AstroGrid-D

Deliverable D6.2

Requirements on grid-enabled Monitoring & Steering Methods in AstroGrid-D Applications[1]

Deliverable	D6.2
Authors	Thomas Radke (AEI)
Editors	Thomas Radke (AEI)
Date	22 December 2006
Document Version	1.0.0
Current Version	1.0.0
Previous Versions	

A: Status of this Document

Approved document for project deliverable D6.2.

B: Reference to project plan

This deliverable document refers to the task TA VI-II *"Feststellung der Anforderungen an gridfähige Überwachungs- und Steuerungsmethoden"* and milestone M6 of work package WP-6 in the project plan.

[1]This work is part of the D-Grid initiative and is funded by the German Federal Ministry of Education and Research (BMBF).

C: Abstract

This document describes the requirements of individual use cases on grid-enabled monitoring and steering methods in AstroGrid-D applications.

D: Changes History

Version	**Date**	**Name**	**Brief summary**
0.1.0	20 November 2006	Thomas Radke	Working Draft Creation
0.1.1	21 November 2006	Thomas Radke	Introduction and Requirements Overview
0.1.2	23 November 2006	Thomas Radke	Specific Requirements Added
0.1.3	24 November 2006	Thomas Radke	Finished Requirements, Outlook Added
0.2.0	1 December 2006	Thomas Radke	Working Draft submitted to entire Project with Request for Comments
0.2.1	8 December 2006	Thomas Radke	Comments and Suggestions from Project Members Added
1.0.0	22 December 2006	Thomas Radke	Document finally approved

E:

Contents

1 Introduction

Based on the information gathered in the use case enquiry about AstroGrid application scenarios, this document determines the WP-VI-specific requirements on grid-enabled monitoring & steering methods. It also incorporates the results from the analysis of already existing monitoring and steering facilities in AstroGrid applications as described in the first deliverable of this Working Group (D6.1: "*Existing Monitoring & Steering Functionality in AstroGrid-D Applications*" [1]).

Monitoring & steering methods are required mostly by astrophysical simulation applications. Data analysis codes typically do not need interactive steering, and monitoring methods are used here mainly for debugging purposes. Robotic telescopes (RTs) will have their own monitoring/steering capabilities built-in. Because actual work on RTs is scheduled to start not before the first year into the project, no RT-specific requirements are contained in this document. Instead, they will be assessed in a following requirements enquiry.

Based on the list of detailed requirements on application monitoring & steering as described in this document, the next task in Working Group VI will be to define an overall architecture which outlines the necessary steps towards the design and implementation of generalized grid-enabled monitoring and steering methods for AstroGrid. Such an architectural design will then be presented in the upcoming D6.3 deliverable design document.

2 Requirements

Given the AstroGrid use case descriptions in [1] and [2], a list of concrete requirements is derived in the following. Also named are interdependencies of Working Group VI with other work packages.

Because many use cases have very similar requirements especially in terms of monitoring, the individual requirements have been generalised into different classes:

1. interactive access to `stdout/stderr` logs
2. interactive access to application logfiles
3. application-specific monitoring methods
4. application-specific steering methods

Each class, which is described in a separate subsection, lists the use cases for which this requirement is relevant, hence each use case itself may be listed in multiple classes.

Table 2 denotes the mapping of individual use cases and their abbreviations as used in the following requirements descriptions. The list of use cases and their numbering scheme corresponds to the results obtained from the use case inquiry taken at the beginning of the project.

Only such uses case are listed which did actually specify concrete requirements on application monitoring and steering. The full list of AstroGrid use cases and their descriptions are available on the AstroGrid intranet webpages [2].

Use Case Number	Title	Responsible
UC1	NIRVANA	AIP
UC2	AMIGA	AIP
UC3	NBODY6++	ZAH
UC4	DYNAMO	AIP
UC5	Cactus	AEI
UC6	Gadget	AIP/MPA
UC7	Astrometric Matching	MPE
UC8	ClusterFinder	MPE
UC11	Millenium Processing	MPE
UC12	GEO600	AEI

Table 2: AstroGrid Use Cases with requirements on monitoring and steering

2.1 Interactive Access to `stdout/stderr` Logs

Required in use cases: UC3, UC5, UC6, UC8, UC11, UC12

Local jobs:

Many applications print runtime information to `stdout` and important notification, warning, and error messages to `stderr`. When started interactively, users can directly follow a local job's progress by monitoring the **stdout/stderr** streams on the console.

For non-interactive jobs (eg. a simulation submitted as a batch job to a cluster resource), the queuing system takes care of redirecting `stdout/stderr` into job-specific logfiles as specified in the user's batch script; depending on how the queuing system is configured, intermediate logfiles are created and being written to while the job is running, and moved to their final user-specified location only after the job has finished. In such a case the user needs some knowledge about the local queuing system's naming scheme (eg. a filename template `"<pbs_server>.<pbs_jobid>"`) and output location (eg. `PBS_SPOOLDIR` as a local spool directory on the first node where the job is running) for intermediate logfiles. This logic can be hidden in a (queuing system specific) helper script which determines the names of a batch job's intermediate logfiles and fetches them from their location.
In general, for non-interactive jobs `stdout/stderr` information is written into logfiles, and a user who wants to monitor the progress of these jobs typically checks the latest contents of such files periodically (eg. using the UNIX command `tail -f`).

Grid jobs:

For Grid jobs, `stdout/stderr` monitoring should work similarly as for local jobs: interactively started jobs can be monitored directly on the console, whereas monitoring of non-interactive jobs requires access to their logfiles. The Globus framework itself already provides commands (`globus-job-get-output, globus-job-get-output-ws`[3]) to fetch remote logfiles of simple Grid jobs (ie. ones that were not submitted as batch jobs to some local queuing system); only the job ID – or some equivalent unique contact address as created during grid job submission – must be specified, names and location of `stdout/stderr` logfiles are handled internally by the Globus job manager.

For non-interactive Grid jobs submitted as batch jobs, the coupling between the Globus job manager and the local queuing system may be such that logfiles become accessable to Globus only after the job has finished; intermediate logfiles are managed by the local job manager only, and not passed on to Globus. In this case, a higher-level service on top of Globus seems necessary which determines the configuration of the local queuing system (filename template for intermediate logfiles, location of local spool directory) and then uses basic Grid file management services (eg. `GridFtp`) to fetch such logfiles.

Conclusion:

Monitoring should be straightforward for simple Grid jobs: the Grid job management service (as being developed in AstroGrid's working group WG-V) must – in addition to the submission of Grid jobs – also provide the corresponding Globus commands to monitor the job's `stdout/stderr` output at runtime.

For Grid jobs submitted to a batch system, a proprietary solution is needed to provide the functionality missing in the standard Globus toolkit: the names and location of intermediate logfiles for batched jobs must be determined, eg. by querying the AstroGrid information service (as being developed in working group WG-II) about metadata for the given local queuing system configuration. Then standard Grid file management services (as being developed in working group WG-III) are used to fetch these files by the client. Special care needs to be taken to prevent the consistency of files which are still open and being written to: a local copy of the current intermediate logfile contents may need to be created and transfered.

Only job owners should be allowed to monitor their applications (jobs) via interactive logfile access.

2.2 Interactive Access to Application Logfiles

Required in use cases: UC1, UC2, UC3, UC4, UC5, UC6, UC8

In addition to writing to `stdout/stderr`, some AstroGrid applications also create their own logfiles and continuously append runtime information for monitoring the job's progress, or even to publish intermediate results. The names and location of such logfiles are solely determined by the application itself and therefore specific for each individual AstroGrid use case – they might be fixed in the code or set by the user as an application startup parameter. Apart from their different naming conventions, application-specific logfiles are typically used for monitoring in the same way as `stdout/stderr` logfiles: users check their latest contents periodically (eg. using the UNIX command `tail -f`).

For the interactive access to application logfiles of Grid jobs, a very similar approach as for `stdout/stderr` monitoring of Grid jobs submitted to a batch system can be applied: in the job description which is used as input for the Grid job submission service (developed in WG-V), the user also specifies the names and location of application-specific logfiles. Such metadata about the Grid job is then stored in the AstroGrid information service (WG-II) and queried by a monitoring service (WG-VI, with support from WG-V's job management service) to determine the URLs for such logfiles. Finally, standard Grid file management services (WG-III) are used to fetch the remote logfiles by the client.

2.3 Application-specific Monitoring Methods

Required in use cases: UC3, UC5

UC3 (NBODY6++) and UC5 (Cactus) have built-in monitoring methods which are very specific to the underlying application code.

The development version of the NBODY6++ code already implements capabilities for remote online monitoring and data visualisation (see section 3.3.1 in [1]). A special data transfer and service protocol SEAP[4] is used to connect from a client through a GUI client interface to an NBODY6++ job running on a remote supercomputer. The connection to the application is one-directional (from client to application) and established indirectly via a SEAP proxy machine, hence it is also possible to connect to an NBODY6++ simulation running on an internal cluster node with no direct access to the (external) internet. A VISIT[4] plugin for UNICORE provides single sign-on and secure transportation facilities (based on X.509 certificates).
Due to their *per se* grid-awareness (the possibility to connect to a remote application and exchange data via some network protocol which is also suitable for Grid environments), the above described monitoring mechanisms seem sufficient to be used also for NBODY6++ simulations submitted as Grid jobs. The SEAP proxy server will need to be configured to use the Globus ephemeridal port range (rather than its own predefined port range) in order to make it work in a Globus environment. No additional requirements are imposed.

Cactus also has sophisticated monitoring methods built into the code (see section 3.3.2 in [1]). They are based on the standard HTTP protocol: a user can use a standard web-browser to connect to a remote Cactus simulation by specifying a unique simulation URL (consisting of the hostname of the first node the job is running on, and a predetermined port number). HTTP communication channels are bi-directional (from client to application, from application back to client) and by default

established as direct connections. This usually imposes no problems for Cactus jobs running on a local resource (eg. an in-house cluster).
In a Grid environment, however, direct connections are sometimes not possible, for instance when the jobs are running on a cluster which is behind a firewall, or whose compute nodes are networked via VPN and hence invisible to the outside world. Currently users can bypass this problem manually by setting up an HTTP proxy on the cluster's head node using SSH tunnelling. In a production Grid environment this technical task should be taken over by a Grid application monitoring service which would automatically establish a proxy connection if necessary.
Because clients can also feed back information to the application (eg. in order to steer it), special care needs to be taken for security: only authorised users (eg. the job owner) should be allowed to interact with the application. Currently such security mechanisms are implemented via HTTP password authentication; for Grid jobs, however, GSI authentication should also be supported.

Another possibility of monitoring Cactus simulations is to have a job actively announce information about its progress (eg. startup/termination time) as well as other monitoring data to some external information service (WG-II) which then can be queried by users. The connection between client and application is indirect in this scenario; Cactus needs to establish only an outgoing connection to such a service which releases some of the security aspects mentioned before. However, the information service should have means to authenticate users appropriately, based on GSI user certificates.

2.4 Application-specific Steering Methods

Required in use cases: UC2, UC3, UC5

Two AstroGrid simulation code use cases UC2 (AMIGA) and UC3 (NBODY6++) provide functionality for controlled termination of jobs: a user can trigger a job to gracefully terminate by creating a special file in the job's output directory. The simulation code is periodically checking after each timestep whether such a file exists.
In order to provide this simple application steering mechanism also for Grid jobs, users need to be able to create the termination files remotely on the resource where the job is running. This can be implemented straightforward using the AstroGrid job management service (WG-V) to find out where the job is running and the file management service (WG-III) to actually create the file.

UC3 (NBODY6++) provides via the xnbody interface [5] as well a steering method: through its user interface certain parameters of the run (time step control, error margin, termination time) can be changed while the remote run is ongoing.
By grid-enabling the SEAP protocol (see section 2.3), remote steering of NBODY6++ simulations is ready to be used also in a Grid context.

As described above in section 2.3, UC5 (Cactus) provides a bi-directional HTTP-based webserver interface which can be used both for monitoring and steering Cactus simulations. The requirements on this existing functionality to make it grid-aware are again: a) network connections between client and application must be established reliably even if the latter is running on a cluster with internal-only network, and b) access by users to monitor and – especially – to steer Cactus simulations must be made secure using GSI-based user authentication.

3 Further Steps

After analysing and defining the requirements of individual use cases in this document, the next task in working group WG-VI will be to design a general architectural picture for grid-enabled monitoring & steering methods in AstroGrid applications and to come up with concrete suggestions on how the functionality necessary to meet these requirements can be implemented.

An architecture overview will be presented in the upcoming D6.3 deliverable document.

F: References / Bibliography

References

[1] Existing Monitoring & Steering Functionality in AstroGrid-D Applications. Comparison Study, Work Group VI deliverable document D6.1, AstroGrid project;
`http://www.gac-grid.org/project-documents/deliverables/wp6/WG6_D6_1.pdf`

[2] AstroGrid-D Use Case inquiry. AstroGrid Intranet webpage;
`http://mintaka.aip.de:8080/lenya/intranet/live/project-documents/usecases.html`
`http://mintaka.aip.de:8080/lenya/intranet/live/project-documents/UseCasesNew.html`

[3] Globus man page for `globus-job-get-output-ws`. Globus Toolkit 4.0 online documentation;
`http://www.globus.org/toolkit/docs/4.0/execution/wsgram/rn01re05.html`

[4] Thomas Eickermann, Wolfgang Frings, Anke Häming: VISIT 2.0 online documentation. Institut für angewandte Mathematik, Forschungszentrum Jülich.
`http://www.fz-juelich.de/zam/visit/visit20beta/manual/node144.html`

[5] Xnbody: an online visualization tool for nbody6++. Institut für angewandte Mathematik, Forschungszentrum Jülich.
`http://www.fz-juelich.de/zam/xnbody/`