

Planck infrastructure software gridification

- Requirements -

Version 1.3

H.-M. Adorf, AstroGrid-D/MPA, 2005-10-20

Abstract: This document sketches the requirements for the planned “gridification” of infrastructure software packages that have been developed within the Planck consortium: the Planck Process Coordinator (ProC) workflow engine and the related Data Management Component (DMC). The gridification will take place within the AstroGrid-D community project of the D-GRID initiative, sponsored by the BMBF. Three phases are distinguished: the first one involves only a “LAN-grid”, the second the execution of workflow jobs on a WAN-grid, and the third access to ProC functionality from the WAN-grid.

[Change Notes: The major revisions compared to the previous version concern the details of code management/distribution, data management, and job control. The second figure has been split into two. The phase-3 gridification section has substantially been extended]

Background

The Max-Planck-Institut fuer Astrophysik (MPA), Garching, is a member of the consortium developing software for the Planck Surveyor mission of the European Space Agency (ESA). The Planck Surveyor satellite is slated to be launched mid 2007. The MPA Planck team has developed several software packages supporting the complex and distributed Planck data analysis, including the Process Coordinator (ProC) and a Data Management Component (DMC).

One of the tasks of the AstroGrid-D community grid project of the D-GRID initiative, consists in gridifying the ProC. Another, related task consists in gridifying the DMC. A third task consists in gridifying the post processing of simulation data.

The Planck Process Coordinator workflow engine

The ProC is a workflow engine originally developed for running simulation and data analysis workflows (called “pipelines” in astronomical parlance) that occur within the Planck Surveyor project. Its design has been kept generic, allowing its application to many other projects.

The ProC software has been implemented in Java 1.4 under Unix flavours (Linux and MacOS X), but should be portable to other Unices and Windows (with restrictions). The ProC allows the execution of tasks written in languages other than Java (e.g. C and Fortran). A transition to Java 5 is planned for the near future.

The ProC uses an XML definition of the workflow together with a set of parameters and input data to execute a directed acyclic graph of interconnected tasks. Its central

elements allow, among other things, the execution of hierarchies of subpipelines, and serial and parallel loops on subpipelines.¹

During start-up the ProC Pipeline Coordinator (PiCo) loops through all the modules of a given workflow, and checks whether they are executable. A module is declared *executable* when all input parameters are specified, and all input data are available. An executable module is readied as a job and submitted to the underlying queuing system.

When the PiCo receives a message that a module “A” has finished processing, it checks whether there are other modules which need the output of “A” and which have now become executable. Again each executable module is submitted to the queuing system.

For further information about the ProC see the architectural design document by Doerl *et al.* (2005) and the poster by Hovest *et al.* (2001).

The Planck MPA Data Management Component

The MPA-DMC is to be used in conjunction with the ProC. (A second DMC, developed by the HFI-team of the Planck-consortium, will not be discussed here.)

At the start each ProC task is given a reference to a so-called *input object*, which it fetches from the central DMC server. The input object specifies the input parameters and the input data, which the job subsequently requests from the DMC. Communication from job to DMC proceeds via client software based on JDO which in turn uses a JDBC-connection to the database backing the DMC-server.

Gridification of the ProC

The design of the ProC has always foreseen the usage of an underlying queuing system for the execution of individual tasks, in order to harvest the compute power available on compute servers or on farms/clusters of compute nodes.

The gridification of the ProC is planned to proceed in three major phases:

- In a first phase the ProC shall² be extended to simply use the computing resources available on a local grid.
- In a second phase the ProC shall be extended/interfaced such that it may use the computing resources on a wide-area network (WAN) grid.
- In a third phase the ProC shall be described/interface in such a way that it can be executed as a grid resource.

Phase-1: ProC using a LAN-based queuing system

We wish to start with something rather simple and guaranteed to run after a very limited amount of time. Therefore the initial efforts will be concentrated on a LAN-based solution. We define the phase-1 gridification of the ProC as interfacing the latter with a

¹ Strictly speaking, an acyclic graph does not permit loops. However, the permitted graph nodes of the ProC comprise not only ordinary modules, but also control elements. One example of such a control element is a loop, which permits the repeated execution of a sub-pipeline dependent on a Boolean condition or a simple counter. Thus general cycles are not allowed in the ProC workflow graph.

² For the appropriate usage of the terms “shall”, “should”, “may”, and “can” see e.g. the ATLAS software development document.

“job manager” or “queuing system”, which controls a cluster or a pool of workstations, interconnected by a local area network (LAN) within institute boundaries.

The phase-1 ProC gridification shall be carried out, without requiring the user to install the Globus Toolkit (GT) software or any portion thereof (such as GRAM). The main obstacle, apart from the installation step, is that the GT would need authentication, which requires grid certificates.

Instead, we wish to use a “resource broker” such as Gridbus or an intermediate adapter layer such as the Grid Application Toolkit (GAT) for submitting jobs to the underlying queuing system (see Fig. 1).

Job manager “queuing” software

Members of the Planck consortium are already using the Portable Batch System (openPBS) and the Sun Grid Engine (SGE), thus the locally gridified ProC shall, at least initially, be running on top of these queuing systems.

PBS is the first local grid engine to which the ProC shall be connected. PBS is already installed at AEI, but not (yet) at MPA.

The SGE is the second local grid engine to which the ProC shall be interfaced to. A local trial installation of the SGE is running on the MPA-Planck group's computing facilities. Another SGE installation is available on the MPA computing system. The Rechenzentrum Garching (RZG) offers a professionally maintained installation of the SGE, but this is not directly available to AstroGrid-D or to D-GRID.

Ideally, for LAN-gridification we should like to channel the job submission through the Java API of a *resource broker* (or equivalent), which should allow a run-time configuration of the queuing manager actually in use.

Five things need to be managed for each job:

1. the code (scripts and/or executables),
2. the input parameters,
3. the input files,
4. the output files, and
5. status information.

For the LAN-grid ProC the dependencies between individual jobs will be handled completely by the ProC. A queuing system probably does not know how to handle a complete workflow anyway.

Code management and distribution

The modules permitted by the ProC may be *heterogeneous*. each may consist of a script and one or more executables., which, for scientific data analysis, are often written in non-portable code (i.e. C/C++ or Fortran). In addition IDL-scripts are permitted.

The executable(s) either need to be pre-installed and accessible from each compute node (preferred), or to be transferred at the start of a job. The latter option is ruled out within the Planck Surveyor project.

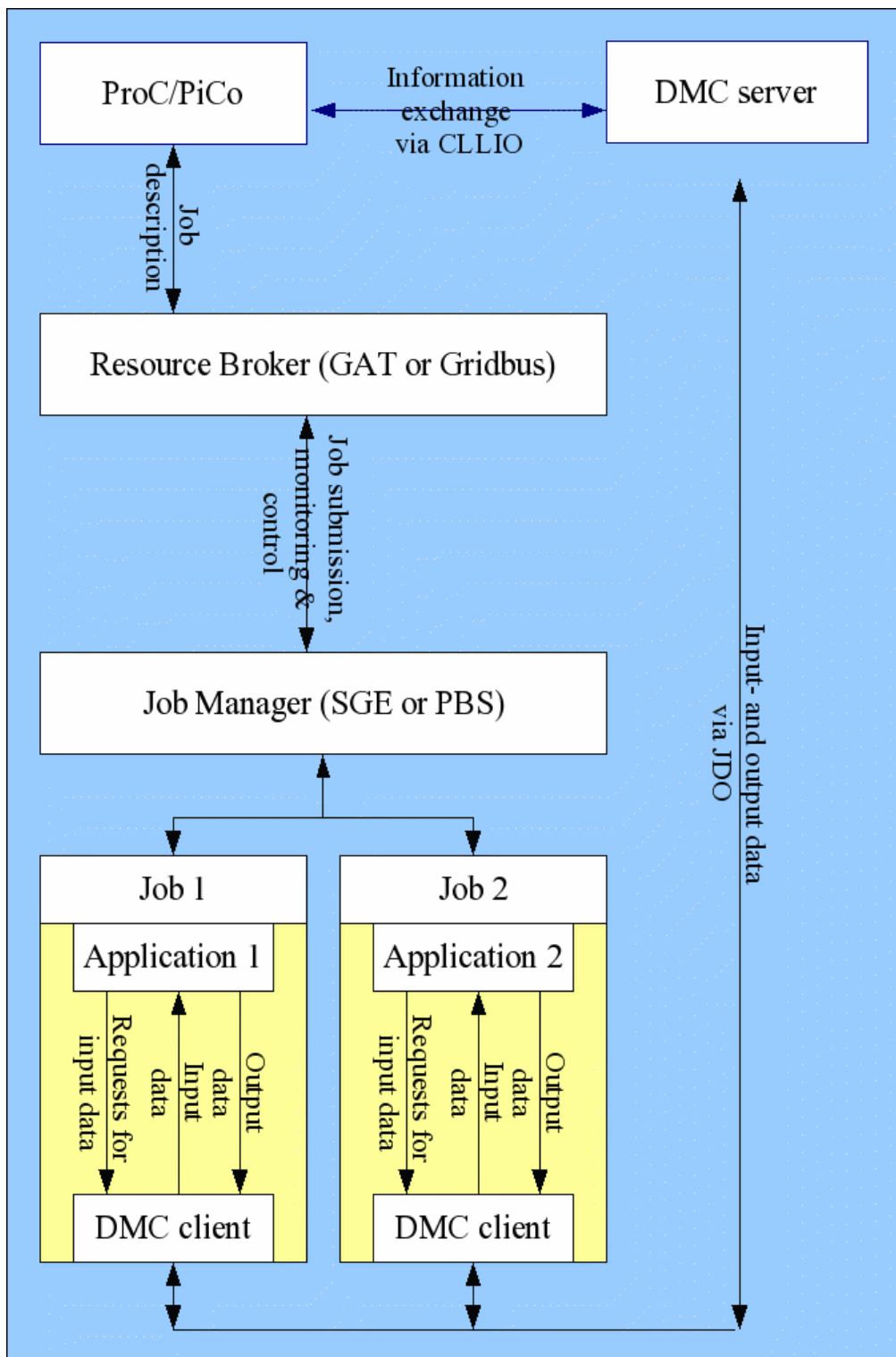


Fig. 1: Architecture of the ProC running on a LAN-grid. The ProC describes a job and passes along the ID of an initialization object. This object is initially fetched from the DMC and further details the input and output data. Subsequently, the job fetches the input data from the DMC server and writes the output back, thus bypassing the grid engine.

Initially it was considered to potentially transfer the sources to the target compute node, and to remotely invoke a compiler on that node. The current scenario for Planck Surveyor data flows, foresees a different operation: executables for certain operating systems are pre-compiled and are part of a software release. In a comprehensive installation process the executables have to be installed in a code repository. It is assumed that each compute node has access to this repository via the LAN.

The module description must contain the *logical* description of the executable which needs to be passed on to the resource broker. The queuing system needs to resolve the full path to the required binary from

- a root path to the local code repository,
- the operating system of the compute node (contained in the resource description of the compute node), and
- the name of the executable (probably augmented by a particular version or version range).

Data management

Transferring the input parameters and input files during job submission is a standard task for a local grid engine. However, in this LAN-grid scenario the input files will not be transferred using this standard mechanism, but be requested directly from the DMC, thus bypassing the grid engine.

Only a few key parameters such as the ID of an initialization object, perhaps a session ID, and a token for DMC authentication need to be included in the job description. Each job itself will use the init object ID for fetching the init object from the DMC, and use its content for subsequent job initialization.

The metadata for a dataset is stored in a central database managed by the DMC server. This database may either also hold the data, or a reference to an externally stored file. The dataset may actually consist of more than one file structured as a whole tree.

At the end of a job, a ProC module will transfer its output (including transient data files) back to the DMC server, where it is stored.

It is assumed that jobs in a LAN-Grid can freely communicate with a DMC server running on the same LAN. Since the jobs are executing on compute nodes interconnected by a LAN, bandwidth and latency for data transfer are probably not very important considerations.

Job control

Apart from job submission, job status information must be relayed back from the grid engine to the ProC. The most important information is the “job finished” message. Without such a message the ProC cannot execute a chained workflow.

Also, it should be possible to relay back error messages, and to potentially abort a hanging process. While the most important functionality of the resource broker layer is job submission, job monitoring and control (e.g. job abort) are also needed as soon as possible.

Schedule requirements

An initial implementation of the ProC on a LAN-grid shall be operational by **mid-December 2005** (this year) in order to allow the usage of the ProC already in the first processing stage of Planck data reduction to which ESA applies the strictest testing requirements 18 months before launch.³

Phase-2: ProC using a WAN-grid

In the second phase the ProC is to be extended allowing the execution of jobs on a wide-area network (WAN). This requires the interfacing of the ProC to global grid middleware (Fig. 2)

Grid middleware

One option consists in using the Globus Toolkit 4. In this case the job management would be channeled through the Globus Resource Allocation Manager (GRAM) included in the Globus Toolkit 4. The Globus Toolkit 4 will be installed at both RZG and LRZ. The LRZ will officially support GT on behalf of the D-GRID integration project.

Another option consists in using the UNICORE grid middleware software, which is in the open domain since 2004. UNICORE will also be available within D-GRID, and we can request its support for AstroGrid-D. UNICORE is officially supported by Forschungszentrum Juelich, and the RZG will be acting as a local know-how proxy.

Ideally, for WAN-gridification we should like to use the same Java API as already employed for the LAN-gridification. However, the increased unreliability of a WAN-grid and limited network bandwidth may require a more complex API. GAT, notably, adapts to UNICORE, and will eventually add an adapter for GTK4.

Data management

In this scenario, the compute nodes are connected to the DMC server via WAN, bandwidth and latency problems occurring during data transfer have to be expected. Ideally, a scheduler with look-ahead capability would be used that could trigger a pre-fetch of data for future processing tasks.

How a job communicates with the DMC needs to be further detailed. In particular the question arises, whether some grid middleware component will be used for data transfer.

Schedule

Work on the gridification of the ProC using a WAN-grid is slated for some time in the year 2006.

³This tight requirement is dictated not by AstroGrid-D schedule considerations, but by Planck-internal deadlines.

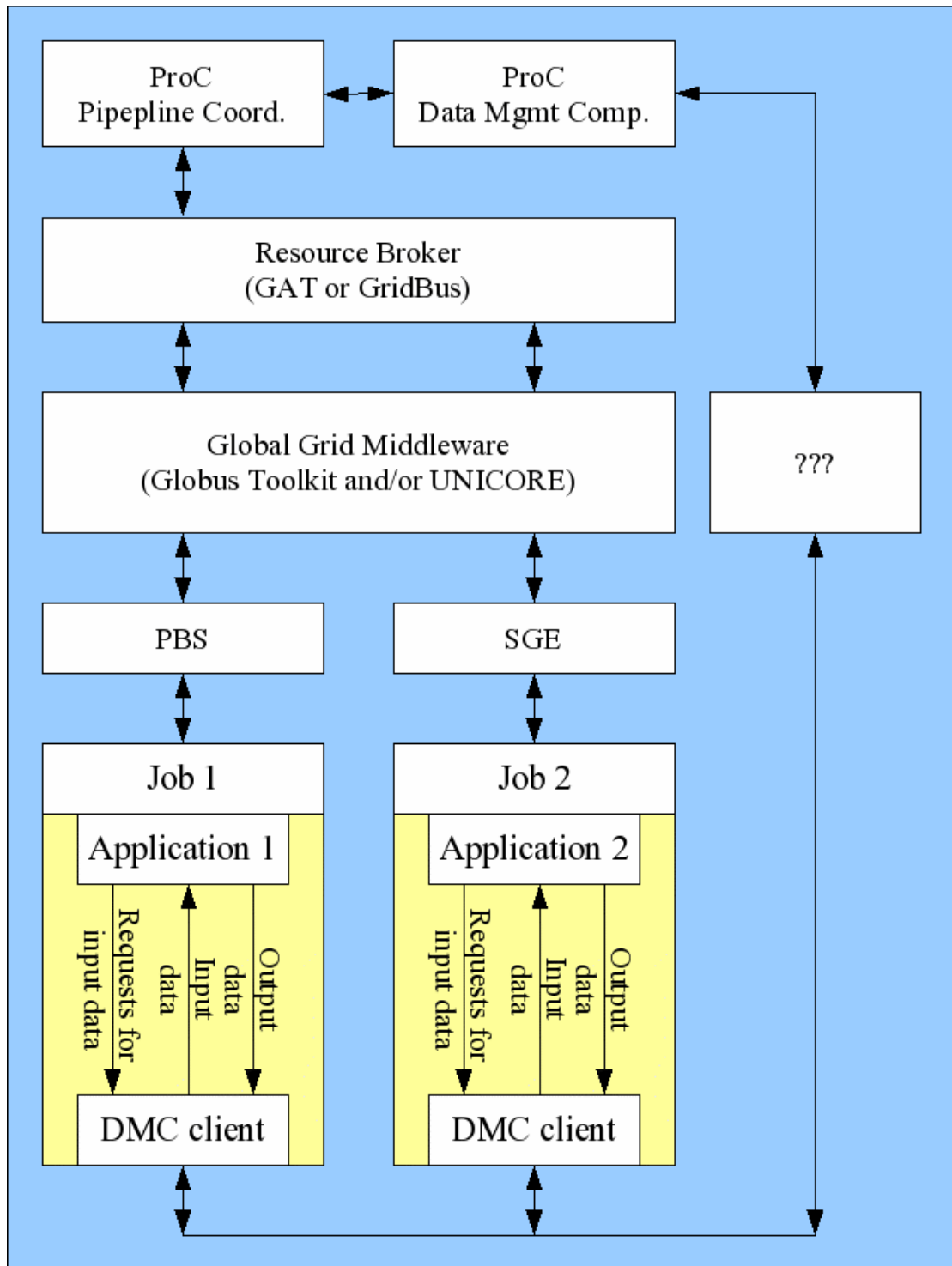


Fig. 2: Architecture of the ProC running on a global grid. The ProC's Pipeline Coordinator submits executable jobs to the resource broker, which passes them, via the global grid middleware layer, to the local job manager (PBS or SGE). The data communication between a job and the DMC needs to be specified in more detail.

Phase-3: ProC executable from a WAN-grid

In the third phase the ProC shall be interfaced in such a way that it can also be started from any authorized node on a global grid (Fig. 3).

This possibly entails a translation of the descriptions of modules, compute resources and jobs into a service and resource description language that can be understood on the grid.

A complication is presented by the fact that the DMC uses a so called “federation layer” for Planck-consortium-internal authentication. It needs to be defined whether the federation layer shall be integrated into a gridified DMC, or bypassed and replaced by a Grid authentication mechanism.

Use case

The primary use case for a ProC that is executable from the Grid is not the processing of Planck Surveyor data, but the post-processing of simulation data upon user request.

The RZG hosts a simulation archive with several tens of Terabytes of data, including the Millenium simulation run produced by the VIRGO consortium. The data is public in principle, but it cannot immediately be used.

In order to make these simulations comparable to observations, the data need to be post-processed. Post-processing cannot be done generically, since the processing steps depend on the scientific question asked. Postprocessing can become complicated requiring up to 100.000 files to be managed.

Another task of the MPA grid-project within AstroGrid-D project consists in opening the simulation data archive to a wider astronomical community.

Since the individual post-processing steps form a workflow, it is natural to use the ProC for workflow definition and execution.

Within the MPA-grid project, predefined template workflows for simulation data post-processing shall be offered. The user shall be enabled to remotely invoke the ProC Pipeline Editor in order to change individual parameters, to modify a given workflow, or to create a new workflow from scratch.

A finished workflow description shall be submitted to the Grid for execution.

Data management

Each simulation data set serving as input to post-processing typically has a size of 10 Gigabyte. 20 to 50 of these data sets need to be processed per user request. Thus 200 Gigabyte to 1 Terabyte of data needs to be shuffled around for a single postprocessing request. Each postprocessing step may require up to a CPU-hour per data set.

Ideally, in order to avoid high network traffic and delays, the data postprocessing would be carried out at RZG. However, it may quite well be that the RZG IBM Regatta machine is less than ideally suited for this data-intensive task.

However, the required I/O-rate will not be a problem for the new SGI Altix 3000 processor supercomputer, to be available in the first half of 2006 at LRZ after its move to Garching. If this machine were used, the data would have to be transferred from RZG to LRZ. It is currently unclear (to us) what bandwidth the network connection between RZG and LRZ will have.

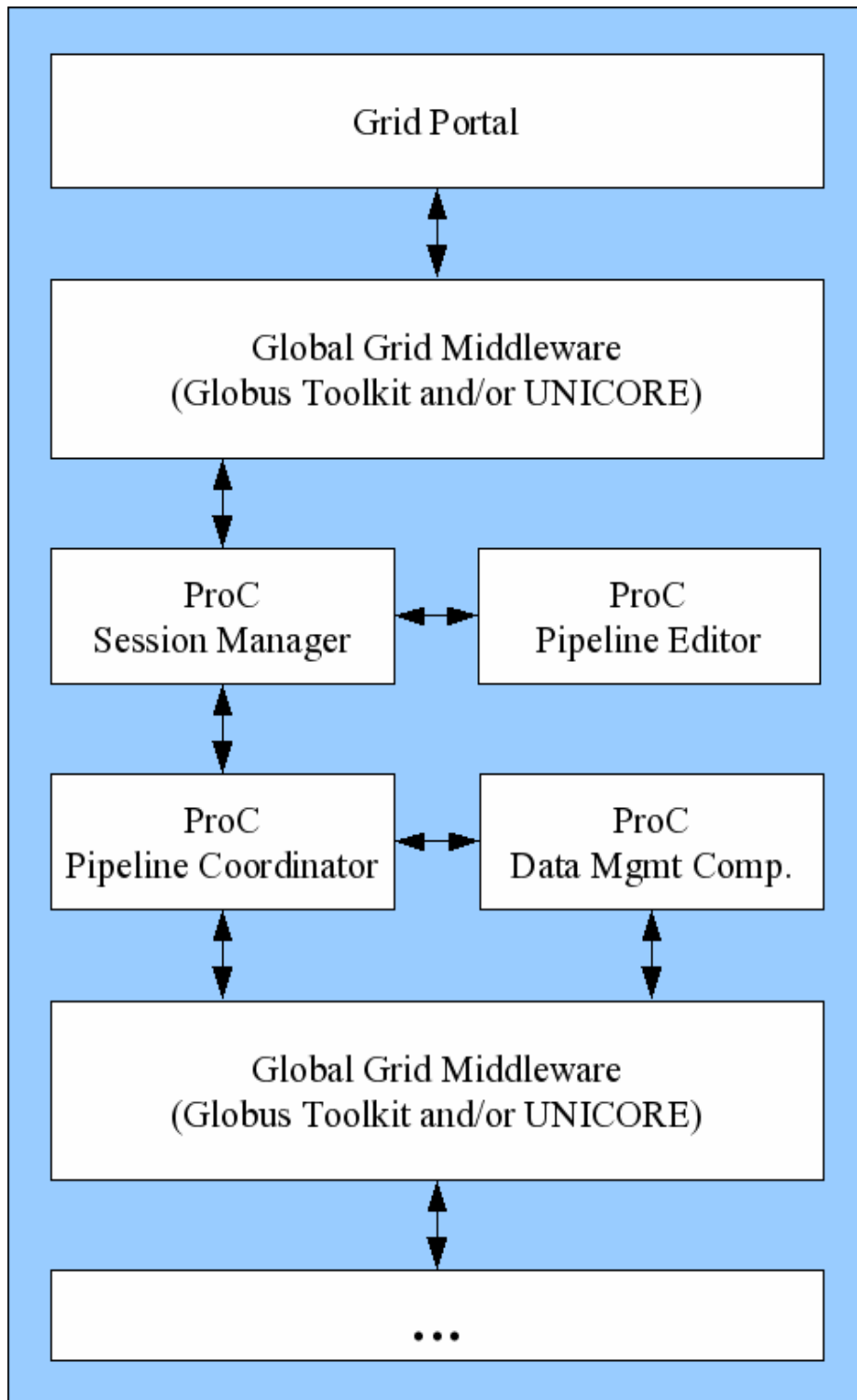


Fig. 3: Simplified architecture of the Process Coordinator workflow engine gridified in such a way that it is usable from the Grid. From a Grid portal, the ProC Session Manager can be started, which in turn invokes the Pipeline Editor. During workflow execution the ProC Pipeline Coordinator submits jobs to the global grid for distributed processing.

Appendix

Glossary

AFS	Andrew File System
API	application programmer interface
BMBF	Bundesministerium fuer Forschung und Technologie
CLLIO	common low-level input-output interface
CPU	central processor unit
DMC	Data Management Component
GAT	Grid Application Toolkit
GRAM	Globus Resource Allocation Manager
GT	Globus Toolkit
HFI	High-Frequency Instrument (on-board the Planck satellite)
IBM	International Business Machines
IDL	Interactive Data Language by Research Systems, Inc.
JDBC	Java database connection
JDO	Java data object
LAN	local-area network
LFI	Low-Frequency Instrument (on-board the Planck satellite)
LRZ	Leibniz Rechenzentrum (Bavarian supercomputer center)
MPA	Max-Planck-Institut fuer Astrophysik, Garching
NFS	network file system
PBS	Portable Batch System (queuing system)
PiCo	Pipeline Coordinator (workflow execution engine of the ProC)
ProC	Planck Process Coordinator (workflow engine)
RZG	Rechenzentrum Garching
SGE	Sun Grid Engine (queuing system)
SGI	Silicon Graphics, Inc.
URL	universal resource locator
WAN	wide-area network

References

Doerl, U., W. Hovest, F. Dannemann, Th. Riller, M. Bartelmann, T. Ensslin, “Planck IDIS Process Coordinator Architectural Design Document”, Issue 2.1, 2005-01-31.

Hovest, W., F. Dannemann, Th. Riller, M. Bartelmann, “A General Process Coordinator – Developed for the Planck Mission”, ADASS XI, Victoria, Canada, 2001.

“ATLAS DAQ Software Development Environment”,
<http://rd13doc.cern.ch/Atlas/DaqSoft/sde/inspect/shall.html>