

D2.2 Prototype Implementation of CGAT

Author(s):	Robert Engel, Thomas Radke
Document Filename:	Gridlab-2-D2.2-0002.CactusGAT
Work package:	WP2 - CactusGAT
Partner(s):	PSNC,MPG,ZIB,MU,VU,CARDIFF
Lead Partner:	MPG
Config ID:	Gridlab-2-D2.2-DRAFT-2
Document classification:	PUBLIC

Abstract: This document describes the plans for the functionality and implementation of the Cactus GAT toolkit, a set of thorns which provide easy access to the Grid Application Toolkit developed by GridLab WP1.



1 Overview

The goal of GridLab WP2 (CGAT) is to enable applications written in the Cactus framework to make use of the Grid-Application-Toolkit-API (GAT-API) for sophisticated Grid related scenarios.

The GAT API provides an abstraction layer which abstracts operations which may need to be Grid-aware in such a way that an application may be developed and run independently of the underlying middleware actually deployed. The *Cactus GAT* (CGAT) toolkit is introduced as a mediating infrastructure layer to glue together the application-specific Cactus Computational Toolkit and the grid-specific GAT-API. It abstracts both the application and Grid interfaces and services used underneath from implementation details.

This document provides details of the functionality and implementation of the CGAT toolkit thorns.

2 CGAT Toolkit

The general concept for the CGAT toolkit and its relation to Cactus and the GAT is illustrated in figure 1. In the illustrated architecture overview, the CGAT toolkit consists of a set of thorns,

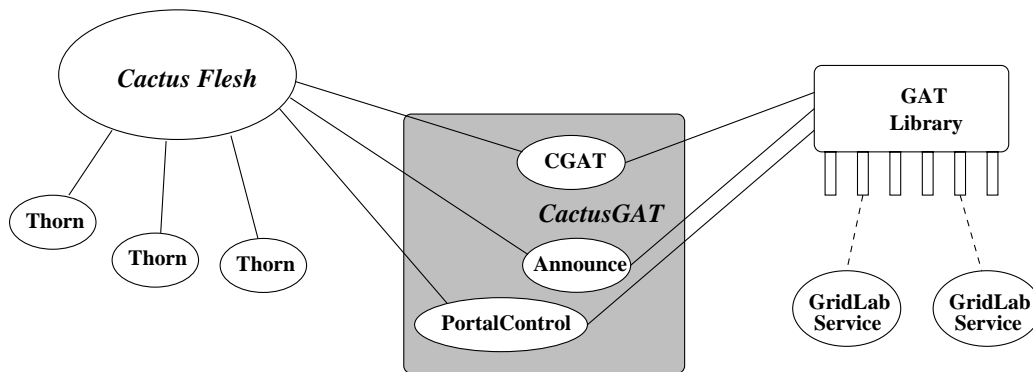


Figure 1: Concept of the CGAT Toolkit

linked to the GAT Engine, which provide services to Cactus applications. The vast majority of Cactus thorns will be unaware of the CGAT or GAT.

The CGAT toolkit supports Cactus-based applications in sophisticated scenarios on the Grid, such as web-based application monitoring and steering, event-triggered application migration, and dynamic spawning of new sub-tasks. The CGAT toolkit provides the key functionalities required by these scenarios through the implementation of currently three thorns which are included in the CGAT arrangement

2.1 Thorn CGAT

Thorn CGAT implements the very basic infrastructure for linking against the GAT and initializing it at startup.

- It provides a script to automatically search for a GAT installation (location of GAT header

files and libraries necessary to compile and link against the GAT) and exports this information to the Cactus make system and other thorns in the CGAT toolkit.

- Before the GAT engine may be used, the GAT libraries must be initialized and adapters to local and remote Grid services must be loaded. This involves the creation of a GATContext object, which may then be used for all GAT operations in this thread. The CGAT thorn creates this context at simulation startup time, and exports functions which allow other CGAT thorns to retrieve it. It also provides parameters which allow the set of adapters and their configuration to be controlled via the Cactus parameter file.
- Both Cactus and the GAT provide checkpointing infrastructure. The CGAT thorn registers that a Cactus simulation is a checkpointable application with the GAT Engine, and therefor provides the necessary callback listeners to allow remote applications to trigger checkpointing. These listeners are configured via the Cactus parameter file to decide which specific Cactus checkpointing mechanism should be used when such a checkpoint request is received.
- A Cactus simulation generates output files in various data formats, using different Cactus I/O methods, and writes them (potentially in parallel) to the local filesystem. GAT provides mechanisms to handle remote files in a transparent manner. The CGAT thorn allows the Cactus I/O thorns and the GAT remote data and file facilities to inter-operate. It uses the GATFile and GATLogicalFile services to advertise all output files created by the Cactus I/O methods and registers them under a unique logical name (GAT job ID plus output directory/filename) with the replica management service. Along with the registration of the application's checkpointing callbacks information about the location of the (potentially multiple distributed) checkpoint files is also registered with the GAT engine so that external Grid services such as the GRMS resource broker (developed in WP-9) are aware of all files necessary to be moved during a migration request.
- When the simulation has finished, thorn CGAT does the necessary calls to unregister checkpoint callbacks, shut down the GAT engine, unload adapters, and finally destroy the GATContext object. This procedure also makes sure that external Grid services are notified about the termination of the current Cactus job.

2.2 Thorn Announce

This thorn registers a running Cactus simulation with the Cactus Simulation Portal (as being developed by WP-4 "Grid Portals").

At startup the simulation will announce itself under a unique URL (hostname of the machine it's running on, plus port number) to the Cactus Portal. With this URL, users can directly connect from the Portal to the ongoing simulation, monitor its current state and steer parameters.

2.3 Thorn PortalControl

The PortalControl thorn provides application control methods for the Cactus Portal through use of the functionality in the Cactus-integrated webserver thorn HTTPD.

3 Integration with other Work Packages

Besides the actual implementation of the CGAT toolkit thorns, much work was spent in making the CGAT functionality easily available to both software developers and application users, and



in the thorough integration of CGAT with Grid services and components developed by other work packages.

3.1 CGAT Application Unit Tests

In close collaboration especially with WP-1 (GAT) and WP-5 (Testbed) a procedure for a fully automated Cactus unit test was designed and implemented on the GridLab testbed. This procedure has been deployed on all testbed machines (currently more than 10 sites in 7 countries) and is run in regular intervals. It includes the following interdependent checks and subtasks:

1. verify that all requirements necessary to configure and build a Cactus application on a give resource are met: select appropriate compilers and system libraries, locate a GAT installation, check system prerequisites such as GNU make, perl, and cvs
2. automatically check out the Cactus source code and the CGAT toolkit thorns from CVS
3. build Cactus executables for two different types of Cactus applications: solving an ordinary wave equation using the standard unigrid driver, and simulating a more complex Black Hole head-on collision using the fixed mesh refinement driver in Cactus. Executables for these applications are configured to be run both as single-processor jobs as well as parallel MPI jobs submitted to cluster resources.
4. All generated Cactus executables are installed in a globally accessible location (defined by `CACTUS_LOCATION`) at each site, along with example parameter files. Developers can use these pre-installed binaries to test their Grid services and components in combination with a real Cactus applications (see integration tests below).

The results for each phase of a unit test are gathered in logfiles and analyzed by a script which finally generates a unit test status webpage displayed on the WP-5 webserver. The unit tests are scheduled to be run on each testbed site once every night, the summary status pages are then updated accordingly.

The deployment of nightly Cactus unit tests was extremely useful to automatize the process of CGAT/GAT software development and installation at different sites. Looking at the status webpages, developers and system administrators can easily identify configuration and/or build problems on individual machines, have them fixed by the responsible work package, and check already the next day if the changes had solved the problems.

3.2 CGAT Application Integration Tests

Based on the Cactus unit tests, advanced scenarios were prototyped to test the integration and runtime interaction of the CGAT application with actual Grid services through the GAT and its service adapters. Together with WP-5 (Testbed), WP-8 (Data Management), WP-9 (Resource Broker), WP-10 (Information Services), and WP-11 (Monitoring), two of these scenarios have already been implemented and deployed globally on the GridLab testbed:

1. Submission of a parallel Cactus job through GRMS: this integration test the GRMS resource broker functionality to submit a parallel MPI application to a given resource. It involves contacting the Grid information services to query for available machines, the interaction with local queuing systems to submit and start a job, and the staging of executables and/or parameter files to selected working directories using remote file transfer services.



2. Externally triggered migration of a Cactus job: in addition to the above subtasks, this integration scenario also uses the monitoring service to watch the simulation's runtime parameters and eventually trigger the generation of an application checkpoint at a simulated contract violation. After the Cactus job has successfully written a checkpoint file and terminated itself, the resource broker now has to find another available resource, migrate the checkpoint to this machine, and restart the simulation there.

The CGAT application job submission test is run regularly every hour at each testbed site. Again, results for this integration test are available via a testbed status webpage.

The CGAT integration tests proved to be very helpful for long-term stress testing of individual Grid services and their interaction. Grid service developers are decoupled from the actual CGAT application implementation details, they can concentrate instead on debugging and optimizing their own Grid components and evaluate their runtime stability under static and/or dynamic conditions.

4 Work in Progress

We are currently working on feature enhancements of existing Grid-enabled Cactus applications (stabilization and optimization of integration tests; enabling direct user interaction with running simulations through the Portal; on-the-fly generation of Cactus executables for user-specified configurations) as well as on the development of new innovative Grid scenarios such as task farming-like parameter surveys or the dynamic spawning of subtasks from within a running Cactus simulation.