URL - bad destination URL	Shouldn't happen
Input file: URL - failed to resolve source locations	File either not registred or other problems related to Data Indexing service.
Input file: URL - failed to resolve destination locations	Shouldn't happen
Input file: URL - failed to register new destination file	Shouldn't happen
Input file: URL - can't start reading from source	Problems related to accessing instance of file at Data Storing service.
Input file: URL - can't read from source	-//-
Input file: URL - can't start writing to destination	Access problems in a session directory
Input file: URL - can't write to destination	-//-
Input file: URL - data transfer was too slow	Timeouted while trying to download file
Input file: URL - failed while closing connection to source	Shouldn't happen
Input file: URL - failed while closing connection to destination	Shouldn't happen
Input file: URL - failed to register new location	Shouldn't happen
Input file: URL - can't use local cache	Problems with GM cache
Input file: URL - system error	Operating System returned error code where unexpected
Input file: URL - delegated credentials expired	Access to source requires credententials and they are either outdated or missing (not delegated).
User file: FILENAME - Bad information about file: checksum can't be parsed.	In job description there is a checksum provided for file uploadable by user interface and this record can't be interpreted.
User file: FILENAME - Bad information about file: size can't be parsed.	In job description there is a size provided for file uploadable by user interface and this record can't be interpreted.
User file: FILENAME - Expected file. Directory found.	Instead of file uploadable by user interface GM found directory with same name in a session directory.
User file: FILENAME - Expected ordinary file. Special object found.	Instead of file uploadable by user interface GM found special object with same name in a session directory.
User file: FILENAME - Delivered file is bigger than specified.	The size of file uploadable by user interface is bigger than specified in job description.
User file: FILENAME - Delivered file is unreadable.	GM can't check user uploadable file due to some internal error. Most probably due to improperly configured local permissions.
User file: FILENAME - Could not read file to compute checksum.	GM can't read user uploadable file due to some internal error. Most probably due to improperly configured local permissions.

Provided by uploader component (URL is replcaced by destination of output file) :

Error string	*Reason/description*
Internal error in uploader	Generic error
Output file: URL - unknown error	Generic error
Output file: URL - unexpected error	Generic error
User requested to store output locally URL	Destination is URL of type *file*.
Output file: URL - bad source URL	Shouldn't happen
Output file: URL - bad destination URL	Destination URL is either malformed or not supported
Output file: URL - failed to resolve source locations	Shouldn't happen
Output file: URL - failed to resolve destination locations	Problems related to Data Indexing service.
Output file: URL - failed to register new destination file	-//-
Output file: URL - can't start reading from source	User request to store output file, but there is no such file or there are problems accessing session directory
Output file: URL - can't start writing to destination	Problems with Data Storing services
Output file: URL - can't read from source	Problems accessing session directory
Output file: URL - can't write to destination	Problems with Data Storing services
Output file: URL - data transfer was too slow	Timeout during transfer
Output file: URL - failed while closing connection to source	Shouldn't happen
Output file: URL - failed while closing connection to destination	Shouldn't happen
Output file: URL - failed to register new location	Problems related to Data Indexing service.
Output file: URL - can't use local cache	Shouldn't happen
Output file: URL - system error	Operating System returned error code where unexpected
Output file: URL - delegated credentials expired	Access to destination requires credententials and they are either outdated or missing (not delegated).

Coming from LRMS (PBS) backend:

Error string	*Reason/description*
Submission: Configuration error.	
Submission: System error.	
Submission: Job description error.	
Submission: Local submission client behaved unexpectedly.	
Submission: Local submission client failed.	

Appendix D. A-Rex WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://www.nordugrid.org/schemas/a-rex"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsa="http://www.w3.org/2005/08/addressing"
 xmlns:bes-factory="http://schemas.ggf.org/bes/2006/08/bes-factory"
 xmlns:bes-mgmt="http://schemas.ggf.org/bes/2006/08/bes-management"
 xmlns:deleg="http://www.nordugrid.org/schemas/delegation"
 xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/rpw-2"
 xmlns:a-rex="http://www.nordugrid.org/schemas/a-rex">
  <wsdl:import namespase="http://schemas.ggf.org/bes/2006/08/bes-factory" location="./bes-factory.w
  <wsdl:import namespase="http://schemas.ggf.org/bes/2006/08/bes-management" location="./bes-manage
  <wsdl:import namespase="http://www.nordugrid.org/schemas/delegation" location="../schemas/delegat
  <wsdl:import namespase="http://docs.oasis-open.org/wsrf/rpw-2" location="http://docs.oasis-open.c
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.nordugrid.org/schemas/a-rex">
      <xsd:import namespace="http://www.w3.org/2005/08/addressing" schemaLocation="./ws-addr.xsd"/>
      <xsd:simpleType name="ActivitySubStateType">
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Accepting"/>
          <xsd:enumeration value="Accepted"/>
          <xsd:enumeration value="Preparing"/>
          <xsd:enumeration value="Prepared"/>
          <xsd:enumeration value="Submiting"/>
          <xsd:enumeration value="Executing"/>
          <xsd:enumeration value="Killing"/>
          <xsd:enumeration value="Executed"/>
          <xsd:enumeration value="Finishing"/>
          <xsd:enumeration value="Finished"/>
      <xsd:enumeration value="Failed"/>
      <xsd:enumeration value="Deleted"/>
      <xsd:enumeration value="Pending"/>
      <xsd:enumeration value="Held"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="State" type="a-rex:ActivitySubStateType"/>
  <xsd:complexType name="ResourceInformationDocumentType">
    <xsd:sequence>
       <xsd:element name="BESFactory" type="bes-factory:FactoryResourceAttributesDocumentType"/>
      <xsd:complexType name="Glue2Resource" minOccurs=’0’>
        <xsd:sequence>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="Activities" minOccurs=’0’>
        <xsd:sequence>
          <xsd:complexType name="Activity" minOccurs=’0’ maxOccurs=’unbounded’>
            <xsd:sequence>
              <xsd:element name="ActivityIdentifier" type="wsa:EndpointReferenceType"/>
              <xsd:element ref="bes-factory:ActivityDocument" minOccurs=’0’/>
              <xsd:complexType name="Glue2Job" minOccurs=’0’>
                <xsd:sequence>
                  <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unb
                </xsd:sequence>
              </xsd:complexType>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ChangeActivityStatusRequestType">
    <xsd:sequence>
      <xsd:element name="ActivityIdentifier" type="wsa:EndpointReferenceType"/>
      <xsd:element name="OldStatus" type="bes-factory:ActivityStatusType" minOccurs="0"/>
      <xsd:element name="NewStatus" type="bes-factory:ActivityStatusType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="ChangeActivityStatus" type="a-rex:ChangeActivityStatusRequestType"/>
  <xsd:complexType name="ChangeActivityStatusResponseType">
    <xsd:sequence>
      <xsd:element name="NewStatus" type="bes-factory:ActivityStatusType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="ChangeActivityStatusResponse" type="a-rex:ChangeActivityStatusResponseType"/
</xsd:schema>
```

```
</wsdl:types>
<wsdl:message name="ChangeActivityStatusRequest">
  <wsdl:part name="ChangeActivityStatusRequest" element="a-rex:ChangeActivityStatus"/>
</wsdl:message>
<wsdl:message name="ChangeActivityStatusResponse">
  <wsdl:part name="ChangeActivityStatusResponse" element="a-rex:ChangeActivityStatusResponse"/>
</wsdl:message>
<wsdl:portType name="a-rex">
  <wsdl:operation name="ChangeActivityStatus">
    <wsdl:documentation>
      This operation allows any simple status change request
      which involves no additional parameters. It should be
      used to modify job/activity execution flow:
        - To put job on hold
        - To rerun job in case of failure
        - To cancel job (same as TerminateActivity of BESFActory)
        - To remove/release job - as long as non-existence is a state
        - Any other status change no supported by BES
    </wsdl:documentation>
    <wsdl:input name="ChangeActivityStatusRequest"
      message="a-rex:ChangeActivityStatusRequest"/>
    <wsdl:output name="ChangeActivityStatusResponse"
      message="a-rex:ChangeActivityStatusResponse"/>
    <wsdl:fault name="NotAuthorizedFault"
      message="bes-factory:NotAuthorizedFault"
      wsa:Action="http://schemas.ggf.org/bes/2006/08/bes-factory/BESFactoryPortType/Fault"/>
    <wsdl:fault name="InvalidActivityIdentifierFault"
      message="bes-factory:InvalidActivityIdentifierFault"
      wsa:Action="http://schemas.ggf.org/bes/2006/08/bes-factory/BESFactoryPortType/Fault"/>
    <wsdl:fault name="CantApplyOperationToCurrentStateFault"
       message="bes-factory:CantApplyOperationToCurrentStateFault"
       wsa:Action="http://schemas.ggf.org/bes/2006/08/bes-factory/BESFactoryPortType/Fault"/>
    <wsdl:fault name="OperationWillBeAppliedEventuallyFault"
       message="bes-factory:OperationWillBeAppliedEventuallyFault"
       wsa:Action="http://schemas.ggf.org/bes/2006/08/bes-factory/BESFactoryPortType/Fault"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="a-rex" type="a-rex:a-rex">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="ChangeActivityStatus">
    <soap:operation soapAction="ChangeActivityStatus"/>
    <wsdl:input name="ChangeActivityStatusRequest">
       <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="ChangeActivityStatusResponse">
       <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="NotAuthorizedFault">
      <soap:fault name="NotAuthorizedFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="InvalidActivityIdentifierFault">
      <soap:fault name="InvalidActivityIdentifierFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="CantApplyOperationToCurrentStateFault">
      <soap:fault name="CantApplyOperationToCurrentStateFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="OperationWillBeAppliedEventuallyFault">
      <soap:fault name="OperationWillBeAppliedEventuallyFault" use="literal" />
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="GetResourcePropertyDocument" type="wsrf-rpw:GetResourcePropertyDocument">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="GetResourcePropertyDocument">
    <soap:operation soapAction="GetResourcePropertyDocument"/>
    <wsdl:input name="wsrf-rpw:GetResourcePropertyDocumentRequest">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="wsrf-rpw:GetResourcePropertyDocumentResponse">
      <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ResourceUnknownFault">
      <soap:fault name="ResourceUnknownFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="ResourceUnavailableFault">
      <soap:fault name="ResourceUnavailabbleFault" use="literal" />
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="GetResourceProperty" type="wsrf-rpw:GetResourceProperty">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="GetResourceProperty">
    <soap:operation soapAction="GetResourceProperty"/>
    <wsdl:input name="wsrf-rpw:GetResourcePropertyRequest">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="wsrf-rpw:GetResourcePropertyResponse">
      <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ResourceUnknownFault">
      <soap:fault name="ResourceUnknownFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="ResourceUnavailableFault">
      <soap:fault name="ResourceUnavailabbleFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="InvalidResourcePropertyQNameFault">
      <soap:fault name="InvalidResourcePropertyQNameFault" use="literal" />
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="QueryResourceProperties" type="wsrf:QueryResourceProperties">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="QueryResourceProperties">
    <soap:operation soapAction="QueryResourceProperties"/>
    <wsdl:input name="wsrf-rpw:QueryResourcePropertiesRequest">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="wsrf-rpw:QueryResourcePropertiesResponse">
      <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ResourceUnknownFault">
      <soap:fault name="ResourceUnknownFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="ResourceUnavailableFault">
      <soap:fault name="ResourceUnavailabbleFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="InvalidResourcePropertyQNameFault">
      <soap:fault name="InvalidResourcePropertyQNameFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="UnknownQueryExpressionDialectFault">
      <soap:fault name="UnknownQueryExpressionDialectFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="InvalidQueryExpressionFault">
      <soap:fault name="InvalidQueryExpressionFault" use="literal" />
    </wsdl:fault>
    <wsdl:fault name="QueryEvaluationErrorFault">
      <soap:fault name="QueryEvaluationErrorFault" use="literal" />
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="a-rex">
  <wsdl:port name="delegation" binding="deleg:DelegationBinding">
  </wsdl:port>
  <wsdl:port name="bes-factory" binding="bes-factory:BESFactoryBinding">
  </wsdl:port>
  <wsdl:port name="bes-mgmt" binding="bes-mgmt:BESManagementBinding">
  </wsdl:port>
  <wsdl:port name="GetResourcePropertyDocument" binding="a-rex:GetResourcePropertyDocument">
  </wsdl:port>
  <wsdl:port name="GetResourceProperty" binding="a-rex:GetResourceProperty">
  </wsdl:port>
```

```
    <wsdl:port name="QueryResourceProperties" binding="a-rex:QueryResourceProperties">
    </wsdl:port>
    <wsdl:port name="a-rex" binding="a-rex:a-rex">
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Appendix E. Delegation WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://www.nordugrid.org/schemas/delegation"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsa="http://www.w3.org/2005/08/addressing"
 xmlns:deleg="http://www.nordugrid.org/schemas/delegation">
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.nordugrid.org/schemas/delegation">
      <!-- Common types -->
      <xsd:simpleType name="TokenFormatType">
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="x509"/>
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:complexType name="ReferenceType">
        <xsd:sequence>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="DelegatedTokenType">
        <xsd:sequence>
          <xsd:element name="Id" type="xsd:string"/>
          <xsd:element name="Value" type="xsd:string"/>
          <xsd:element name="Reference" type="deleg:ReferenceType" minOccurs="0" maxOccurs="unbounded
        </xsd:sequence>
        <xsd:attribute name="Format" type="deleg:TokenFormatType" use="required"/>
      </xsd:complexType>
      <xsd:element name="DelegatedToken" type="deleg:DelegatedTokenType"/>
      <xsd:complexType name="TokenRequestType">
        <xsd:sequence>
          <xsd:element name="Id" type="xsd:string"/>
          <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
      <xsd:attribute name="Format" type="deleg:TokenFormatType" use="required"/>
    </xsd:complexType>
    <xsd:element name="TokenRequest" type="deleg:TokenRequestType"/>
    <!-- Types for messages -->
    <xsd:complexType name="DelegateCredentialsInitRequestType">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DelegateCredentialsInit" type="deleg:DelegateCredentialsInitRequestType"/>
    <xsd:complexType name="DelegateCredentialsInitResponseType">
      <xsd:sequence>
        <xsd:element name="TokenRequest" type="deleg:TokenRequestType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DelegateCredentialsInitResponse" type="deleg:DelegateCredentialsInitRespo
    <xsd:complexType name="UpdateCredentialsRequestType">
      <xsd:sequence>
        <xsd:element name="DelegatedToken" type="deleg:DelegatedTokenType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="UpdateCredentials" type="deleg:UpdateCredentialsRequestType"/>
    <xsd:complexType name="UpdateCredentialsResponseType">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="UpdateCredentialsResponse" type="deleg:UpdateCredentialsResponseType"/>
    <!-- Faults -->
    <xsd:complexType name="UnsupportedFaultType">
      <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="UnsupportedFault" type="deleg:UnsupportedFaultType"/>
    <xsd:complexType name="ProcessingFaultType">
      <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="ProcessingFault" type="deleg:ProcessingFaultType"/>
    <xsd:complexType name="WrongReferenceFaultType">
      <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="WrongReferenceFault" type="deleg:WrongReferenceFaultType"/>
  </xsd:schema>
</wsdl:types>
```

```
<wsdl:message name="DelegateCredentialsInitRequest">
  <wsdl:part name="DelegateCredentialsInitRequest" element="deleg:DelegateCredentialsInit"/>
</wsdl:message>
<wsdl:message name="DelegateCredentialsInitResponse">
  <wsdl:part name="DelegateCredentialsInitResponse" element="deleg:DelegateCredentialsInitResponse"
</wsdl:message>
<wsdl:message name="UpdateCredentialsRequest">
  <wsdl:part name="UpdateCredentialsRequest" element="deleg:UpdateCredentials"/>
</wsdl:message>
<wsdl:message name="UpdateCredentialsResponse">
  <wsdl:part name="UpdateCredentialsResponse" element="deleg:UpdateCredentialsResponse"/>
</wsdl:message>
<wsdl:message name="UnsupportedFault">
  <wsdl:part name="Detail" element="deleg:UnsupportedFault"/>
</wsdl:message>
<wsdl:message name="ProcessingFault">
  <wsdl:part name="Detail" element="deleg:ProcessingFault"/>
</wsdl:message>
<wsdl:message name="WrongReferenceFault">
  <wsdl:part name="Detail" element="deleg:WrongReferenceFault"/>
</wsdl:message>
<wsdl:portType name="DelegationPortType">
 <wsdl:operation name="DelegateCredentialsInit">
    <wsdl:documentation>
    </wsdl:documentation>
    <wsdl:input name="DelegateCredentialsInitRequest"
      message="deleg:DelegateCredentialsInitRequest"/>
    <wsdl:output name="DelegateCredentialsInitResponse"
      message="deleg:DelegateCredentialsInitResponse"/>
    <wsdl:fault name="UnsupportedFault"
      message="deleg:UnsupportedFault"/>
    <wsdl:fault name="ProcessingFault"
      message="deleg:ProcessingFault"/>
  </wsdl:operation>
 <wsdl:operation name="UpdateCredentials">
    <wsdl:documentation>
    </wsdl:documentation>
    <wsdl:input name="UpdateCredentialsRequest"
      message="deleg:UpdateCredentialsRequest"/>
    <wsdl:output name="UpdateCredentialsResponse"
      message="deleg:UpdateCredentialsResponse"/>
    <wsdl:fault name="UnsupportedFault"
      message="deleg:UnsupportedFault"/>
    <wsdl:fault name="ProcessingFault"
      message="deleg:ProcessingFault"/>
    <wsdl:fault name="WrongReferenceFault"
      message="deleg:WrongReferenceFault"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="DelegationBinding" type="deleg:DelegationPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="DelegateCredentialsInit">
      <soap:operation soapAction="DelegateCredentialsInit"/>
      <wsdl:input name="DelegateCredentialsInitRequest">
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="DelegateCredentialsInitResponse">
         <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="UpdateCredentials">
      <soap:operation soapAction="UpdateCredentials"/>
      <wsdl:input name="UpdateCredentialsRequest">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="UpdateCredentialsResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Appendix F. ARC extensions for JSDL schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.nordugrid.org/ws/schemas/jsdl-arc"
            xmlns:jsdl-arc="http://www.nordugrid.org/ws/schemas/jsdl-arc"
            targetNamespace="http://www.nordugrid.org/ws/schemas/jsdl-arc">
 <xsd:simpleType name="GMState_Type">
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="ACCEPTED"/>
   <xsd:enumeration value="PREPARING"/>
   <xsd:enumeration value="SUBMIT"/>
   <xsd:enumeration value="INLRMS"/>
   <xsd:enumeration value="FINISHING"/>
   <xsd:enumeration value="FINISHED"/>
   <xsd:enumeration value="DELETED"/>
   <xsd:enumeration value="CANCELING"/>
  </xsd:restriction>
 </xsd:simpleType>
 <xsd:complexType name="Version_Type">
  <xsd:sequence>
  <xsd:element name="UpperExclusive" type="xsd:string" minOccurs="0"/>
  <xsd:element name="LowerExclusive" type="xsd:string" minOccurs="0"/>
  <xsd:element name="Exact" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element name="Exclusive" type="xsd:boolean" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="SessionType_Type">
 <xsd:documentation> For jsdl:Resources_Type </xsd:documentation>
 <!-- xsd:element ref="SessionType" minOccurs="0"/ -->
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="INTERNAL"/>
  <xsd:enumeration value="LIMITED"/>
  <xsd:enumeration value="READONLY"/>
  <xsd:enumeration value="FULL"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="IsExecutable_Type">
 <xsd:documentation> For jsdl:DataStaging_Type (default: false) </xsd:documentation>
 <!-- xsd:element ref="IsExecutable" minOccurs="0"/ -->
 <xsd:restriction base="xsd:boolean"/>
</xsd:simpleType>
<xsd:simpleType name="FileParameters_Type">
 <xsd:documentation> For jsdl:DataStaging_Type </xsd:documentation>
 <!-- xsd:element ref="IsExecutable" minOccurs="0"/ -->
 <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="JoinOutputs_Type">
 <xsd:documentation> For jsdl:JobDescription_Type (default: false) </xsd:documentation>
 <!-- xsd:element ref="JoinOutputs" minOccurs="0"/ -->
 <xsd:restriction base="xsd:boolean"/>
</xsd:simpleType>
<xsd:simpleType name="Reruns_Type">
 <xsd:documentation> For jsdl:JobDescription_Type (default: false) </xsd:documentation>
 <!-- xsd:element ref="Reruns" minOccurs="0"// -->
 <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>
<xsd:complexType name="RunTimeEnvironment_Type">
 <xsd:documentation> For jsdl:Resources_Type </xsd:documentation>
 <!-- xsd:element ref="RunTimeEnvironment" minOccurs="0" maxOccurs="unbounded"/ -->
 <xsd:sequence>
  <xsd:element name="Name" type="xsd:string"/>
  <xsd:element name="Version" type="Version_Type" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Middleware_Type">
 <xsd:documentation> For jsdl:Resources_Type </xsd:documentation>
 <!-- xsd:element ref="Middleware" minOccurs="0" maxOccurs="unbounded"/ -->
 <xsd:sequence>
  <xsd:element name="Name" type="xsd:string"/>
  <xsd:element name="Version" type="Version_Type" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="RemoteLogging_Type">
 <xsd:documentation> For jsdl:JobDescription_Type </xsd:documentation>
 <!-- xsd:element ref="RemoteLogging" minOccurs="0" maxOccurs="3"/ -->
 <xsd:sequence>
  <xsd:element name="URL" minOccurs="1" maxOccurs="1" type="xsd:anyURI"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CredentialServer_Type">
 <xsd:documentation> For jsdl:JobDescription_Type </xsd:documentation>
 <!-- xsd:element ref="CredentialServer" minOccurs="0"/ -->
 <xsd:sequence>
  <xsd:element name="URL" minOccurs="1" maxOccurs="1" type="xsd:anyURI"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LocalLogging_Type">
 <xsd:documentation> For jsdl:JobDescription_Type </xsd:documentation>
 <!-- xsd:element ref="LocalLogging" minOccurs="0" maxOccurs="1"/ -->
 <xsd:sequence>
  <xsd:element name="Directory" minOccurs="1" maxOccurs="1" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="AccessControlType_Type">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="GACL"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="AccessControl_Type">
 <xsd:documentation> For jsdl:JobDescription_Type </xsd:documentation>
 <!-- xsd:element ref="AccessControl" minOccurs="0"/ -->
 <xsd:sequence>
  <xsd:element name="OwnerAlwaysAllowed" type="xsd:boolean" minOccurs="0"/>
  <xsd:element name="Type" type="AccessControlType_Type" minOccurs="0"/>
  <xsd:element name="Content" minOccurs="0" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="NotificationType_Type">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="Email"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="Notify_Type">
 <xsd:documentation> For jsdl:JobDescription_Type </xsd:documentation>
 <!-- xsd:element ref="Notify" minOccurs="0" maxOccurs="3"/ -->
 <xsd:sequence>
  <xsd:element name="Type" type="NotificationType_Type" minOccurs="0"/>
  <xsd:element name="Endpoint" minOccurs="0" type="xsd:string"/>
  <xsd:element name="State" minOccurs="1" maxOccurs="unbounded" type="GMState_Type"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="SessionLifeTime_Type">
 <xsd:documentation> For jsdl:Resources_Type </xsd:documentation>
 <!-- xsd:element ref="SessionLifeTime" minOccurs="0" maxOccurs="1"/ -->
 <xsd:restriction base="xsd:long"/>
</xsd:simpleType>
<xsd:simpleType name="GridTimeLimit_Type">
 <xsd:documentation> For jsdl:Resources_Type </xsd:documentation>
 <!-- xsd:element ref="GridTimeLimit" minOccurs="0" maxOccurs="1"/ -->
 <xsd:restriction base="xsd:positiveInteger"/>
</xsd:simpleType>
<xsd:complexType name="CandidateTarget_Type">
 <xsd:documentation> For jsdl:Resources_Type </xsd:documentation>
 <!-- xsd:element ref="jsdl-arc:CandidateTarget" minOccurs="0" maxOccurs="1"/ -->
 <xsd:sequence>
  <xsd:element name="HostName" minOccurs="0" type="xsd:string"/>
  <xsd:element name="QueueName" minOccurs="0" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="Time_Type">
 <xsd:documentation> For jsdl:JobDescription_Type </xsd:documentation>
 <!-- xsd:element ref="ProcessingStartTime" minOccurs="0" maxOccurs="1"/ -->
 <xsd:restriction base="xsd:dateTime"/>
</xsd:simpleType>
<!--=========================================================-->
<xsd:element name="IsExecutable" type="IsExecutable_Type"/>
<xsd:element name="FileParameters" type="FileParameters_Type"/>
<xsd:element name="RunTimeEnvironment" type="RunTimeEnvironment_Type"/>
<xsd:element name="Middleware" type="Middleware_Type"/>
<xsd:element name="RemoteLogging" type="RemoteLogging_Type"/>
<xsd:element name="LocalLogging" type="LocalLogging_Type"/>
<xsd:element name="AccessControl" type="AccessControl_Type"/>
<xsd:element name="Notify" type="Notify_Type"/>
<xsd:element name="SessionLifeTime" type="SessionLifeTime_Type"/>
<xsd:element name="SessionType" type="SessionType_Type"/>
<xsd:element name="JoinOutputs" type="JoinOutputs_Type"/>
<xsd:element name="Reruns" type="Reruns_Type"/>
<xsd:element name="CredentialServer" type="CredentialServer_Type"/>
<xsd:element name="GridTimeLimit" type="GridTimeLimit_Type"/>
<xsd:element name="CandidateTarget" type="CandidateTarget_Type"/>
<xsd:element name="ProcessingStartTime" type="Time_Type"/>
```

```
</xsd:schema>
```

References

[1] A. Anjomshoaa *et al.* (2005, December) Job submission description language (jsdl) specification v1.0. GFD-R-P.056. [Online]. Available: http://www.ggf.org/ggf_docs_final.htm

[2] I.Foster *et al.* (2007, August) Ogsa basic execution service version 1.0. GFD.108. [Online]. Available: http://www.ogf.org/gf/docs/?final

[3] OASIS. (2006, April) Oasis web services resourceproperties specification. . [Online]. Available: http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf

[4] A. Konstantinov, *Protocols, Uniform Resource Locators (URL) and Extensions Supported in ARC*, The NorduGrid Collaboration, NORDUGRID-TECH-7.

[5] ——, *Configuration and Authorisation of ARC (NorduGrid) Services*, The NorduGrid Collaboration, NORDUGRID-TECH-6.

[6] I. Foster and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit,” *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[7] M. Ellert, *The NorduGrid toolkit user interface*, The NorduGrid Collaboration, NORDUGRID-MANUAL-1.

[8] O. Smirnova, *Extended Resource Specification Language*, The NorduGrid Collaboration, NORDUGRID-MANUAL-4.