



DataGrid

ARCHITECTURAL DESIGN AND EVALUATION CRITERIA WP4- FABRIC MANAGEMENT



Document identifier:	DataGrid-04-D4.2-0119-2_1
Date:	16/11/2001
Work package:	WP4
Partners:	CERN, INFN, KIP, NIKHEF, PPARC, ZIB
Document status	DRAFT
Deliverable identifier:	DataGrid-D4.2

Abstract: This document describes the architecture design of the Fabric Management work package.

Delivery Slip

	Name	Partner	Date	Signature
From	WP4		16/11/2001	
Verified by				
Approved by				

Document Log

Issue	Date	Comment	Author
1_0	10/10/2001	First draft	German Cancio (Editor)
2_0	29/10/2001	Integrated pre-PTB comments from PTB reviewers	German Cancio
2_1	13/11/2001	Integrated post-PTB comments from M. Parsons	German Cancio

Document Change Record

Issue	Item	Reason for Change

Files

Software Products	User files
Word	WP4-architecture.doc

CONTENTS

1. INTRODUCTION.....	5
1.1. OBJECTIVES OF THIS DOCUMENT	5
1.2. APPLICATION AREA	5
1.3. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS	5
1.4. DOCUMENT EVOLUTION PROCEDURE	6
1.5. FUTURE ADDENDA	6
1.6. TERMINOLOGY	7
1.7. AUTHORS AND CONTRIBUTORS	8
2. EXECUTIVE SUMMARY	9
3. OVERVIEW	10
3.1. BACKGROUND.....	10
3.2. CLIENTS OF THE FABRIC MANAGEMENT WORK PACKAGE	11
3.3. GENERAL ARCHITECTURE	11
3.4. ARCHITECTURAL CONTEXT	13
3.5. OPEN ISSUES	15
4. FABRIC MANAGEMENT - DESIGN	16
4.1. OPERATIONS AND ADMINISTRATIVE SCRIPTS	16
4.2. MAINTENANCE TASKS	17
4.3. CONFIGURATION CHANGES AND THEIR DEPLOYMENT	18
4.4. SUBSYSTEM CONTROL FUNCTIONS	20
5. SUBSYSTEM: GRIDIFICATION	22
5.1. INTRODUCTION.....	22
5.2. FUNCTIONALITY	22
5.3. SUBSYSTEM DIAGRAM	23
5.4. CURRENT STATUS	23
6. SUBSYSTEM: RESOURCE MANAGEMENT.....	24
6.1. INTRODUCTION.....	24
6.2. FUNCTIONALITY	24
6.3. SUBSYSTEM DIAGRAM	25
6.4. CURRENT STATUS	26
7. SUBSYSTEM: CONFIGURATION MANAGEMENT.....	27
7.1. INTRODUCTION.....	27
7.2. FUNCTIONALITY	27
7.3. SUBSYSTEM DIAGRAM	28
7.4. CURRENT STATUS	29
8. SUBSYSTEM: INSTALLATION MANAGEMENT	30
8.1. INTRODUCTION.....	30
8.2. FUNCTIONALITY	30
8.3. SUBSYSTEM DIAGRAM	31
8.4. CURRENT STATUS	31
9. SUBSYSTEM: FABRIC MONITORING AND FAULT TOLERANCE.....	32
9.1. INTRODUCTION.....	32
9.2. FUNCTIONALITY	32
9.3. SUBSYSTEM DIAGRAM	35
9.4. CURRENT STATUS	36

10. USE CASES	38
10.1. INTRODUCTION.....	38
10.2. USE CASE: GRID JOB SUBMISSION.....	38
10.3. USE CASE: UPGRADE OF NFS SERVER ON A CLUSTER	39
10.4. USE CASE: FAULT RECOVERY IN CLIENT/SERVER ENVIRONMENTS	41
11. APPENDIX: GRIDIFICATION COMPONENTS	44
11.1. COMPONENT: COMPUTING ELEMENT (CE)	44
11.2. COMPONENT: LOCAL COMMUNITY AUTHORIZATION SERVICE (LCAS)	45
11.3. COMPONENT: LCAS PLUG-IN AUTHORIZATION MODULES	46
11.4. COMPONENT: FLIDS.....	46
11.5. COMPONENT: LCMAPS	47
11.6. COMPONENT: GRIFIS.....	48
11.7. COMPONENT: FABNAT	49
12. APPENDIX: RESOURCE MANAGEMENT COMPONENTS.....	51
12.1. COMPONENT: RMS INFORMATION SYSTEM.....	51
12.2. COMPONENT: REQUEST HANDLER.....	52
12.3. COMPONENT: SCHEDULER	53
12.4. COMPONENT: PROXIES	55
12.5. COMPONENT: PLUGIN FOR RESOURCE AVAILABILITY CHECKS	56
12.6. COMPONENT: INFORMATION PROVIDERS FOR GRIFIS.....	56
13. APPENDIX: CONFIGURATION MANAGEMENT COMPONENTS	58
13.1. COMPONENT: CONFIGURATION DATABASE (CDB).....	58
13.2. COMPONENT: CONFIGURATION CACHE MANAGER (CCM)	59
13.3. COMPONENT: SOFTWARE LIBRARY IMPLEMENTING THE NODE VIEW ACCESS API (NVA API)	59
14. APPENDIX: INSTALLATION MANAGEMENT COMPONENTS.....	61
14.1. COMPONENT: NODE MANAGEMENT AGENT (NMA)	61
14.2. COMPONENT: SOFTWARE PACKAGE (SP)	62
14.3. COMPONENT: SOFTWARE REPOSITORY (SR).....	64
14.4. COMPONENT: BOOTSTRAP SERVICE (BS)	65
14.5. COMPONENT: INFORMATION PROVIDERS FOR GRIFIS.....	67
15. APPENDIX: MONITORING AND FAULT TOLERANCE COMPONENTS.....	68
15.1. BASIC DEFINITIONS	68
15.2. COMPONENT: MONITORING SENSOR AGENT.....	68
15.3. COMPONENT: MONITORING REPOSITORY.....	69
15.4. COMPONENT: MONITORING USER INTERFACE.....	70
15.5. COMPONENT: ACTUATOR DISPATCHER	71
15.6. COMPONENT: MONITORING SENSOR	72
15.7. COMPONENT: FAULT TOLERANCE ACTUATOR	73
15.8. COMPONENT: FAULT TOLERANCE CORRELATION ENGINE.....	73

1. INTRODUCTION

1.1. OBJECTIVES OF THIS DOCUMENT

The main objective of this document is to provide an overview of the architectural design of the Fabric Management Work Package of the DataGrid project. Functionality and interactions between identified subsystems are described. [A1] provides a general description of the overall DataGrid architecture.

1.2. APPLICATION AREA

This document applies to the entire Fabric Management work package.

1.3. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

[A1] The DataGrid Architecture.

G. Cancio, S. M. Fisher, T. Folkes, F. Giacomini, W. Hoschek, B.L. Tierney. Version 2, June 2001.
<http://cern.ch/grid-atf>

Reference documents

[R1] The Anatomy of the Grid.

I. Foster, C. Kesselman, et al. Technical Report, GGF, 2001.

<http://www.globus.org/research/papers/anatomy.pdf>

[R2] Job Description Language How-To.

F. Pacini.

<http://www.infn.it/workload-grid/documents.htm>

[R3] OpenPBS project Homepage.

<http://www.openpbs.org>

[R4] Load Sharing Facility (LSF) Homepage.

<http://www.lsf.com>

[R5] Condor project Homepage.

<http://www.cs.wisc.edu/condor>

[R6] Architecture of the Resource Management System of WP4.

T. Roebnitz, F. Schintke, T. Schuett.

<http://cern.ch/hep-proj-grid-fabric/architecture/rms.pdf>

[R7] Globus CAS – Community Authorisation Service.

<http://www.globus.org/research>

[R8] Node Profile Specification.

<http://cern.ch/hep-proj-grid-fabric-config/documents/np.pdf>

[R9] Cache Manager Protocol Specification.

<http://cern.ch/hep-proj-grid-fabric-config/documents/cmp.html>

[R10] Configuration Distribution Protocol Specification.

<http://cern.ch/hep-proj-grid-fabric-config/documents/cdp.text>

[R11] Node View Access API Specification.

<http://cern.ch/hep-proj-grid-fabric-config/documents/nva/>

[R12] RPM - RedHat Package Manager.

<http://www.rpm.org>

[R13] dpkg - Debian Packaging system.

<http://www.debian.org>

[R14] Solaris pkg. Solaris 7 reference manual.

<http://docs.sun.com>

[R15] A grid monitoring service architecture.

B. Tierney, R. Wolski, R. Aydt and V. Taylor. Technical report, GGF, 2001.

- [R16] Global Grid Forum. <http://www.gridforum.org>
- [R17] The Globus Project. <http://www.globus.org>
- [R18] A Resource Management Architecture for Metacomputing Systems. K. Czajkowski, I. Foster, N. Karonis, et al. <http://www.globus.org/research>
- [R19] PXE- Preboot Execution Environment. <ftp://download.intel.com/ial/wfm/pxespec.pdf>
- [R20] bpbatch homepage. <http://www.bpbatch.org>
- [R21] Collected reference documents for the Configuration Management task. [http:// cern.ch/hep-proj-grid-fabric -config/refs.html](http://cern.ch/hep-proj-grid-fabric-config/refs.html)
- [R22] WP4, report on current Technology. M. Barroso. <http://cern.ch/hep-proj-grid-fabric>
- [R23] ASIS, manage and distribute Application Software in the HEP Community. P. Defert et al. Technical report, CHEP'97. <http://cern.ch/gcancio/paper -chep97 - asis.ps.gz>
- [R24] SUE – Standard Unix Environment. <http://wwwinfo.cern.ch/pdp/ose/sue>
- [R25] Large Scale Linux Configuration with LCFG. P. Anderson et al. Atlanta Linux Showcase, 2000 <http://www.dcs.ed.ac.uk/~paul/publications>
- [R26] Performance and Exception Monitoring project home page. <http://cern.ch/proj-pem/>
- [R27] PEM – Monitoring Agent Subsystem. S. Chapeland. Report on prototyping http://wwwinfo.cern.ch/pdp/monitoring/report_12_06_2001.doc
- [R28] DMTF (Distributed Management Task Force) homepage. <http://www.dmtf.org>

1.4. DOCUMENT EVOLUTION PROCEDURE

The architectural design described in this document represents an early view and is subject to evolution. This Work Package is developing middleware and documentation in a rapidly changing technology field. The Grid and large farm paradigms represent new concepts in computing. Prototypes developed will help fabric administrators and users to get experienced with these technologies. Their feedback will influence the evolution of the design and allow the refinement of their requirements. The architectural design will therefore evolve considerably over the three-year period covered by this Project.

The present document is based on work that has been carried out over the first 6 months of the Project. Even though the document is self-consistent and constitutes a finished Project Deliverable, important additions will be published as Addenda in due course. As advancement has not been uniform across all WP4 tasks, some are described and treated in greater detail than others.

1.5. FUTURE ADDENDA

Future addenda to this document will include:

- More details in the description of subsystem functions and API's. For the moment, function calls and API's are mostly shown schematically.
- More work on security and error recovery.

1.6. TERMINOLOGY

Acronyms

AD	Action Dispatcher
AD	Actuator Dispatcher
API	Application Programming Interface
ATF	DataGrid Architecture Task Force
BS	Bootstrap Service
CAS	Community Authorisation Service
CE	ComputingElement
CERT	X.509 Certificate
CCM	Configuration Cache Manager
CDB	Configuration DataBase
CDP	Configuration Distribution Protocol
CLI	Command Line Interface
CMP	Cache Manager Protocol
DMTF	Distributed Management Task Force
FabNAT	Fabric Network Address Translation service
FLIDS	Fabric-Local Identity Service
FMFT	Fabric Monitoring and Fault Tolerance
FTA	Fault Tolerance Actuator
FTP	File Transfer Protocol
FTDU	Fault Tolerance Correlation Engine
GGF	Global Grid Forum
GMA	Grid Monitoring Architecture
GRAM	Grid Resource Allocation Management
GriFIS	Grid Fabric Information Service
GS	Gridification Subsystem(s)
GUI	Graphical User Interface
HLD	High-Level Description
HTTP	HyperText Transfer Protocol
JDL	Job Description Language
LCAS	Local Centre Authorisation Service
LCMAPS	Local Credential MAPPING Service
LDAP	Light-weight Directory Access Protocol
LLD	Low-Level Description

LRMS	Local Resource Management System
MDS	Globus Meta-computing Directory Service
MS	Monitoring Sensor
MSA	Monitoring Sensor Agent
MR	Monitoring Repository
MUI	Monitoring User Interface
MLD	Machine Level Description
NIS	Network Information Service
NMA	Node Management Agent
RMS	Resource Management Subsystem
SP	Software Package
SR	Software Repository
SSL	Secure Sockets Layer
TFTP	Trivial File Transfer Protocol
WP	Work Package
WP1	Work Package 1 – Workload Management
WP4	Work Package 4 – Fabric Management
XML	eXtensible Markup Language

1.7. AUTHORS AND CONTRIBUTORS

The following persons have contributed directly or indirectly as authors of the present document or by participating otherwise in the elaboration process of the fabric management work package (by partner, and in alphabetic order):

CERN: *Vlado Bahyl, Olof Barring, Maria Barroso, Julian Blake, Germán Cancio, Sylvain Chapeland, Catherine Charbonnier, Lionel Cons, Philippe Defert, Jan van Eldik, Jan Iven, Javier Jaen-Martinez, Mohammad Jaudet, Peter Kelemen, Thorsten Kleinwort, Jaroslaw Polok, Piotr Poznanski, Bernd Panzer-Steindel, Alan Silverman, Tim Smith.* INFN: *Massimo Biasotto, Andrea Chierici, Alberto Crescente, Roberto Ferrari, Enrico Ferri, Enrico Forte, Gaetano Maron, Michele Michelotto, Alessandro De Salvo, Marco Serra.* KIP: *Lord Hess, Volker Lindenstruth, Markus Schulz, Timm Steinbeck, Frank Pister.* NIKHEF: *Kors Bos, Ton Damen, David Groep, Willem Van Leeuwen, Ton Damen, Wim Heubers, Theo vd Akker.* PPARC: *Paul Anderson, Tim Colles, Alexander Holt, Alastair Scobie.* ZIB: *Alexander Reinefeld, Thomas Röblitz, Florian Schintke, Thorsten Schütt.*

2. EXECUTIVE SUMMARY

The objective of the fabric management work package (WP4) is to develop new automated system management techniques that will enable the deployment of very large computing fabrics constructed from mass market components with reduced systems administration and operations costs. The fabric must support an evolutionary model that allows the addition and replacement of components, and the introduction of new technologies while maintaining service. The fabric management must be demonstrated within the project in production use on testbeds that may be large as several thousand nodes, and it must be able to scale to tens of thousands of nodes.

The present document presents an architecture to achieve this objective. It is in general difficult to draw a sharp line between architecture and design. The strategy taken here is to present the salient functionality features of the individual fabric management subsystems and how they are tied together into a homogeneous control interface for the human administrators. It is demonstrated through examples and use-cases how this interface is used to fulfil fabric wide operations and how those operations are co-ordinated with the running of Grid user jobs.

The level of details varies in the descriptions of the subsystems. The descriptions are based on the current understanding of the architecture. The level of detail reflects the amount of experience already gained from existing tools and in some cases also from early prototyping. There has been no particular attempt to hide those differences in the subsystem descriptions.

The rest of the document is structured as follows:

- Chapter 3 gives an overview of the WP4 architecture.
- Chapter 4 deals mainly with the interaction of the different WP4 subsystems for fabric management tasks.
- Chapters 5 to 9 describe the architecture of the WP4 subsystems.
- Chapter 10 describes use-cases for user job and fabric management.

The appendixes (chapters 11 to 15) contain a description of the subsystem components. As explained above, the level of detail varies slightly from one subsystem and component to another.

3. OVERVIEW

The objective of the DataGrid Work Package 4, Fabric Management, can be summarised as follows:

Deliver all the tools necessary to manage computing fabrics providing Grid services on clusters of thousands of nodes.

The target is to provide the *software infrastructure* to manage very large clusters running Grid jobs. The main goal of this chapter is to present first the basic problems in fabric and large cluster management, to give an overview of the proposed architecture, and to place the architecture in context with the other DataGrid middleware work packages.

3.1. BACKGROUND

A typical computer fabric consists of clusters of computing nodes, where the user jobs are run, and a number of infrastructure elements, including:

- Master nodes which co-ordinate computing node clusters for batch and interactive services
- Storage servers such as disk servers (e.g. NFS, RFIIO, GridFTP) and tape infrastructure (tape servers and robotics).
- Installation and software repository servers.
- Information servers (e.g. database and monitoring servers)
- Network infrastructure (switches, DNS servers)
- Miscellaneous servers (time servers, password and credential servers)

Managing a computer fabric with those components not only involves the management of the components individually but also the management of the dependencies between the components and the ordering between operations that have to be performed. This latter complication is very important. For instance, shutting down an NFS server for maintenance involves a prior configuration change on all served clients to use an alternate server. This quite trivial example already exposes:

- The need for modelling the dependencies between fabric components, i.e. the dependency of the NFS clients on the NFS server
- The importance of the order in which the operations have to be performed, in this case that the configuration of the clients has to be changed prior to that of the server

A further complication is that some operations can take a significant amount of time. In the example above, if there are user jobs running on the NFS clients, then the configuration change has to wait until the jobs have finished. This may take days on some nodes whereas it may be immediate on others (without running jobs). Such long time-spans increase the risk for queuing up several operations on some nodes, for instance an OS upgrade is requested for the following day on some of the NFS clients in the example above.

Currently those types of complex operations are mostly performed manually, which is error-prone, and difficult and expensive to manage in large cluster environments. The WP4 middleware allows for a significantly increased automation of those operations while still leaving the overall control and supervision of the operations in the hands of the expert system administrators. It is also important that the automation takes into account the integrity of user jobs. Specifically, an operation should not be allowed to abort a user job nor its run-time environment unless there are very good reasons for it (e.g. emergency due to a security incident).

3.2. CLIENTS OF THE FABRIC MANAGEMENT WORK PACKAGE

The main roles in fabric management referred to throughout this document are the following:

- *Grid users* submit their jobs to the Grid via resource brokers for execution on fabrics that fulfil the job requirements.
- *Local users* are registered and run jobs on a specific fabric without needing Grid credentials.
- *Fabric administrators* are responsible for planning, defining, implementing and configuring fabric services.
- *Fabric operators* run and maintain on a day-to-day basis the services defined by the fabric administrators.

The WP4 architecture addresses the requirements from all four roles.

3.3. GENERAL ARCHITECTURE

The functionality that WP4 is going to provide for computing fabrics can be classified into two main categories:

- User job control and management (Grid and local jobs) on fabric batch and/or interactive CPU services
- Automated system administration of computing fabric elements

The first functionality category listed above is provided by the *Gridification* and *Resource Management* subsystems.

- The *Gridification* subsystem (chapter 5) provides the interface from the Grid to the resources available inside a fabric for batch and interactive CPU services. It provides the interface for job submission/control and resource publication to the GRID services (WP1, WP3). It also provides functionality for local authentication and policy-based authorisation and mapping Grid credentials to local credentials.
- The *Resource Management* subsystem (chapter 6) is a layer on top of the fabric's available cluster batch and interactive services¹. While the WP1 Grid Resource broker manages workload distribution between fabrics, the Resource Management subsystem manages the workload distribution and resource sharing of all batch and interactive services inside a fabric, according to defined policies and user allocations.

The second functionality category listed above, the infrastructure for automating the management of computing fabric elements, is handled by the *Configuration Management*, *Installation Management*, *Fabric Monitoring and Fault Tolerance* subsystems. These subsystems are reserved for system administrators and operators for performing system maintenance.

- The *Configuration Management* subsystem (chapter 7) provides the components to manage and store centrally all fabric configuration information. This includes the configuration of all WP4 subsystems as well as information about the fabric hardware, system and services.
- The *Installation Management* subsystem (chapter 8) handles the initial installation of computing fabric nodes. It also handles software distribution, configuration and maintenance according to information stored in the Configuration Management subsystem.
- The *Fabric Monitoring and Fault Tolerance* subsystem (chapter 9) provides the necessary components for gathering, storing and retrieving performance, functional, setup and

¹ Also known as local resource management systems – LRMS.

environmental data for all fabric elements. It also provides the means to correlate that data and execute corrective actions.

The WP4 subsystems are tied together using a *scripting layer* (chapter 4) for automation of complex fabric-wide system administration operations. The scripting layer allows coordinated execution of user jobs with the execution of administrative tasks on fabric nodes such that the integrity of user jobs is preserved. Template scripts for typical operations will be delivered, which are then adapted and configured by administrators. These scripts are then triggered manually or scheduled for future execution. They can also be executed by the Monitoring and Fault Tolerance subsystem as part of an automated corrective action. The scripting layer allows for tailoring and building a growing system administration knowledge base.

The use cases found in chapter 10, together with their figures, illustrate more in detail the relationship of the different WP4 subsystem components.

3.3.1. Scalability and use of standards

The WP4 subsystems follow a distributed design where the autonomy of individual fabric nodes is preserved as much as possible. The distributed design implies that there are local instances of almost each one of the subsystems. This means that operations are performed locally where possible and hence the scalability is ensured. The scripting layer coordinates collective control operations requiring central steering. De-facto industry standards, like the Globus toolkit software, will be integrated and re-used in the architecture. Standard protocols and formats, like HTTP and XML, are used whenever possible.

3.4. ARCHITECTURAL CONTEXT

Figure 1 shows the layered Grid architecture according to the proposal by Foster and Kesselman in [R1]. The services offered by the subsystems are the same as those defined in the ATF document [A1]. The WP4 subsystems are highlighted.

The WP4 subsystems are located in the *Underlying Grid* and *Fabric* services layers. The ComputingElement (CE) component of the Gridification subsystem is located within the Underlying Grid services layer. This layer represents the basic Grid services and provides the links between the Fabric and the Grid. The Resource Management, Configuration Management, Node Installation Management, and Monitoring/Fault Tolerance subsystems are part of the Grid *Fabric* layer. The services in this bottom fabric layer are not accessible by the Grid, but can be understood as delivering the building blocks for the upper level Grid layers.

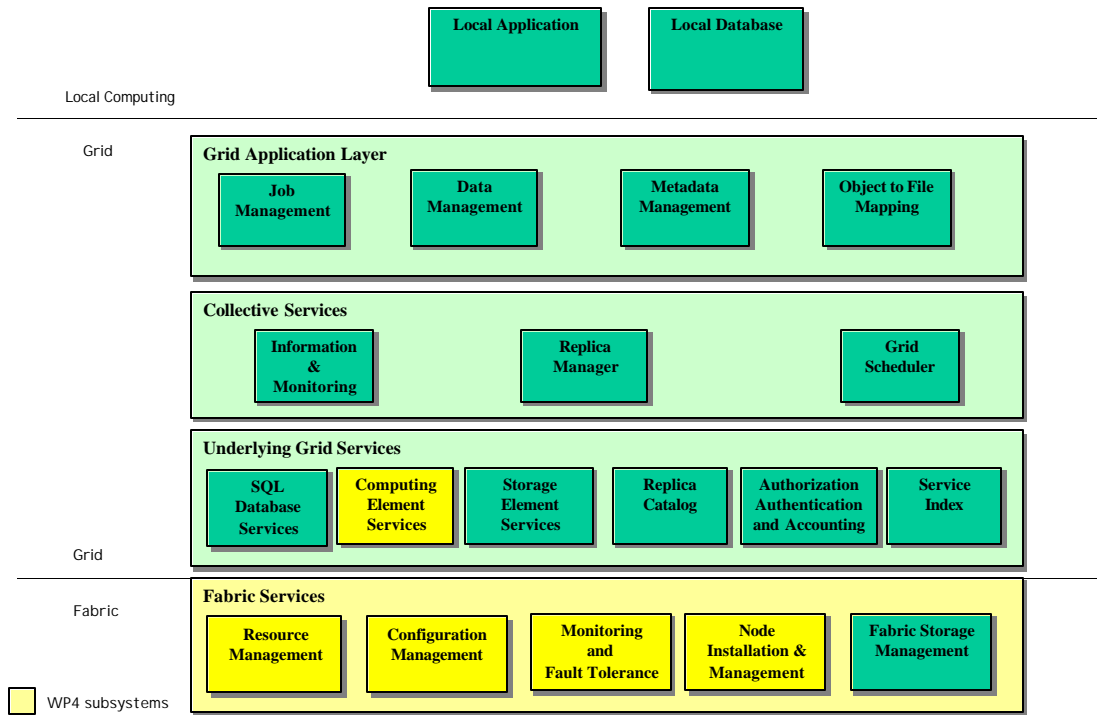


Figure 1: The WP4 subsystems in the Grid layered architecture.

3.4.1. Interaction with other workpackages

Figure 2 depicts the interactions of WP4 with the other middleware work packages. The architecture document from the DataGrid Architecture Task Force (ATF) [A1] describes in more detail the interactions between all the middleware work packages.

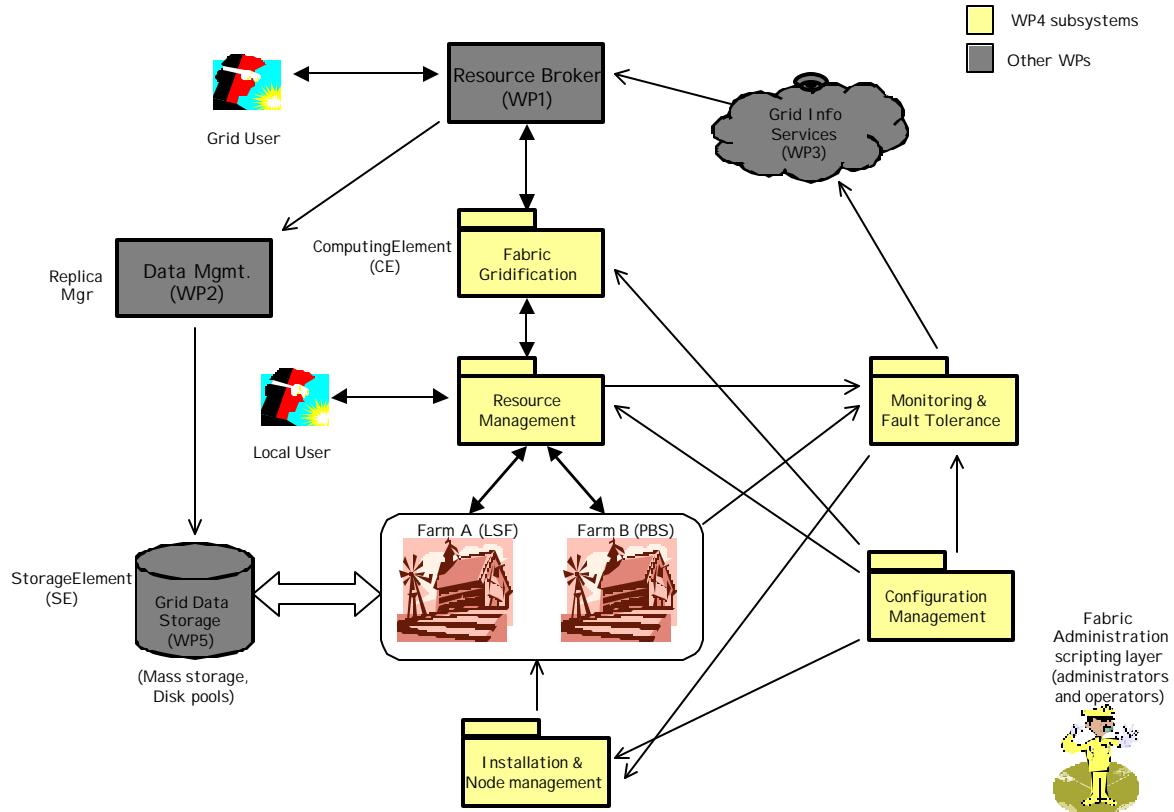


Figure 2: Major interactions between WP4 subsystems and other work packages.

Grid users interact with the Grid Resource Broker provided by the Workload Management work package (WP1) to find and select an appropriate fabric on which to run their jobs. The Grid Scheduler uses matchmaking strategies for comparing the job requirements expressed in JDL (Job Description Language, [R2]) with the available fabrics and the status of their resources.

The Grid Resource Broker also takes into account the location and availability of data input replicas, and if necessary, initiates new replications via the Replica Manager system of WP2. Replicas are copied and stored via the StorageElement (SEs) interface from WP5, which sits on top of mass storage (eg. HPSS and Castor) and disk pool systems. A StorageElement may support multiple file access protocols including RFIO, GridFTP and standard Unix I/O, and can be local or remote to a fabric.

Relevant resource status information coming from the Fabric Monitoring and Fault Tolerance subsystem is made available to the Grid Resource Broker via the Grid Information and Monitoring System from WP3.

Once a fabric has been selected and the data is available on a nearby StorageElement, the Grid Resource Broker submits the job to the ComputingElement (CE) component of the Gridification subsystem for authentication and authorisation. Job execution for Grid and local users is managed by the Resource Management subsystem on the fabric's available batch and/or interactive services. Job control commands (e.g. kill a job) from the Grid Resource Broker are received by the Gridification subsystem and forwarded to the Resource Management subsystem. Updates to the job status (e.g. from queued to running, from running to finished) are made available to the Grid Resource Broker.

3.5. OPEN ISSUES

- *Interactive services*: The full implications of supporting interactive services have yet to be understood. The current WP4 understanding is that interactive services, as requested by the applications, could be provided with very high priority/turnaround queues. However, a good and common understanding of the precise meaning of "interactivity" is needed between the middleware and application work packages.
- *Job priorities and resource reservations*. Support for both reservation mechanisms and priorities have been required by the application work packages. A further clarification is needed of what kinds of priorities are requested, and how they can be made compatible with conflicting reservations.
- *Job checkpointing and job migration* can be vital for optimal resource use. The fabric management could also be improved, since such techniques would allow for quickly removing a node from production for maintenance.
- *Local user job submission* may interfere with Grid user job submission and have an impact on resource availability prediction and planning. Further research is required in this field, as done in [R6].
- *Configuration changes and their deployment* is another research topic, as methods and strategies have to be investigated for how to handle inconsistencies between current and desired configurations, and how to recover and rollback if required.

4. FABRIC MANAGEMENT - DESIGN

WP4 provides a framework that allows for automated fabric-wide management operations. The expert knowledge resides in *administrative scripts* written and maintained by fabric administrators.

The basic ideas behind this design are the following:

- Nodes in a WP4 fabric are highly autonomous. Operations are handled within the node whenever possible.
- Nodes have separated maintenance and production states. This minimises conflicts between user jobs and automated (and/or manual) system interventions.
- A framework architecture allows for separation between the programming of fabric-wide complex administrative operations, and the scheduling and execution of those operations.

In this chapter the syntax is defined as well as the details of the design. The co-ordination between user job and node management is described, and how to assure the scalability of large deployment operations. Finally the interfaces for the fabric management script layer to interact with the low-level WP4 subsystems are presented.

The architecture is based on our current understanding of how to automate large-scale cluster management. The level of detail in the descriptions of the subsystems varies and schematically defined interfaces, such as the high-level configuration language, reflect that substantial research and development is still required. This will be pointed out where appropriate.

4.1. OPERATIONS AND ADMINISTRATIVE SCRIPTS

An *operation* is a consistent and complete change to the state of a computing fabric. Operations include, for example, upgrading a system package on a set of nodes, removing a CPU node, or adding a new disk server. Multiple nodes, and multiple subsystems, may be affected by such an operation. An operation is decomposed in several steps, which have to be executed in a specific order on these subsystems.

The operations are coded in *administrative scripts* that may be written by experienced fabric administrators. Within these scripts, the control flow for performing the operation is coded. These scripts act as a 'glue' and ensure that an operation is executed completely and in the right order. The subsystems keep their independence and internal coherence – the administrative scripting layer only aims at connecting them for building high-level operations. Policies stored in the Configuration Management subsystem can be enforced within these scripts, e.g. monthly reboots of cluster nodes.

An example of an administrative script is the operation to add a new CPU batch node to a farm. This includes declaring the node to the Configuration Management subsystem with the right profile, installing the node with the correct environment according to its profile, and adding the node to the right user job queues in the Resource Management subsystem. Another example of administrative scripts is the operation for obtaining weekly service status and hardware inventory reports.

An administrative script is invoked either:

- Manually by the system administrator from the command line
- Via user-friendly form-based interfaces (e.g. web forms)
- Automatically through some scheduling mechanism (e.g. `cron`)
- Through an actuator launched by the Fabric Monitoring and Fault Tolerance subsystem for automatic recovery

The administrative script calls one or more *control functions* of the WP4 subsystems, such as a query or change request to the Configuration Management subsystem. The scope of the control functions is

the entire fabric, while only a subset of them are meaningful at the node level, e.g. queries to the Configuration Management subsystem for the local node configuration.

Administrative scripts are implemented using standard programming languages (like Perl and Java).

4.2. MAINTENANCE TASKS

Control function calls to the Node Management Agent component (NMA) of the Installation Management subsystem (section 14.1) are also known as *maintenance tasks*. A maintenance task is *non-intrusive* if it can be executed without interfering with user jobs (for example: cleanup of log files), or *intrusive* otherwise (for example: kernel version upgrade, node reboot, node re-installations). It is up to the administrator to mark a maintenance task as non-intrusive or intrusive inside the administrative script.

Maintenance tasks are not sent directly to each node's NMA but using the *Actuator Dispatcher* component (see 15.5), which queues up maintenance tasks per node for execution.

It is necessary to differentiate user jobs from maintenance tasks because

- Maintenance tasks may imply actions like upgrades, reboots, re-installations, which are not suitable for a network connection-based resource management system.
- Maintenance tasks may change the node's configuration completely while execution of user jobs does not affect a node's set-up.
- Maintenance tasks can run on all nodes on a fabric (computing and infrastructure nodes), while user jobs run only on computing (batch/interactive) nodes.

Maintenance tasks are initiated by administrative scripts launched through any of the means described above.

4.2.1. Maintenance Tasks and User Jobs

From the system administration point of view, there are two basic node states:

- *Production*: the node is running user services (eg. NFS server) or executing user jobs (eg. CPU farm node). In Production state, non-intrusive maintenance tasks can be executed in parallel to user jobs.
- *Maintenance*: no user services or jobs running. In Maintenance state, both non-intrusive and intrusive maintenance tasks can be executed.
- The state of a node is set via a control function call from an administrative script.

The node is set to Maintenance state when it is convenient to perform maintenance operations. Prior to changing the state of a node to Maintenance, it is necessary to assure that the node has become idle (no user services or jobs running) and will remain so. For instance, on CPU nodes user jobs have to be finished or terminated and the submission of new jobs prohibited. On server nodes, the service has to be shut down after its clients have been reconfigured. However, in exceptional emergency situations (e.g. urgent security-related upgrades or operations), the policy may be that a node is forced to Maintenance state without having gone idle first.

The node is set to Production state when it is ready to receive user jobs or service requests. Following the change to Production state, the node should be re-enabled to continue its functions.

Manual administration interventions (e.g. swapping a physical system component, such as a disk) should be performed only when the node is in Maintenance.

When the execution of an intrusive maintenance task (e.g. kernel upgrade) is stale because the node is in Production state all subsequent maintenance tasks, no matter whether they are intrusive or non-

intrusive, are queued up behind. This is to avoid conflicts because the order between maintenance tasks can be significant as they may depend on each other. When the state of the node changes to Maintenance, the queued-up maintenance tasks are executed in the order in which they were queued.

4.2.2. (Advance) Reservations for Maintenance Tasks on CPU Nodes

Support for immediate or future *reservations* of CPU resources may be necessary for user jobs as well as for maintenance tasks. On CPU nodes running user jobs, a reservation scheme ensures that no conflicts will arise when scheduling user jobs and maintenance tasks on these. Future, or *advance*, reservations for maintenance tasks are required for regular and periodic system interventions (such as monthly reboots for cleaning up zombie processes) or for planned and scheduled upgrades (e.g. upgrading system libraries).

Reservations for maintenance tasks require interaction with the Resource Management subsystem. This is achieved by requesting the Resource Management subsystem to disable those nodes from a given time. The Resource Management subsystem, in collaboration with the underlying batch system, makes sure that eventual running or scheduled user jobs on those nodes are finished in time or, if possible, moved or rescheduled to other similar nodes that match the jobs' requirements. The node is then taken out from the user job queues and made available for maintenance.

A main difference between reservations for maintenance tasks and reservations for user jobs is that the former target *specific* nodes in a farm, while the latter just require *any* node that matches the requirements expressed in the job description. Thus the Resource Management subsystem is allowed to move user reservations around the *compatible* nodes that match the job requirements. If a system administrator wishes to upgrade or reinstall a specific set of nodes, user jobs can be scheduled (and/or migrated if possible) to compatible nodes, as long as the pool of available compatible nodes is large enough. If conflicts still arise with user jobs or user advance reservations, e.g. because there are not enough available compatible nodes, the Resource Management subsystem notifies back to the calling administrative script, which then has to act accordingly.

Because a maintenance task may result in a complete reconfiguration of a node (e.g. a reinstallation with a new OS release), the node will have to be re-declared to the Resource Management subsystem with its new configuration once the task has finished.

Having an advance reservation for a maintenance task does not imply its successful execution, as it cannot be validated beforehand. An administrative script relying on an advanced reservation has to be carefully programmed, taking into account that the fabric environment may change between the time it is scheduled and when it is executed.

4.3. CONFIGURATION CHANGES AND THEIR DEPLOYMENT

The Configuration Management subsystem allows for a hierarchical structuring of configuration information. Having tens of thousands of different fabric elements with similar configurations it is natural to structure the information in hierarchical layers (e.g. fabric, service and node level settings) such that commonalities can be shared through inheritance. This is to avoid information duplication and thus to make the configuration information more manageable.

A configuration *change* is an update to the configuration information in the Configuration Management subsystem. A single high-level configuration change in the Configuration Management subsystem may affect the specific configuration of many nodes. The *deployment* of the change is when the software installed on the nodes is updated against the new configuration via a maintenance task. While the registering of configuration changes may be instantaneous, its deployment on some nodes can take time because the nodes are in Production and hence not ready to update their configurations. Therefore the deployment may well span a long time period during which some parts of the nodes have been updated while others are not. The Monitoring and Fault Tolerance subsystem is informed and will not consider this as an inconsistency.

Here, further research is required on configuration changes and their deployment. The current proposal is a first step in this direction, but configuration change validation, the handling inconsistencies between current and desired configurations, and the need for rollback and recovery is yet to be investigated.

4.3.1. Operation Types

From the operational point of view, there are three different types of operations:

- *Configuration change operations*: A fabric administrator responsible for the configuration of one or several software packages (also known as a *Product Maintainer*) writes and/or executes scripts for adding, updating or deleting these packages in the Configuration Management subsystem.
- *Deployment operations*: A fabric administrator responsible for a set of machines (also known as a *Service Manager*) decides at what time configuration changes have to be *applied* to these machines. This may involve stopping/suspending/migrating running services such as user jobs on these nodes during the deployment, and changing the state of the node from Production to Maintenance.
- *Maintenance operations*: Routine maintenance tasks, which do not imply a configuration change. For example, monthly reboots of nodes for cleanup of zombie processes. This may involve stopping/suspending/migrating services and changing the state of the node from Production to Maintenance.

4.3.2. Partitioned Deployment

Separating configuration *changes* from their *deployment* is necessary because product maintainers and service managers are usually different roles: product maintainers are responsible for defining the supported software packages and service managers are responsible for running services. Moreover this separation facilitates very complex and lengthy deployments because it allows for *partitioning*, i.e. the configuration change is deployed stepwise on the affected nodes. Carrying out the deployment of high-level configuration changes, such as the default OS version or a system update on a service, has to be partitioned because:

- There are user jobs running on farm nodes that have to be finished or migrated out first.
- Guaranteed resource availability (e.g. the number of available CPU's) has to be ensured per service.
- The maintenance timeslots for each service are different.
- There are Infrastructure limitations, e.g. not all nodes can be upgraded at the same time due to limited network and installation server capacities.

The database structure of the Configuration Management subsystem should reflect permanent logical criteria, and not deployment views, which are conditioned by environment and may therefore vary from time to time. For instance, deploying a change of a configuration value for a service that is comprised of hundreds of nodes may in practice be arbitrarily partitioned so that the first ten nodes are upgraded, then the next ten nodes, etc. The deployment strategy is selected for optimal operational convenience and should not affect the way the configuration information is structured.

4.3.3. Deployment example

The Package Maintainer responsible for AFS decides to upgrade the AFS client software package. This requires running an administrative script containing the basic steps:

- Commit the fabric level configuration change of the default AFS package version.
- Submit a maintenance task (flagged as intrusive) to the NMA component via the Actuator Dispatcher (see 15.5) of all the affected nodes to update their state to the new configuration.

The script will first call the Configuration Management subsystem to load the new configuration. The configuration management system updates the node profiles and returns a list of nodes affected by the configuration change. Thereafter the script calls the Actuator Dispatcher to schedule the basic action that will update the state of the affected nodes to match their new configuration.

Now, a service manager wants to deploy the new configuration to a large service. To avoid a complete service interruption the deployment is partitioned. The script to be run for each deployment partition includes the following steps:

- For each node, tell the Resource Management subsystem to drain/migrate user jobs and disable the node.
- For each node, ask the NMA to set the node into Maintenance state.
- For each node, wait for the Actuator Dispatcher to finish execution of all pending maintenance tasks (which includes the AFS client update).
- For each node, ask the NMA to set the node into Production state.
- For each node, ask the Resource Management subsystem to re-enable the node with its new configuration.

The script runs in parallel for all nodes belonging to the deployment partition.

4.4. SUBSYSTEM CONTROL FUNCTIONS

WP4 provides a set of base libraries for accessing each subsystem via the control functions and a library of methods common to all subsystems. Template administrative scripts for the most usual fabric management operations are also provided.

The Resource Management, Configuration Management, Installation Management and Fabric Monitoring and Fault Tolerance subsystems offer access to their control functions via APIs.

These APIs do not replace the way the subsystems configure themselves according to their configurations stored in the Configuration Management subsystem. They are only used for status queries and actions that do not imply a reconfiguration. Reconfigurations are done via the Configuration Management subsystem.

For each subsystem, there is a base library on top of the control function API, which provides higher-level services, for example, atomicity and idempotence, handling of multiple requests, serialising/locking, etc.

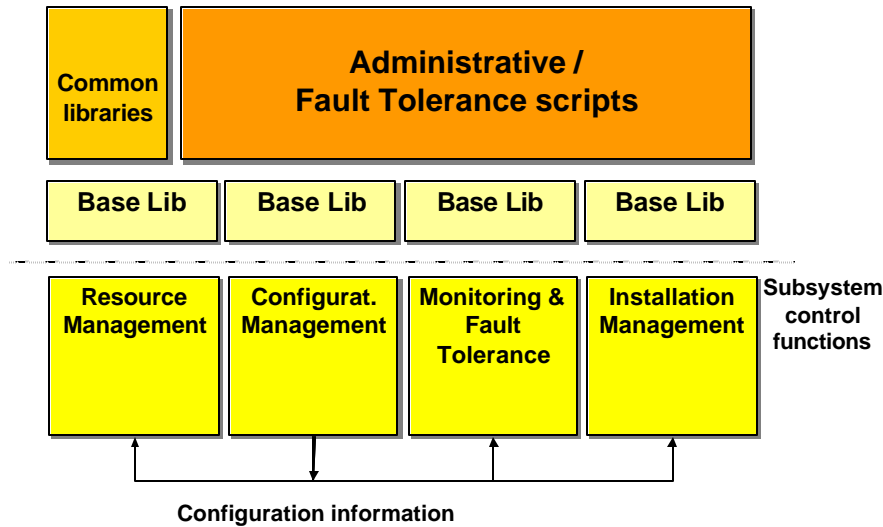


Figure 3: subsystem control functions and libraries

Above the base libraries there is a common library layer containing methods, which affect multiple subsystems. The common libraries can be used by administrative scripts. These methods include support for parallel processing (e.g. parallel execution of the same operation on multiple nodes).

Additional libraries, such as for locking, advanced error recovery strategies or transaction support could later on be built on top of the WP4-provided libraries.

The administrative scripts sit on the top level. A set of template administrative scripts is provided that together with the libraries will be extended and refined as experience is gained.

The most important control functions per subsystem are:

- Resource Management: Disable and enable CPU nodes, get availability schedules for nodes
- Installation Management: Dispatch maintenance tasks, set the node's state, verify configuration set-up. Manage the Installation servers.
- Configuration Management: Queries on configuration values, get list of nodes affected by a configuration change.
- Fabric Monitoring and Fault Tolerance: Queries on monitoring information on nodes, metrics, times.

5. SUBSYSTEM: GRIDIFICATION

5.1. INTRODUCTION

The *Gridification* subsystem (GS) interfaces the local fabric to other grid middleware components. It provides on one hand, the mechanisms for grid-wide services (eg. job control and submission, resource reservations) to access the local fabric services, and on the other, the means to publish fabric configuration and status information to the Grid. At least logically, all interaction from the 'outside' grid with a computer centre is mediated by services provided by components belonging to this subsystem.

The GS does not mediate between services that are local to the fabric. However, since a consistent security infrastructure is an integral part of the *gridification* of a fabric, the GS may provide security components that have intra-fabric functionality, although not conceptually part of the gridification of a fabric.

The GS co-operates closely with the Resource Management subsystem for Grid job submission and management handling. It retrieves its static configuration values from the Configuration Management subsystem and relies on the Installation Management subsystem for its deployment. Relevant information is made available to the Monitoring and Fault Tolerance subsystem for later usage (e.g. auditing).

5.2. FUNCTIONALITY

The Gridification subsystem is composed of five basic components:

- **CE**, ComputingElement: to mediate the request (eg. job execution, resource reservation) received from any grid entity (such as the Grid Scheduler from WPI) to the Resource Management System. This component is the only one that can be accessed from outside the fabric.
- **LCAS**, local credential and authorisation service: to provide local authorisation for requests made to the fabric by grid services.
- **FLIDS**, An automated local certifying entity that can sign certificate requests according to a predefined policy list.
- **LCMAPS**, local credential mapping service: to provide all local credentials needed for jobs allowed into the fabric.
- **GriFIS**, grid fabric information service: to supply aggregate or abstracted information content about the local fabric to the Grid Information and Monitoring service.
- **FabNAT**, fabric NAT gateway: to provide a mechanism to support connections from individual farm nodes to locations outside the fabric, for example for MPI parallel programs, for interactive work or for graphics output.

The GS components provide to the Monitoring and Fault Tolerance subsystem auditing information generated by the components. This auditing information is to be logged and retained.

The components of the Gridification subsystem are detailed out in chapter 11.

5.3. SUBSYSTEM DIAGRAM

Figure 4 is a schematic view of the Gridification components in relation to the Resource Management subsystem, the Configuration Management subsystem, the Grid Scheduler (WP1) and the Grid Information and Monitoring System (IMS, from WP3).

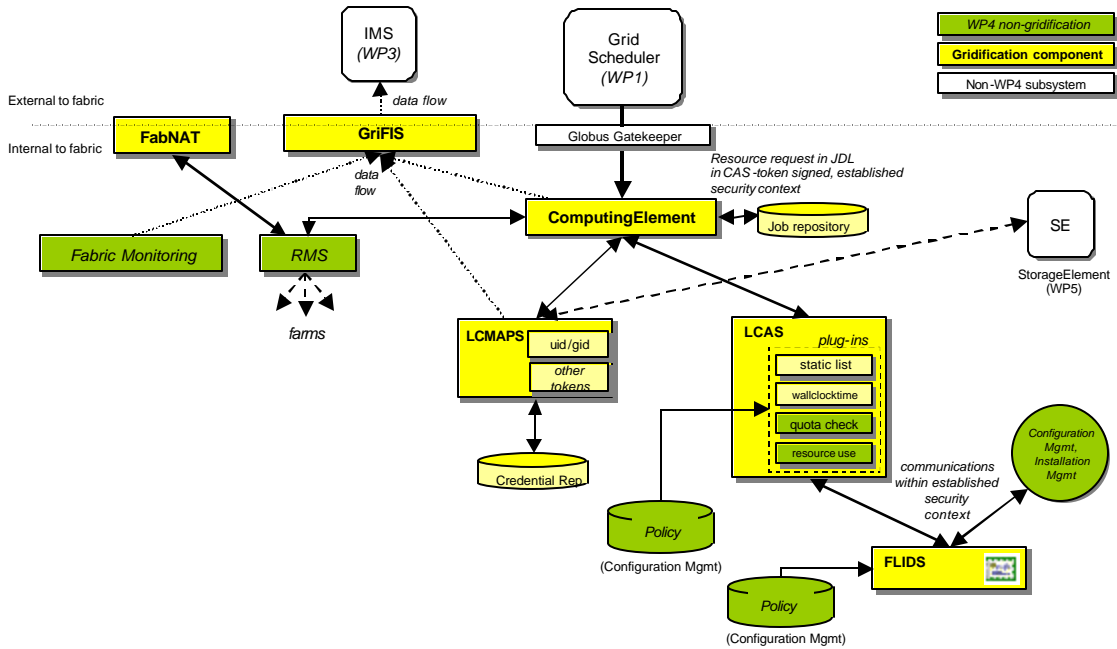


Figure 4: Gridification components within WP4.

5.4. CURRENT STATUS

For most of the gridification subsystem, basic functionality is already provided by the Globus project [R17] implementation (core ComputingElement functionality is covered by the *Job manager* [R18], a GriFIS prototype is covered by the current Globus MDS implementation, and the delivery of information providers for Testbed 1 has shown the viability of such a component). For the LCAS and LCMAPS components, design work has to be done to extend up on the current GSI implementation. The FLIDS design is largely based on existing tools available as part of regular PKI implementations. The work on FabNAT is experimental and considerable research in this area is still required, since the underlying network and QoS infrastructure is still being either designed or developed.

6. SUBSYSTEM: RESOURCE MANAGEMENT

6.1. INTRODUCTION

A typical DataGrid fabric contains one or more clusters. Job submissions to these clusters may be managed by different cluster batch systems, e.g. PBS [R3], LSF [R4], and Condor [R5]. The main task of the Resource Management subsystem (RMS) is to maintain control over the fabric's farm resources and to ensure the efficient scheduling and execution of user jobs and their co-ordination with maintenance tasks. It sits on top of the different cluster batch systems that a fabric offers, and provides transparent access to different cluster batch systems, e.g. job submission and information retrieval. It also enhances their capabilities with advance reservation and co-allocation if necessary. The RMS also offers support for load balancing between the cluster batch systems available in the fabric.

Because of the huge number of resources to manage, the RMS has to meet some requirements to achieve its general goals:

- *Scalability:* Applied mechanisms for scheduling, co-allocation and information provision must scale to tens of thousands of nodes.
- *Automation:* To ensure a high quality of provided services and to reduce cost of ownership administration procedures, e.g. software installation/upgrading or fault recovery, must be highly automated.
- *Extensibility:* Because of the intended long-term use of the developed system, it is necessary that the system may be adapted and/or extended easily in order to support future technologies.

6.2. FUNCTIONALITY

The tasks of the RMS are the handling of jobs and the provision of information about local resources and jobs. This includes:

- Adding/deleting resources to/from the pool of managed resources based on decisions made by the monitoring and fault tolerance subsystem, or by administrators.
- Handling resource requests originating from WP1 (Workload Management) via the Gridification subsystem.
- Handling resource requests from local users of a fabric.
- Scheduling user (Grid and local) jobs.
- Enhanced scheduling strategies like *First Come First Serve (FCFS)*, *Backfill*, *Shortest Job First*, *Longest Job First*, *Deadline Scheduling* or *Advance Reservation*.
- Support for resource reservations and co-allocations.
- Adaptive scheduling on resource failure.
- Performing accounting.

This is reflected in the architecture of the resource management system as shown in Figure 5. In this figure, all components in dotted lines are provided by the RMS. The main RMS components are:

- **RMS Information System**: manages job and resource information needed by the RMS.
- **Request Handler**: verifies, validates and manages incoming job requests.

- **Scheduler:** assigns resources to job requests.
- **Proxies:** interface to cluster batch systems like PBS and LSF.

The RMS also provides a component to perform authorisation checks, e.g. a plug-in for **resource availability checks** deployed by the LCAS (see 11.2). It also delivers **Information Providers** for publication of fabric information to the Grid.

The Job Description Language (JDL) based on Condor ClassAds [R2] is used to describe jobs, e.g. the name of the script or binary executable, input and output data, and resource requirements.

The components of the Resource Management subsystem are detailed out in chapter 12.

6.2.1. Interactions with other Fabric Management subsystems

The RMS closely interacts with other WP4 subsystems:

6.2.1.1. Configuration subsystem

The RMS is built on top of cluster batch systems, which can be configured in two different ways. First, a cluster batch system may be configured *directly*, i.e. as if the RMS did not exist. Second, a cluster batch system may be configured *indirectly* by using the Configuration Management of WP4, i.e. all settings are stored within the Configuration Management subsystem and then used to configure the cluster batch system. The RMS supports both possibilities.

The Configuration Management subsystem stores the static configuration information of the RMS (not only the information regarding the cluster batch systems). This information may be accessed either directly via the Configuration Management subsystem or indirectly via the RMS Information System.

6.2.1.2. Gridification subsystem

The RMS provides modules (plug-ins) that perform accounting and quota checks and dynamic checking for resource availability. These modules are deployed by the LCAS. All necessary credentials are obtained from the LCMAPS. The RMS should not schedule or start a job before having received all necessary credentials. The LCMAPS will be contacted if a job has been scheduled, started and finished. The RMS accepts job requests from the CE and sends job results back to it. Ports or port ranges used by the FabNAT to provide a method for streaming connections have to be considered as resources by the RMS.

6.3. SUBSYSTEM DIAGRAM

Figure 5 shows the architecture of the Resource Management subsystem and its relation with the other WP4 subsystems.

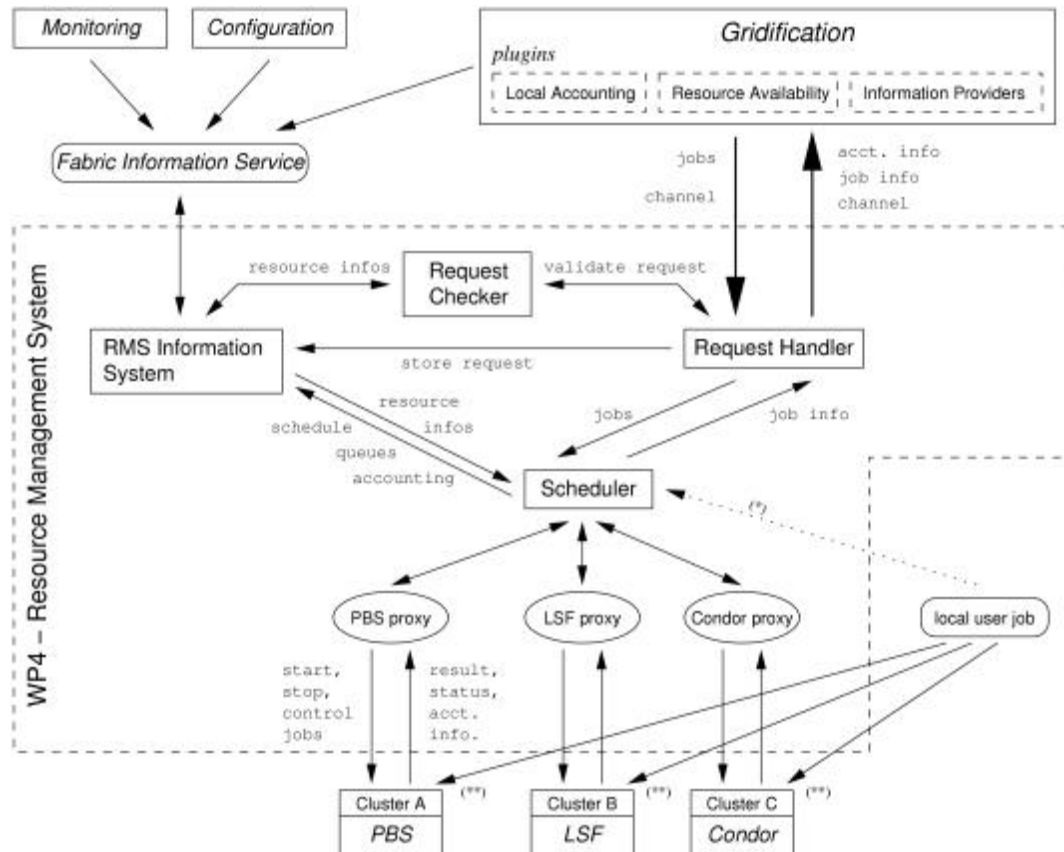


Figure 5: Architecture of the Resource Management subsystem of WP4. (*): Submission of local user jobs via the RMS scheduler. (**): Submission of local user jobs directly to the batch systems.

6.4. CURRENT STATUS

Research efforts are being carried out mostly on scheduling, e.g. scheduling strategies (see [R6]). Investigation has been done on CCS, Globus, cluster batch systems like PBS, LSF, Condor, and the Maui-scheduler. While the architectural design has been done, a detailed design of the components will be developed until the end of 2001. Implementation will be started at the beginning of 2002.

7. SUBSYSTEM: CONFIGURATION MANAGEMENT

7.1. INTRODUCTION

The Configuration Management (CM) subsystem provides a framework to access configuration information. It consists of a central database, a set of protocols and libraries implementing APIs to store and retrieve information. A language to express configuration will also be provided.

7.2. FUNCTIONALITY

The CM subsystem allows components to store, and retrieve *configuration information*.

Configuration information is any piece of information that is needed in order to statically configure a component. It does not include dynamic information that changes (e.g. the contents of a database hosted on a computing node) and information generated by the machine itself - such as system load.

The configuration information can be represented in a *tree structure* (like a UNIX file system or the Windows registry) made of *elements* that can be either leaves called *properties* (similar to files) or interior nodes called *resources* (similar to directories). For instance:

```
/hardware/disks/1/dev = /dev/hda  
/hardware/disks/1/size = 4200
```

could represent a part of the configuration information with the resource `/hardware/disks/1` representing the first disk and the property `/hardware/disks/1/size` representing its size.

The information to be stored is defined and provided by the subsystems/components that use the Configuration Management subsystem.

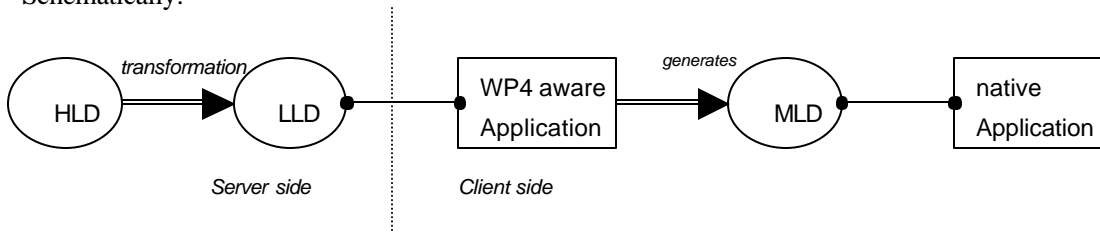
The configuration information is stored in the *configuration database* (CDB) that is the central component holding configuration information for a given set of machines (e.g. for a computer centre or for a whole site). It provides different *views* on the stored information, optimised for the different access patterns of the programs requesting configuration information. Views that have been identified so far:

- The *high-level description* (HLD) is the view optimised for high-level operations such as configuration management of a large number of nodes (eg. site or group configuration, farm or service configuration, hardware type configuration).
- The *node view* or *low-level description* (LLD) is the view optimised for normal node configuration operations: it offers a scalable, read-only interface containing only the configuration information relevant to the node requesting it.

The LLD can be compiled or derived out of the HLD. In section 13.1, an example of LLD is presented.

The LLD is accessed by subsystems/components running on the node for their own use. The information in the LLD can be transformed by these into configuration information as understood by the operating system and applications, the *machine level description* (MLD). Examples of MLDs are the `/etc/sendmail.cf` configuration file for sendmail, or `/etc/inetd.conf` for inetd.

Schematically:



The Configuration Management subsystem contains the following components (components that have been identified so far):

- **Configuration Database (CDB)**
- **Configuration Cache Manager (CCM)**
- **Library implementing Node View Access API (NVA API)**

Figure 6 depicts the diagram of the Configuration Management subsystem. The Client node has the access to LLD called node view information. Applications running on the node access their configuration information via the NVA API [R11]. The NVA API communicates with the CCM using the Cache Manager Protocol (CMP) [R9]. The NVA API gives transparent access to the LLD.

Communication between CDB and CCM is done using the Configuration Distribution Protocol (CDP) [R10], which is based on HTTP.

The components of the Configuration Management subsystem are detailed out in chapter 13.

7.2.1. Profile Specification

Configuration information is stored in the form of profiles. There are two main profile types: node profiles, which contain the configuration of specific nodes, and high-level profiles, which contain HLDs.

The node profile specification is based on XML and it is described in Node Profile Specification [R8]. The structure of high-level profiles and their transformation into low-level or node profiles is still to be defined.

7.3. SUBSYSTEM DIAGRAM

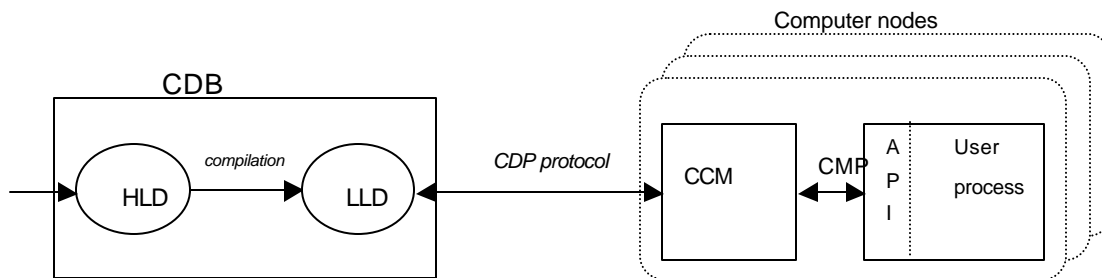


Figure 6: Diagram of the Configuration Management subsystem.

7.4. CURRENT STATUS

Effort has been invested for the survey and evaluation of existing standards, technologies and solutions, like DMTF [R28], XML, SNMP, LDAP, LCFG [R25] and others [R21]. Research is going on in the area of high-level description of configuration information, including a survey of existing models. The CCM and the Node View Access API have been prototyped and deployed on TestBed 1.

8. SUBSYSTEM: INSTALLATION MANAGEMENT

8.1. INTRODUCTION

The *Installation Management subsystem* (IMS) provides the means to install and manage all software running on the nodes in a computing fabric. This includes e.g. CPU nodes for user job execution, infrastructure servers (like HTTP, database or password servers), data servers (disk and tape servers). Personal workstations and PC's can also be managed by the IMS. The IMS allows for managing the installation and upgrade of the operating system and applications, configuring system parameters and applying site policies.

8.2. FUNCTIONALITY

The IMS handles on one hand, the automated bootstrap installation and reinstallation of fabric nodes, and on the other hand, the software distribution and management on nodes according to profiles stored in the Configuration Management subsystem.

The Installation Management subsystem contains the following components:

- **NMA**, Node Management Agent: An agent that runs on all nodes and which manages the installation, upgrade and removal of software packages (**SP's**).
- **SR**, Software Repository: Central fabric store for software packages (**SP's**).
- **SP**, Software Packages: Bundled software applications, modules, libraries, etc. to be deployed by the NMA on nodes.
- **BS**, Bootstrap Service: Service for initial installation of computer nodes.
- **Information Providers**: for publishing information about installed software to the Grid via the GriFIS (see 11.6).

The NMA is the core component. It runs on every node that is part of a WP4 managed fabric. It fetches the node configuration from the Configuration Management subsystem and gets the software packages (SP) to install from appropriate SR servers. It then calls the appropriate methods on the SP's to install and configure them. The NMA is used for the installation and management of all software on a node, including system components, middleware components and end-user applications. The Actuator Dispatcher (AD) component from the Monitoring and Fault Tolerance subsystem controls the execution of the NMA (see 15.5).

The initial installation of a node is performed using the BS, which can store system boot images for multiple operating system versions and configurations.

The components of the Installation Management subsystem are detailed out in chapter 14.

The Installation Management subsystem components provide monitoring, log and auditing information to the Monitoring and Fault Tolerance subsystem. The NMA can be used as a monitoring sensor for verifying the correct installation and state of the node.

Operations that involve the Installation Management subsystem are co-ordinated within the fabric administrative scripting layer. For this purpose, the NMA (via the Actuator Dispatcher), the SR and BS components offer control functions that are called via their API's by administrative scripts. In this way, the execution of fabric -wide system and application installation and upgrade operations can be orchestrated with the other WP4 subsystems, in particular the Resource Management subsystem. This is important, e.g. for minimising the impact on running and scheduled user jobs during cluster -wide re-installations and upgrades.

8.3. SUBSYSTEM DIAGRAM

Figure 7 is a schematic deployment view of the Installation Management subsystem components in relation to the administrative scripting layer, the Configuration Management subsystem, the Actuator Dispatcher (AD) and the Monitoring Sensor Agent (MSA) components from the Monitoring and Fault Tolerance subsystem.

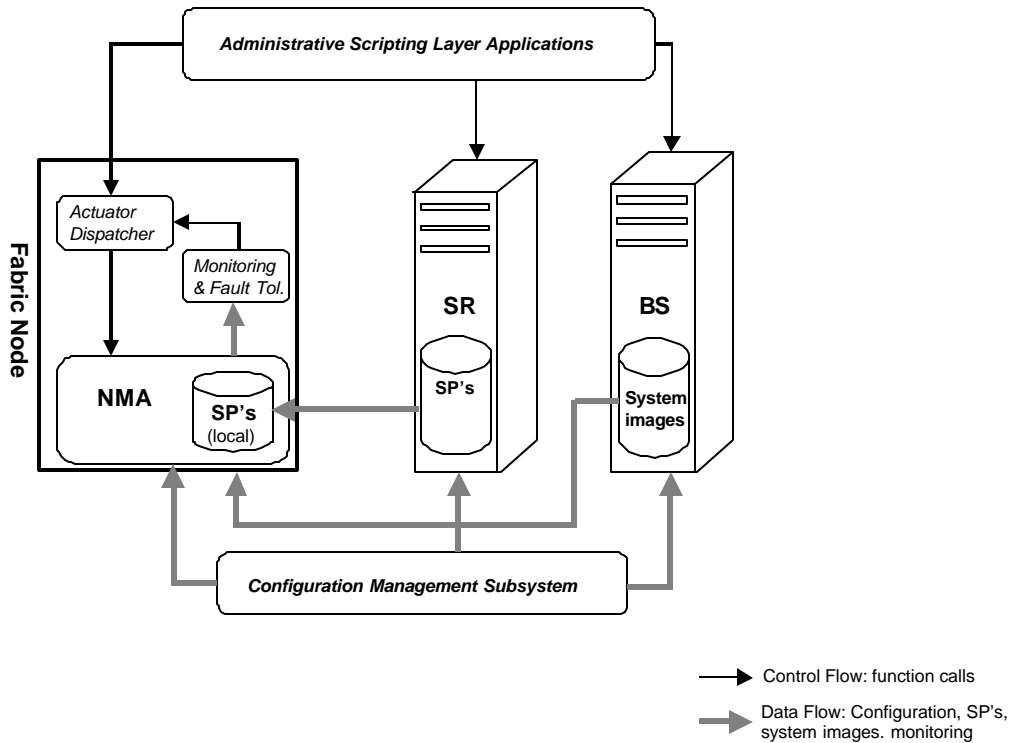


Figure 7: The Installation Management subsystem components in relationship with other WP4 subsystems.²

8.4. CURRENT STATUS

Investigation has been carried out on existing system management solutions [R22], including ASIS [R23], SUE [R24], and LCFG [R25]. The overall architecture design is done, but the component design is due for end of 2001. For TestBed 1, a prototype version of the Bootstrap Service for image-based installations has been deployed. Also, an interim solution for package-based installations, based on LCFG, has been made available for the installation of middleware packages on TestBed 1.

² Other subsystems like the RMS are not depicted in this picture.

9. SUBSYSTEM: FABRIC MONITORING AND FAULT TOLERANCE

9.1. INTRODUCTION

The *Fabric Monitoring and Fault Tolerance* (FMFT) subsystem provides the framework for the monitoring of performance and functional and environmental changes for all resources contained in a fabric. The framework contains a global distributed repository for all monitoring measurements and a well-defined interface to plug-in data analysis routines (*Correlation Engines*) that regularly check that measurements are within configured limits and trigger alarms or automatic recovery actions in case they are not. The FMFT subsystem consists of two parts:

- The monitoring framework for gathering, transporting, storing and accessing monitoring information
- A basic set of monitoring sensors, fault tolerance correlation engines and the recovery actuators

The monitoring system should hold all relevant dynamic information (e.g. CPU load and temperature, disk contention, network status) for proper functioning of the fabric, very much like the Configuration Management subsystem holds all static configuration information. Duplication of the gathering of data should be avoided wherever possible. When developing other fabric components requiring the gathering of dynamic information it should first be checked if there is already a monitoring sensor collecting the same information. If not, the choice of writing such a sensor and using the monitoring framework for transporting the information should be considered. The advantage of using the monitoring is to get a history record of changes and their timestamps, which in turn will allow for detailed tracing of problems.

The monitoring measurements are stored in such a way as to allow for efficient retrieval with a triplet key (*node, metrics, time*). For instance, a Correlation Engine could once per minute request all measurements of the 'uptime' metric for all nodes in cluster 'A' and notify the operator if the value is below one minute for more than 10% of the cluster nodes. The data in the measurement repository is not structured according to any assumption about the fabric layout. The physical layout of how nodes make up clusters and services etc. is configuration information and therefore maintained by the Configuration Management subsystem. This means that measuring collective metrics such as the status of a service or cluster of machines implies combined queries to both the Configuration Management subsystem and the Monitoring Repository.

Finally, the impact of running the monitoring and fault tolerance components on the monitored resources must be under control of the monitoring system itself, limiting the resource utilisation of the controlled sensors.

9.2. FUNCTIONALITY

The monitoring framework part of the FMFT subsystem consists of:

- **Monitoring Sensor Agent (MSA)**. The MSA is responsible for calling the monitoring sensors, receiving the measurement data from the sensors and assuring that the data is forwarded to the Measurement Repository.
- **A Measurement Repository (MR)**. The MSAs insert the monitoring measurements into the MR where the information is stored together with a timestamp. The MR consists of a client API and a global repository server. The client API provides methods for inserting data in the repository and a metric-subscription/notification mechanism for clients to subscribe to metrics and be notified every time those metrics have been measured. The latest measurements are cached in persistent local storage (disk), which ensures autonomy for the Fault Tolerance components that are local to

the node in case the network is unreachable. The MR client API also includes query methods that are used by the Monitoring User Interface and the Fault Tolerance correlation engines. Those methods can also be used by other WP4 subsystems or administrative scripts to access monitoring data.

- **Monitoring User Interface (MUI).** The MUI provides an easy-to-use graphical interface to the measurement repository. It automatically queries the Configuration Management subsystem for high-level configuration information and presents the user with comprehensive views of the monitoring information, for instance health and status displays for entire services rather than individual nodes. To facilitate problem-tracing, the MUI also provides access to the node level information in case the user requests this.
- **Monitoring Sensor (MS).** An MS is an implementation of the *MonitoringSensor* interface that performs the measurements of one or several metrics. The MS is typically driven by rules stored in the Configuration Management subsystem. A given MS implementation can thus be used for several similar metrics, e.g. a single “daemon dead” sensor can be used to monitor the running of several daemons. The MS provides the plug-in layer for any data producer to insert its data into the monitoring system. Other WP4 subsystem components provide MS implementations, for instance the Software Package (SP) component of the Installation Management subsystem provides sensors for checking the SP installation. External information producers, e.g. collectors of application monitoring data, can also provide MS implementations. The sampling is normally triggered by the MSA but the API also allows the MS to trigger asynchronous samplings.

The fault tolerance part of the FMFT subsystem consists of:

- **Fault Tolerance Correlation engine (FTCE).** The FTCE is the active correlation engine of the FMFT subsystem. The FTCE runs as a daemon process on all nodes and should be implemented to be robust to most system component failures. The FTCE processes measurements of one or several metrics stored in the MR to determine if something has gone wrong or is on its way to go wrong on the the system and if so, determine what recovery actions are needed, and call the Actuator Dispatcher to launch the those actions. The FTCE implements the *MonitoringSensor* interface and is sampled by the MSA as a normal sensor. The output metrics values contains normally a Boolean that reflects if any fault tolerance actuators were launched, and if so, the identifiers of the actuators and their return status. The FTCE processing for a given metric is triggered either through a periodic sampling request from the MSA or through the metric-subscription/notification mechanism provided by the MR.
- **Actuator Dispatcher (AD).** The AD is used by the FTCE to dispatch fault tolerance actuators. It consists of a client API and an agent that controls all actuators on a local system. The AD agent does not maintain any permanent channel to the requestor. Instead the client API returns a unique handle for each dispatch request and is able to return the status of any dispatched actuator given the unique handle. The completion status of a dispatched actuator can be retrieved asynchronously. The running of the actuators is serialized so that only at most one actuator can run at any given time. The received requests are queued for FIFO scheduling. Certain dispatcher requests, e.g. immediate shutdown due to hardware failure, are allowed to bypass the normal queue.
- **Fault Tolerance Actuator (FTA).** An FTA is an implementation of the *FaultToleranceActuator* interface that executes automatic recovery actions. The FTA is typically driven by rules stored in the Configuration Management subsystem. A given FTA implementation can thus be used for several similar recovery actions, e.g. a single “daemon restart” FTA can be used to call a *restart* method in the Software Package (SP) class of all software packages that are installed on the node. The FTA is dispatched by the AD agent. Since the FTA may trigger a reboot of the system, there



is no open channel between the FTA and the AD agent and the return status must be stored in permanent local storage. For the same reasons the FTA notifies the AD agent when the return status is available.

The components of the Monitoring and Fault Tolerance subsystem are detailed out in chapter 15.

9.3. SUBSYSTEM DIAGRAM

The figure below shows a schematic logical view of the Fabric Monitoring and Fault Tolerance components and their dependencies. Not shown in the picture is the dependency on other WP4 subsystems but basically all monitoring and fault tolerance components get their configurations from the Configuration Management subsystem. The fault tolerance correlation engine also uses methods of the Installation Management and Resource Management subsystems.

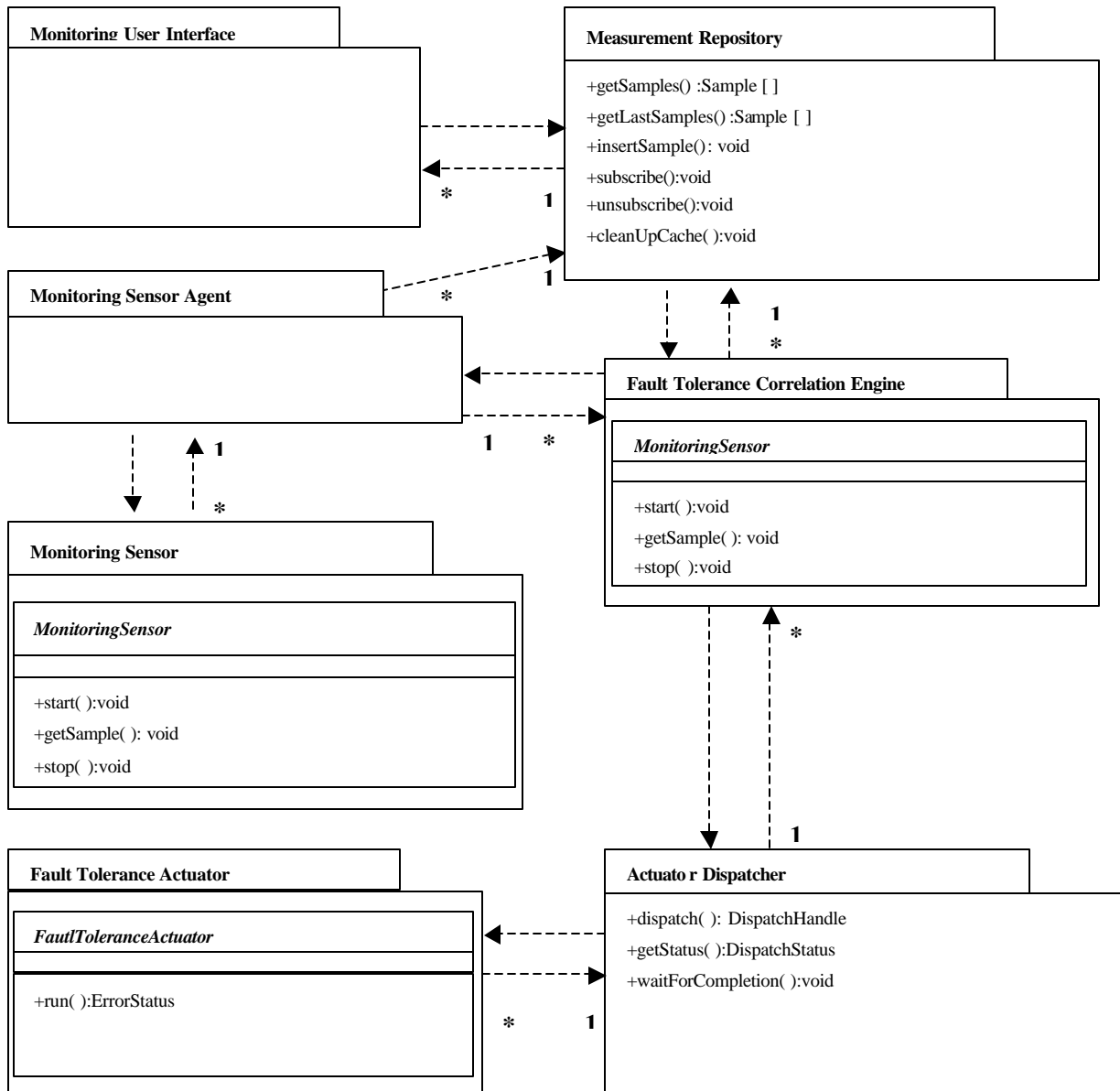


Figure 8: Logical view of the fabric monitoring and fault tolerance components. Formal parameters are not shown in the method prototypes. The relation cardinality is schematically indicated (e.g. '1 .. *' means 'one' to 'zero or more').

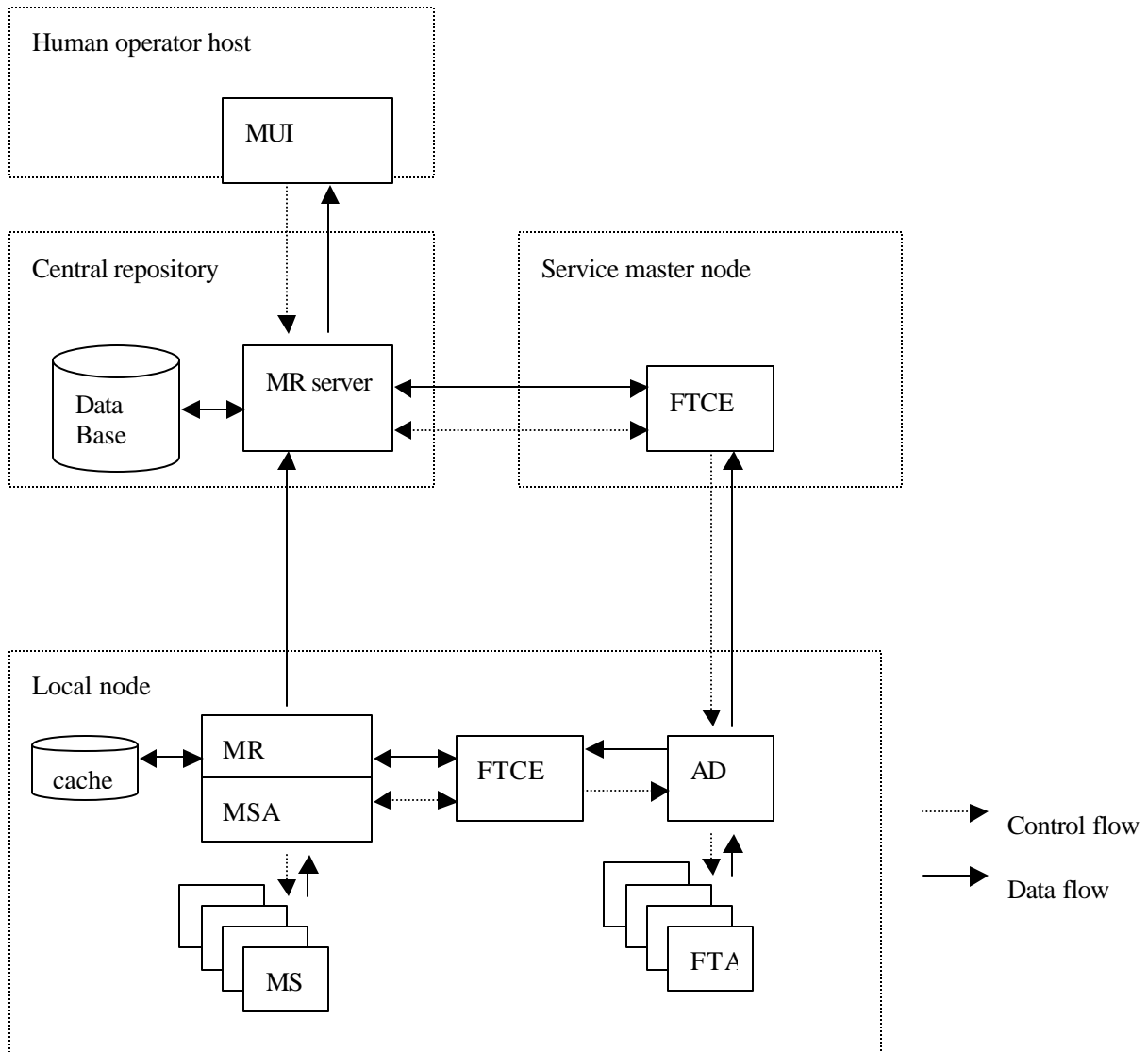


Figure 9: Deployment view of the fabric monitoring and fault tolerance components.

Figure 9 shows the deployment view of the fabric monitoring and fault tolerance components together with the information flow. As can be seen the MR part managing the cache on the local node is contained in the same process as the MSA. The FTCE runs both on locally and centrally. The local FTCE handles all actions, which can be decided locally. The central FTCE runs on, for instance, a cluster master, and it handles actions, which have to be correlated between several nodes (e.g. all CPU nodes in the cluster).

9.4. CURRENT STATUS

Much of the monitoring work is based on the requirements and design inherited from a previous project, Performance and Exception Monitoring (PEM) [R26]. The PEM project delivered an early prototype, which has been tested and evaluated (see [R27]). That report also includes a description of a simple framework for gathering and storing monitoring data. This latter framework has been used to develop sensors and has also provided the basis for the interface definitions. For the component design, key issues that have to be addressed are: MR database design and performance, safe transport of measurement data to the MR. The new design should also include the MUI. Several MS components have been developed. Apart from the interface to the MSA those components can be re-used within a new framework. Further research is needed to address the scalability of the MR database.

10. USE CASES

10.1. INTRODUCTION

In this chapter, a collection of use-cases is presented. The number of available use-cases is still small. They will be refined and extended as the architecture progresses.

10.2. USE CASE: GRID JOB SUBMISSION

Summary: A Grid user wants to execute a simple job on a fabric using the Grid Resource Broker from WPI. Figure 10 depicts the use-case.

Actors involved: Grid user

Subsystems and components directly involved:

- Gridification subsystem (chapters 5 and 11):
 - ComputingElement (CE) (11.1), LCAS (11.2), LCMAPS (11.5)
- Resource Management subsystem (chapters 6 and 12):
 - Request Handler (12.2), RMS Information System (12.1), RMS Scheduler (12.3), Proxy (12.4)

Pre-conditions: The Grid user has a valid credential (certificate). The job description is available, expressed in JDL. The Resource Broker has already selected a Computing Element. The required executable(s) and input files are already in place (installed locally on the CE cluster nodes and available on an accessible StorageElement, respectively).

Event flow:

1. The CE is contacted (via the Globus Gatekeeper) by a Grid Resource Broker, using the SubmitJob() interface, which requires a JDL job description and a credential (certificate).
2. The CE creates a jobID, and updates its repository with a new attribute set containing the jobID, the JDL description, and the presented credential.
3. The CE calls the LCAS via the get_fabric_authorisation() interface.
4. The LCAS calls all the registered authorisation modules in sequence, with the JDL and the credential as input. If a module rejects the request, the LCAS returns an error to the CE, and the latter returns an error to the Grid Resource Broker, stating that local authorisation has failed.
5. The CE passes the JDL and the credential to the LCMAPS component for obtaining a leaseID for the required local credential(s).
6. The CE calls the Request Handler, passing the JDL and the leaseID as arguments.
7. The Request Handler stores and verifies the request. If the request is accepted, the local credential(s) are retrieved from the LCMAPS.
8. The Request Handler contacts the RMS Scheduler for preparing the request, putting the job into schedule, and submitting the job to the selected cluster batch system via its proxy.
9. When the job is reported finished by the cluster batch system, the Scheduler informs the Request Handler with the job return status (possibly including an error message), which is then returned to the ComputingElement.
10. The CE will inform LCMAPS that the job has finished. The ComputingElement reports to the Resource Broker the job result.

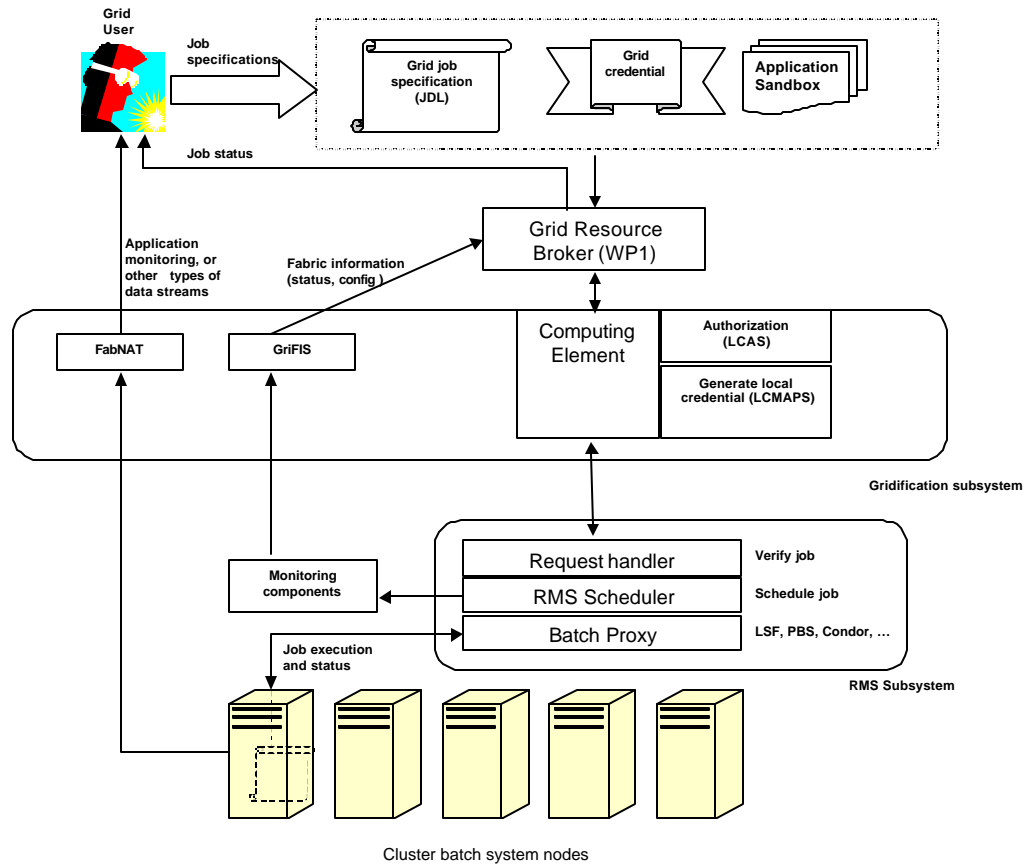


Figure 10: Grid user job submission use-case.

10.3. USE CASE: UPGRADE OF NFS SERVER ON A CLUSTER

Summary: The default kernel version of all NFS server machines of a given type is to be upgraded by the Product Maintainer A. This affects server S1, which is accessed by the nodes of a cluster P. This cluster P, which is managed by Service Manager B, is currently running production jobs. As a kernel version upgrade requires a reboot of the NFS server, this implies a disruption of the service. For this, the clients of server S1 will be pointed to the server S2, which is a fail-over replica of S1. Figure 11 depicts this use-case.

Actors involved: Product Maintainer A, Service Manager B (4.3.1)

Subsystems and components directly involved:

- Installation Management subsystem (chapters 8 and 14):
 - Software Package (14.2): kernel of server S1, cluster's client NFS configuration. Software Repository (14.3), Node Management Agent (14.1)
- Monitoring and Fault Tolerance subsystem (chapters 9 and 15):

- Actuator Dispatcher, AD (15.5)
- Resource Management subsystem (chapters 6 and 12):
 - RMS Scheduler (12.3)
- Configuration Management subsystem (chapters 7 and 13):
 - Configuration Database, CDB (13.1)

Pre-conditions: The server S2 is up and running.

Event flow:

The Product Maintainer *A* runs a *configuration change* type (see 4.3.1) administrative script for upgrading the kernel version of S1-type machines, which consists of the following steps:

1. Add the SP with the new kernel version to the SR.
2. Change inside the CDB the default configuration for NFS-server S1 type machines for upgrading the kernel SP version.

To apply the change, Service Manager *B*, responsible for cluster P, executes the following *deployment* type (see 4.3.1) administrative script, when he considers it appropriate:

3. Query to CDB: Get the list of nodes (N1...Nn) that depend on server S1.
4. Change inside the CDB the configuration of the nodes (N1...Nn) to point from server S1 to server S2.
5. For each node in (N1...Nn), do (*in parallel*):
 - 5.1. Contact the RMS scheduler and ask for the node Ni to become available for maintenance using the `waitForFreeNode()` interface.
 - 5.2. Using the AD `dispatch()` interface, ask the NMA of Ni to reconfigure itself. This maintenance task is marked as intrusive. It will cause Ni to change its NFS mount table from S1 to S2 as soon as the machine enters maintenance state.
 - 5.3. If Ni is in production state, ask its NMA to change the state into maintenance.
 - 5.4. Wait for the reconfiguration issued in 5.2. to terminate using the `waitForCompletion()` interface of the AD.
 - 5.5. If Ni was in production, ask its NMA to put the node back to production state.
 - 5.6. If Ni was in production, notify the RMS Scheduler that the node is again available using the `releaseNode()` interface.
6. Wait for step 5 to complete on all nodes (N1,...,Nn).
7. On the NFS server S1, ask the NMA to reconfigure itself using the AD `dispatch()` interface. This maintenance task is marked as intrusive. As soon as S1 enters maintenance state, this will cause the NMA on S1 to install and configure the new kernel SP and to reboot for the new kernel becoming active.
8. If S1 was in production, ask the AD to put the server in maintenance state.
9. Wait for the reconfiguration issued in 7. to terminate using the `waitForCompletion()` interface of the AD.
10. If S1 was in production, ask the AD to put the server back to production state.

An extension of this use-case could be to revert to the initial configuration, e.g. to re-configure again the nodes to point back to the previous NFS server S1 instead of S2. This would imply to re-do similar operations as in step 5 after completion of step 9.

Step 5 is executed in parallel on all the nodes. This way, all nodes can be updated at the same time to use the new server.

Step 5 is generic as it does not depend on the specifics of this change operation, but could be applied to any NMS update on RMS-dependent cluster nodes. Therefore, all its sub-steps can be coded in a generic library as described in 4.4. Also, the two administrative scripts could also be written generically such as to be re-usable. This can be achieved by providing specific server and cluster identifiers as parameters.

The present use-case shows well the difference between configuration change and deployment operations as described in 4.3. The Service Manager can concentrate on programming the deployment of the maintenance interventions, while the Product Maintainer focuses in defining the supported system environment.

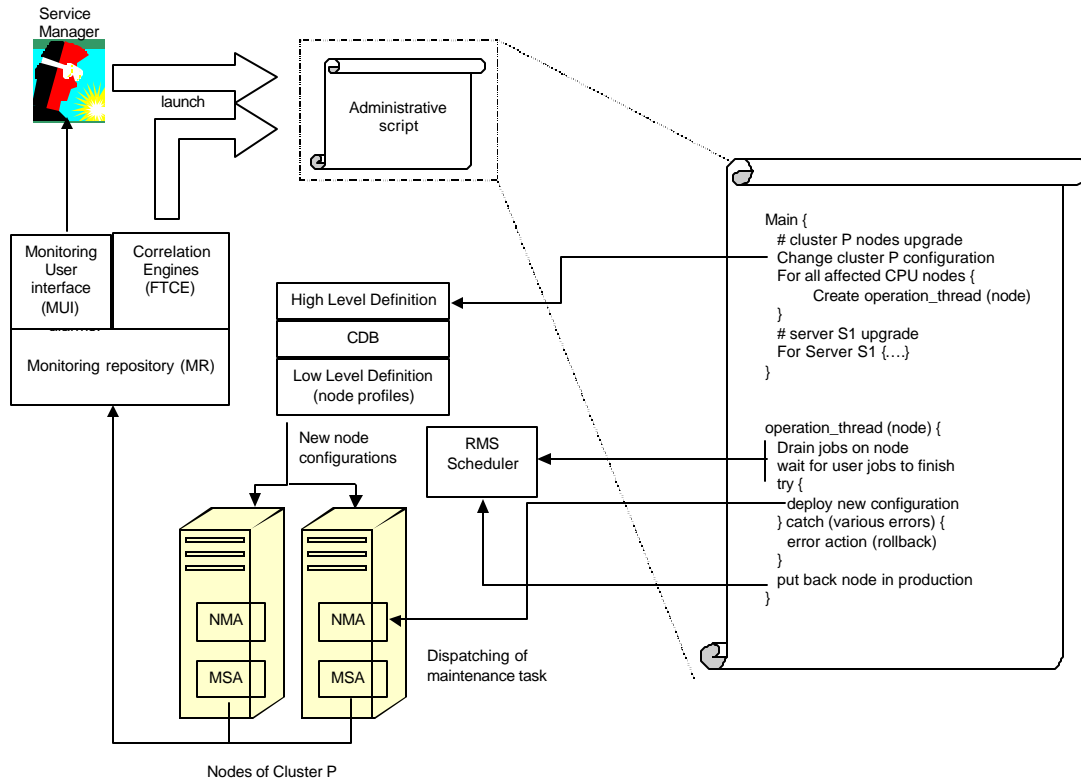


Figure 11: Upgrade of NFS server on a cluster use-case.

10.4. USE CASE: FAULT RECOVERY IN CLIENT/SERVER ENVIRONMENTS

Summary: A StorageElement server (e.g. running RFIO) fails and causes the rfio client application (rfiod) on cluster nodes to report an error when trying to access data files. The Monitoring and Fault Tolerance subsystem detects that and tries to repair the problem.

Actors involved: None.

Subsystems and components directly involved:

- Monitoring and Fault Tolerance subsystem (chapters 9 and 15):
 - Monitoring Sensor, MS (15.6)
 - Monitoring Sensor Agent, MSA (15.2)
 - Fault Tolerance Correlation Engine, FTCE (15.8)

Pre-conditions: Except for the RFIO server, all other involved components and media (e.g. network) are up and running. The rfio client is registered as a Monitoring Sensor (MS) (e.g. it implements the MS interface, possibly using a wrapper). The local node's FTCE is subscribed to the rfio failure metric (rfio_failure), and the Global FTCE (running on another node) is subscribed to inability to resolve a rfio local failure metric (local_rfiod_repair_failure).

Event flow:

1. The rfio client application running on the nodes depending on the failing rfio server instance detects a timeout error when trying to connect to its server for accessing data files.
2. The rfio client application reports the error via the MS API to the MSA using the newData() method of the Notify interface.
3. The local FTCE is notified that a rfio failure metric (rfio_failure) has been sampled.
4. The local FTCE will try to solve the problem locally according to internal rules, e.g. first checking the network status to the RFIO server, and then dispatching an actuator that restarts the rfio client. The rfio client is restarted successfully.
5. The local FTCE writes back a metric to the MSA indicating that the rfio client has been restarted successfully.
6. The rfio client still fails to connect to the RFIO server. *Steps 1 to 3 are repeated.*
7. The local FTCE detects (by consulting the MR) that the problem already happened a short time interval ago, and decides, according to its internal rules, to escalate the problem by writing back to the MSA another metric indicating its inability to resolve the rfio local failure (local_rfio_repair_failure).
8. The global FTCE is notified that the local_rfio_repair_failure has been sampled on several nodes.
9. The global FTCE concludes that the network is up and running, but that the RFIO server daemon is not responding correctly. As a consequence, it launches an actuator (a NMA restart method) via the Actuator Dispatcher on the RFIO server machine.
10. The RFIO server process is restarted via the NMA restart method.
11. The global FTCE writes back a metric to the MSA indicating that the RFIO server has been restarted successfully.

The principle outlined in this use-case can also be applied to other components that require client/server interaction, for example access to Software Repositories and other StorageElements.



APPENDIXES

11. APPENDIX: GRIDIFICATION COMPONENTS

11.1. COMPONENT: COMPUTING ELEMENT (CE)

11.1.1. Functionality

The ComputingElement (CE) receives requests for operations on resources from the Grid Scheduler (WP1) via the *Gatekeeper* [R18] from the Globus project [R17]. Examples of such control operations are job submission, job cancellation, resource reservations and reallocations. The CE receives together with the control operation, a credential (certificate), and the request description expressed in the Job Description Language (JDL [R2], defined by WP1).

The protocol between the Grid Scheduler and the ComputingElement is GRAM [R18]. It is based on and extends the functionality of the Globus *Job manager* [R18]. The ComputingElement is the sole entry point for Grid user jobs into a computing fabric.

The CE generates a per-fabric unique local jobID for every incoming job and maintains a repository of current local jobs.

The CE calls the LCAS and the LCMAPS service and acts according to the output of those components. In case of failure, it returns an error and does not call any further fabric-internal components.

The need for certain kinds of credentials (e.g., Kerberos tickets) is to be made known to the LCMAPS system before the job starts. Therefore, the need for such credentials should be specified as part of the JDL.

The CE contacts afterwards the Resource Management subsystem (RMS) and presents the entire operation description there. The RMS is not contacted if the authorisation in the LCAS or credential mapping in the LCMAPS fails.

The CE notifies the request originator (e.g. the Grid Scheduler) after a job has been declared 'finished' by the RMS. It also tells the LCMAPS component to invalidate temporary credentials if necessary.

The CE provides to the other components a repository in which to look up references to the user grid credential, local credentials and job description, based on the local unique job ID.

11.1.2. Dependencies

The CE is purely an interfacing and mediating component. The real functionality comes from the components it talks to.

11.1.3. Interfaces

- **submitJob**(request:JDL, allocationToken): JobID
- **getJobStatus**(id:JobID): JobStatus
- **cancelJob**(id:JobID):Result
- **allocateResource** (request:JDL): allocationToken
- **freeResource**(allocationToken:kindOfToken):Boolean

- **gGetCredential**(jobID): LCAScertificate
- **getRequest**(jobID): JDL

All methods can be called only within an established security context containing a global CAS signed authorisation certificate of the requesting party.

11.1.4. Internal Data

A *job repository* with the mapping between job ID's, and grid credentials and JDL request descriptions is kept.

11.2. COMPONENT: LOCAL COMMUNITY AUTHORISATION SERVICE (LCAS)

11.2.1. Functionality

The user's certificate signed by a Grid authorisation service (like the Globus Community Authorisation Service – CAS [R7]) is received by the LCAS component, together with the operation request expressed in JDL. The LCAS verifies the authorisation in an iterative and extensible way by presenting the operation request to *plug-in authorisation modules*, which grant or deny permission to the request.

A series of *basic plug-in authorisation modules* are provided by default. These are: static user checking, static user banning, and the application of resource-independent policies.

The LCAS provides hooks to insert *external authorisation plug-in modules*, e.g., to apply resource-dependent and availability policies. These external modules are to be provided by the other subsystems, for example the Resource Management subsystem for CPU, and the StorageElement (WP5) for SE storage resources.

The end result of the authorisation sequence is a user certificate signed by the LCAS. It includes an authorisation audit trail. This certificate is obtained from the FLIDS component.

11.2.2. Dependencies

Grid-wide authorisation: The LCAS assumes that a grid-wide authorisation service (eg. the Globus CAS) exists, that classifies users or roles as being part of a group. Off-line arrangements between the local centres and the grid-wide CAS define high-level authorisation for classes of users (for example: NIKHEF will accept ATLAS, LHCb and ALICE but not CMS jobs). The grid scheduler should take this authorisation into account before passing jobs to a fabric. This is yet to be resolved with WP8-10 WP1 and the WP7 Security team.

The LCAS primary focus is on individual or role authorisation. For this to work, the job credentials provided to the LCAS must include a unique identification of the user or role that submitted the job (for example: the Distinguished Name (DN) as stated in the user personal certificate).

The LCAS accesses the FLIDS.

11.2.3. Interfaces

- **get_fabric_authorisation** (request:JDL): LCAScertificate

This interface can only be called in an established security context.

11.2.4. Internal Data

A policy database is needed by the LCAS. This policy database is stored within the Configuration Management subsystem and is read by the LCAS. The LCAS can read but not modify this policy database.

11.3. COMPONENT: LCAS PLUG-IN AUTHORISATION MODULES

11.3.1. Functionality

The LCAS provides, as described, a framework for plug-in authorisation modules.

Subsystems that provide resources accessible via the Grid, have to provide such a module for granting or denying access to them. Examples are:

- The Resource Management subsystem for accounting and quota-based authorisation plug-ins, and for external network connectivity requests (see FabNAT).
- The StorageElement (WP5) for file access/space reservations.
- GriFIS for fabric information requests coming from the Grid.

The authorisation modules provided by default together with the LCAS component itself are:

- Static user checking against a banned list
- Application of high-level policy decisions that are dependent only on static resources like wall clock time
- Application of rules regarding external connectivity, based on a fixed list of allowed remote networks

When the LCAS calls an authorisation module, it provides it with the resource request description in JDL, together with the originator's certificate. With this information the authorisation module decides to grant access or not by returning a Boolean value.

11.3.2. Dependencies

The authorisation modules may require accessing information in the Configuration Management and Monitoring subsystems. For 3rd-party modules delivered by a subsystem, they may want to access subsystem-internal functionalities or data.

11.3.3. Interfaces

- **confirm_authorisation** (request:JDL, cred:Certificate): boolean

As this module is called between two local components, the credential needs to be passed explicitly.

11.3.4. Internal Data

This component should not maintain any permanent internal data.

11.4. COMPONENT: FLIDS

11.4.1. Functionality

The fabric-local identity service (FLIDS) provides an automated local certifying entity that can sign certificate requests (based on X.509 certificates) according to a predefined policy list.

The FLIDS is used by the LCAS for signing the local certificate requests generated by the LCAS.

The FLIDS is also used by the Installation Management subsystem for signing certificates required for initial installation.

11.4.2. Dependencies

None.

11.4.3. Interfaces

- **sign_certificate**(requesttosign:CertificateRequest): Certificate

This method is to be called within an established security context with the request generating party.

11.4.4. Internal Data

The FLIDS maintains a public-private key pair in unencrypted form in a private secure repository (and therefore not in the Configuration Management subsystem). A signing policy is maintained in the Configuration Management subsystem.

11.5. COMPONENT: LCMAPS

11.5.1. Functionality

The credential mapping service (LCMAPS) provides all credentials necessary to access services within the fabric. It only accepts requests that can present a credential properly signed by the LCAS.

The need for authentication mechanisms by a job is to be specified as part of the JDL.

If the identity of the user exists within an administrative domain addressed by the job, the LCMAPS returns the local credentials corresponding to this pre-existing identity.

For those users who have no pre-existing identity within the administrative domain addressed, the LCMAPS is able to generate a new identity.

The LCMAPS will at least provide for generation of UNIX user IDs and group IDs. If a local fabric supports other authentication methods, like Kerberos, the LCMAPS may provide mappings for those systems. The availability of these methods and the authentication and authorisation types will be dependent on the underlying mechanism.

If necessary, the LCMAPS registers new local credentials within the fabric's existing credential managing entity (for example, NIS or LDAP password servers).

The CE calls the LCMAPS on start-up of a job. The StorageElement (SE, WP5) may also call the LCMAPS for allocating a credential required for storage. The LCMAPS returns a unique handle to each lease request.

When a request has been finished (for example, because a job has finished, or because the SE has removed all files belonging to a user), the LCMAPS is called.

When the last request for a given local credential has been finished, non-permanent leases may be removed depending on local policy as specified in the Configuration Management subsystem. The erasure can only happen after the last subsystem holding a lease on the credential has finished (e.g. a UID still held by the StorageElement).

The issued local credentials may have a limited lifetime. For UNIX uids and gids, the LCMAPS service will have the possibility to make the mapping persistent and re-usable.

The LCMAPS must create and issue local credentials for every authorised user. No additional authorisation is done at this level. Sole reason for refusing the mapping is lack of resources at this level, e.g. no more free uids available.

11.5.2. Dependencies

- Access to local databases containing the ‘permanent’ or ‘site’ repository of issued local ID’s and their associated certificates.

The LCMAPS provides auditing logs for storage to the Monitoring and Fault Tolerance subsystem.

11.5.3. Interfaces

All methods can be called only within an established security context containing a LCAS-signed authorisation certificate of the requesting party.

- **newLeaseLocalCredential**: leaseID
- **queryLeaseLocalCredentials**: leaseID[]
- **addCredentialType**(leaseID, type:localCredentialType): localCredential
- **queryCredentialType**(leaseID, type:localCredentialType): localCredential
- **removeCredential**(leaseID, localCredential): Boolean
- **endLeaseLocalCredential**(leaseID): Boolean

11.5.4. Internal Data

The LCMAPS maintains a repository of issued local credentials (a table with index key the LeaseID, containing a list of all issued local credentials). It also keeps a table with all leases related to an LCAS-signed role identity (the key being the subject of the LCAS credential, e.g. "LHCb MC production manager", who might have submitted multiple independent jobs under different local credentials.)

Access to this repository is restricted. Entries can only be read by processes currently running with one of the credentials associated with this identity.

11.6. COMPONENT: GRIFIS

11.6.1. Functionality

The GriFIS provides a plug-in framework for *information providers* for abstracting externally relevant information from intra-fabric monitoring and configuration information. The information obtained from these subsystems can also be correlated with the dynamic and semi-static information available from the Resource Management subsystem. The resulting information from the GriFIS correlator is,

when possible, presented as attributes using the JDL semantics for easier matchmaking by the Grid Scheduler. This way, the GriFIS allows for plugging in information providers that can be called to calculate monitoring metrics needed by other DataGrid work packages. The GriFIS publishes this information in the IMS (WP3) associated to the fabric.

The published information is defined by each information provider, and may include:

- List and types of available resources (eg. queues)
- Resource boundaries (eg. minimal available temporary working storage, maximum CPU time, maximal running jobs)
- Current resource status (eg. current running jobs, total jobs)
- Resource availability (eg. time windows where the resource is up)
- Installed application environments
- Attached Storage Elements (SE's) and their access protocol

11.6.2. Dependencies

Monitoring and Fault Tolerance subsystem: The GriFIS information providers include a correlation engine plug-in component for the Monitoring and Fault Tolerance subsystem.

Configuration Management subsystem: The GriFIS information providers also obtain data from the configuration database, e.g. available application environments.

IMS (WP3): The correlated information has to be published to the IMS.

11.6.3. Interfaces

To be defined.

11.6.4. Internal Data

To be defined.

11.7. COMPONENT: FABNAT

11.7.1. Functionality

Enabling in- and outgoing network traffic from fabric nodes is a per site policy decision. No assumptions on the visibility of computing nodes from the external network can be made for several reasons, including for example, security policies and IPv4 address space limitations. But such communication may be necessary to support, for instance, MPI between fabrics with restricted external connectivity. In the case that the nodes do not have external connectivity, the FabNAT component allows mapping connections between jobs running on local fabric nodes and the outside Grid.

The FabNAT component provides a method for streaming connections (data pipes for visualisation, interactive sessions, MPI, etc) to be channelled out of the local fabric onto the wide-area Grid environment. This component is not intended to address the staging of input or output files or programs.

The FabNAT provides a gateway system to create and destroy streaming connections between individual worker nodes within a fabric and the external fabric boundary. This fabric boundary is defined as being on the same connectivity level as the ComputingElement CE.

The need for external communication streams by a job is to be specified as part of the JDL, together with the final destination and the communication type (eg. TCP or UDP sockets)

The FabNAT provides an LCAS plug-in authorisation module to check the validity of the communications destination requested. This is a static check only as it does not depend on the job characteristics.

The FabNAT subsystem relies on the Resource Management subsystem to approve or disapprove of the use of connections by a specific job. For this, the need for connections needs to be accepted by a RMS plug-in authentication module in the LCAS. The availability of communications channels are announced to the Resource Manager on request.

The FabNAT is then contacted by the Resource Manager subsystem for the actual communications channel request.

If the internal connectivity of the nodes uses a network protocol or network addressing space different from the one used on the external side of the ComputingElement and it is required that connectivity is provided to a final destination that cannot be tunnelled transparently, then the FabNAT component will maintain a repository where the mapping between intra-fabric connectivity and the externally visible connectivity is stored. This repository is an integral part of the fabric's GIS.

The FabNAT subsystem does *not* guarantee that a persistent communications channel can be maintained in a fabric that allows job migration from a node to another.

The FabNAT subsystem assigns the ports and port-ranges to a specific job or a specific machine. The ports or port ranges should be considered a resource, to be managed in a way consistent with the machine and storage requirements of a job.

11.7.2. Dependencies

None.

11.7.3. Interfaces

- Request a connection for a given external destination and a given connection type
- Close an already open connection
- Verify availability for a given connection type

11.7.4. Internal Data

FabNAT maintains a repository with the mapping between intra-fabric connectivity and the externally visible connectivity.

12. APPENDIX: RESOURCE MANAGEMENT COMPONENTS

12.1. COMPONENT: RMS INFORMATION SYSTEM

12.1.1. Functionality

The RMS uses static and dynamic information. This information describes the states of the RMS and its managed resources. Static information, as the characteristics of nodes, is stored by the Configuration Management subsystem. Dynamic information may be obtained from cluster batch systems via Proxies, or from the RMS scheduler or the monitoring subsystem. Dynamic information is stored in the *RMS Information Repository*. Both static and dynamic information is accessed by components of the RMS and other WP4 subsystems. The *RMS Information Repository Manager* (RMSIRM) is responsible for controlling access to the repository by components of the RMS.

The RMS Information Repository can be organised as a collection of files or by using a relational database. Information about entities is combined within logical structures that are described by schemata. The schemata for node, queue and job information can be found in [R6].

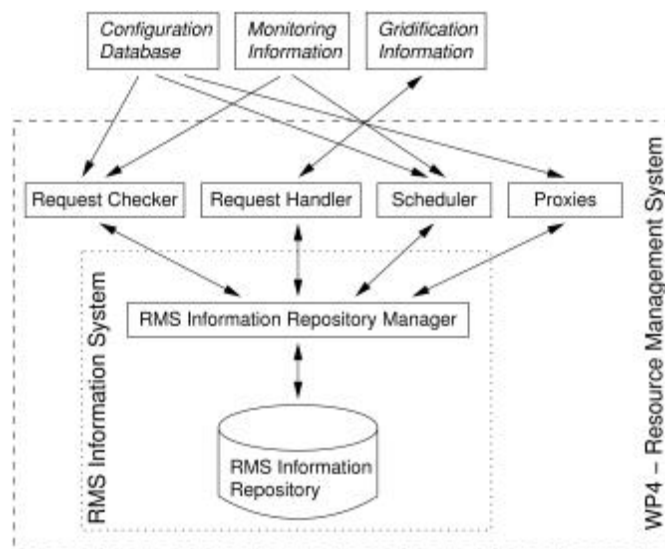


Figure 12: Detailed architecture of the RMS Information System

12.1.2. Dependencies

None.

12.1.3. Interfaces

Functions to store, delete or retrieve information:

- **select** (schema, [attribute, value/value_range]*): matching records
To retrieve records which are stored for a given schema.
- **add** (schema, [attribute, value]*): result
Store a record for a given schema.

- **delete** (schema, [attribute, value/value_range]*): result
Remove all matching records from the repository.

12.1.4. Internal Data

Information about nodes, queues, and jobs (see [R6]).

12.2. COMPONENT: REQUEST HANDLER

12.2.1. Functionality

The Request Handler accepts job control requests delivered via the ComputingElement (CE) component of the Gridification subsystem. It also contains a sub-component, the Request Checker, which is responsible for verifying and validating job requests. After receiving a job request, the Request Handler manages it by going through the following steps:

1. Store the request in the RMS Information System.
2. Verify the request, e.g. certified by LCAS, perform a basic check for resource requirements, etc. This is done by the Request Checker sub-component, using information from the Configuration Management subsystem.
3. Obtain all local credentials from the Gridification subsystem.
4. Schedule the job via the RMS Scheduler component (12.3), i.e. by matching the resource requirements, performing eventual advance reservations and co-allocations, and inform the ComputingElement.
5. When the job should start, ask the Scheduler to recheck the job requirements with the current status of the cluster batch systems. This is done with information from the Monitoring System. If this is not successful, try to reschedule the job or cancel the job with a failure message.
6. Submit the job via the Scheduler to the cluster batch system.
7. During the job execution, status information received from the Scheduler is transmitted to the CE and to the RMS Information system.
8. When the job has finished, send information back to the CE, containing job status and accounting information.

If the verification or scheduling step fails, the Request Handler replies with a failure message.

Job status information is made available and includes:

- Job status (i.e. started, suspended, scheduled, running, failed, done)
- Job queue information (length and/or estimated job start time)
- Job resource consumption (i.e. times, memory, swap, I/O, number of processes)

The Request Checker sub-component verifies several characteristics of a job request. It may check the existence and integrity of the LCAS certificate. Furthermore, it verifies whether the fabric can provide the requested resources.

12.2.2. Dependencies

The Request Handler depends on the Request Checker for validating requests. The RMS information system is used for storing requests. The Scheduler component is required for job reservations and executions.

12.2.3. Interfaces

The Request Handler provides the following functions:

- **submitJob** (JDL_description): jobID

Submits a job with the job description expressed in JDL. This method may also be called to ask for advance reservation or co-allocation.

- **cancelJob** (jobID): result

Cancels a job.

- **getJobStatus** (jobId): status of the job

Obtains job information like the status, queue information, or resource consumption.

Other interfaces to be defined include:

- Setting and retrieving job input and output environments (*sandbox*)³

12.2.4. Internal Data

Requests are stored in the RMS Information System.

12.3. COMPONENT: SCHEDULER

12.3.1. Functionality

The task of the Scheduler component is to assign resources to incoming job requests. It deploys different strategies to generate a schedule that fulfils the job requirements, makes efficient use of available resources and applies site policies. First, the scheduler matches the resource requirements described by the job request with the existing and available resources. If this process is successful, the scheduler makes provisional assignments of the resources to the job, i.e. it generates or updates a schedule. The schedule is provisional in the sense that the assignment to actual resources is only done just before the scheduler submits jobs to the underlying batch systems. An advantage of this approach is that the scheduler can perform better load balancing, adapt the schedule to resource failures and consider maintenance events. Enhanced native scheduling strategies and advance reservation techniques are currently subject to research.

12.3.1.1. Local Batch Jobs

One requirement for the RMS is to integrate clusters that are managed by cluster batch systems with a minimal impact on the behaviour of local users when submitting jobs. The main possibilities for local job submission are:

- a) All local user jobs are submitted to cluster batch systems via the RMS Scheduler (Figure 5, first case).

³ A temporary solution has been developed by WP1 for testbed 1.

- b) Jobs of local users are submitted directly to the cluster batch systems (e.g. the `bsub` interface in LSF), without passing via the scheduler (Figure 5, second case).

In the first case, a local user may submit the job directly to the RMS Scheduler, or via a wrapper application that emulates the common environment and interface from the batch system, (i.e. emulating `bsub` syntax). In the latter case, for retaining control over the jobs running on the fabric resources, the RMS Scheduler can manage the execution of local jobs and jobs received via the Grid, by suspending or enabling them. Another approach is to send local jobs to a queue where execution is steered by the RMS scheduler.

The decision is policy-based and has an important impact on the available functionality. For example, advance reservation schedules can be invalidated, and fair-share load balancing made impossible, if jobs can be submitted directly to the batch system.

A detailed discussion on the impact on fairness, advance reservations, security, job status and load balance can be found in [R6].

12.3.1.2. Node failure

If a node crashes due to some arbitrary reason, the RMS Scheduler must take this into account, i.e. for scheduling incoming job requests and for rescheduling already-scheduled jobs. The Monitoring and Fault Tolerance subsystem observes the failure and triggers administrative scripts for recovery to deal with the failure. The recovery procedure may involve the calling of one or more control methods of the RMS to perform maintenance jobs on the node. The RMS Scheduler may verify if its schedule is affected. The RMS Information System may also be notified in order to update its information repository.

12.3.1.3. Maintenance tasks

If a maintenance task (see 4.2) has to be executed, the initiating component (e.g. an administrative script application) asks the RMS Scheduler for a time slot on the affected nodes. This is necessary as both intrusive and non-intrusive maintenance tasks can affect the performance of a node and thus invalidate the job requirements. If the duration of a maintenance task is known in advance, the RMS Scheduler can plan for future reservations and job scheduling. This is done taking into account that configuration of the nodes, e.g. the installed software environment, may have been changed after the maintenance task's completion.

The Control Functions defined below define the fabric management interface of the RMS. Administrative script applications may use the primitives to trigger actions or set the state of the RMS and its managed resources.

12.3.2. Dependencies

The RMS scheduler depends on the RMS information system for information on jobs. It also relies on the proxies for interfacing to the batch systems.

12.3.3. Interfaces

Control functions:

- **getTimeSlot** (node/s, begin, end, duration, priority): result, per node (begin, end)
Obtain a time slot on one or more nodes, e.g. for maintenance jobs
- **announceNode** (node/s): result
Declare one or more nodes available for production

- **removeNode** (node/s, when): result
Remove one or more nodes from production
- **setNodeState** (node/s, state/s): result
Set the state of one or more nodes
- **getNodeState** (node/s, state/s): state/s
Get the state of one or more nodes
- **announceQueue** (name): result
Declare a queue available for queuing
- **removeQueue** (name): result
Remove a queue
- **setQueueParameters** (name, queue attributes): result
Set parameters of a queue
- **getQueueParameters** (name): queue attributes
Get parameters of a queue
- **getQueueState** (name, state): result
Set the state of a queue
- **getSchedule** ([jobid], [queue]): schedule
Obtain the schedule
- **waitForFreeNode** (node, waitTimeout, [duration], force): result
Wait until the node is free, i.e. not executing a job nor about to execute a job
- **releaseNode** (node): result
Release a node, usually called in conjunction with **waitForFreeNode**
- **announceConfigChange** (entity, [jobid], [date])
Announce a configuration change, a parameter may specify when the configuration will be updated (by jobid or date)

12.3.4. Internal Data

A schedule defined by the assignment of resources to job requests.

12.4. COMPONENT: PROXIES

12.4.1. Functionality

Proxies enable the RMS Scheduler to access cluster batch systems, such as PBS , LSF and Condor. Each cluster batch system has its own proxy. A proxy consists of one or more scripts that translate job descriptions written in the Job Description Language (*Condor ClassAds*) into commands and parameters of the cluster batch system. The translation of the cluster batch system specific syntax for job status information is also done within the Proxy.

12.4.2. Dependencies

Proxies depend on the underlying batch system they interface to.

12.4.3. Interfaces

Defined through the needs of the RMS Scheduler and the interfaces that are provided by the cluster batch systems.

12.4.4. Internal Data

None.

12.5. COMPONENT: PLUGIN FOR RESOURCE AVAILABILITY CHECKS

12.5.1. Functionality

The Gridification subsystem performs a set of authorisation checks. These checks can be implemented by modules that are called by the LCAS component. The RMS provides a plug-in module that checks and verifies the availability of resources.

12.5.2. Dependencies

The plug-in module retrieves information from the RMS Information System and from the Configuration and Monitoring subsystems.

12.5.3. Interfaces

This is a plug-in module for the LCAS. The interfaces are defined by the LCAS.

12.5.4. Internal Data

None.

12.6. COMPONENT: INFORMATION PROVIDERS FOR GRIFIS

12.6.1. Functionality

The subsystems and components from the Workload Management Work Package (WP1) need information about resources available in a Computing Fabric. Information on the RMS subsystem is provided via information providers, which are integrated in the GriFIS component of the Gridification subsystem. The GriFIS component publishes the information to the Grid. The required information is obtained by querying the Resource Management components, in particular the RMS Scheduler and the RMS Information System. It includes:

- List and types of available resources (e.g. queues)
- Resource boundaries (e.g. minimal available working storage, maximum CPU time)
- Current resource status (e.g. current running jobs, total jobs)
- Resource availability (e.g. time windows where the resource is up)

12.6.2. Dependencies

The RMS information providers are integrated in the GriFIS component.

12.6.3. Interfaces

To be defined.

12.6.4. Internal Data

None.

13. APPENDIX: CONFIGURATION MANAGEMENT COMPONENTS

13.1. COMPONENT: CONFIGURATION DATABASE (CDB)

13.1.1. Functionality

The CDB stores configuration information and manages modification and retrieval access. The CDB transforms HLD into LLD node configurations, by means of one-way compilation. The CDB performs validation of configuration information. The CDB notifies nodes when their configuration has been changed.

13.1.2. Dependencies

None.

13.1.3. Interfaces

Profile retrieval – implements server side of CDP:

- Client (CCM) accesses the CDB server using CDP. Transport protocol is implemented on top of the HTTP protocol.
- Access to high level profiles (as well as the shape and transformations of the profile) is still to be defined.

Profile management:

- Nodes profiles are put as XML files into a specified directory that is known to the CDB's HTTP server.
- The design of the interface to express HLD is to be defined.

13.1.4. Internal Data

The CDB server keeps all configuration information. It holds a master HLD and other views of the information (LLD) computed from the master. LLD are stored in XML format. An example of the XML node profile is presented below:

```
<?xml version="1.0"?>
  <!DOCTYPE profile SYSTEM "http://cfg.inf.ed.ac.uk/1.0/profile.dtd">
<profile xmlns="http://cfg.inf.ed.ac.uk/1.0/profilens"
  xmlns:cfg="http://cfg.inf.ed.ac.uk/1.0/cfgns">

  <vmware>
    <encrypt>no</encrypt>
    <license>118456</license>
  </vmware>
  <disk>
    <device>/dev/hda</device>
    <partitions>
      <partition>
        <size>1000</size>
        <mount>/</mount>
      </partition>
      <partition>
        <size>250</size>
        <mount>/tmp</mount>
      </partition>
    </partitions>
  </disk>
</profile>
```

```
</disk>  
</profile>
```

Details of the node profile are found in [R8].

13.2. COMPONENT: CONFIGURATION CACHE MANAGER (CCM)

13.2.1. Functionality

The CCM is used by the applications running on the fabric nodes to access their configuration information. The CCM downloads the node profiles from the CDB and stores them locally on the node. Applications using the NVA API can access the information and use services such as configuration change notification. Thanks to storing the information locally, the information can be accessed even if the network link is not available (e.g. network timeouts), provided the profile has been downloaded previously or stored by any other means.

Applications can be synchronously or asynchronously notified by the CCM about changes to all or some specified part of their configuration information.

13.2.2. Dependencies

The CCM communicates with the CDB in order to download node profiles, and to receive notification about changes of the information.

13.2.3. Interfaces

- Interface to the CDB for fetching the node profile. The interface implements the client side of the CDP.
- Interface for receiving notifications about changes to the node profile.
- Interface for reading the contents of the node profile. The interface is used by the applications via the NVA API. The interface implements the server side of CMP.

13.2.4. Internal Data

The CCM parses and validates the contents of a node profile after it has been downloaded from the CDB. A successfully validated profile is stored in the internal repository.

13.3. COMPONENT: SOFTWARE LIBRARY IMPLEMENTING THE NODE VIEW ACCESS API (NVA API)

13.3.1. Functionality

This Library is used by the applications running on the node to communicate with the CCM and access their configuration information. The API calls give the means to query for data and subscribe for notifications about configuration information changes. As the complete API is large, an example of PERL code using the API is placed below:

```
use constant GRAPHICS  
=> "/hardware/graphics";  
use constant NETINTF  
=> "/system/network/interfaces";  
  
$cfg = getLockedConfig()
```

```
    or die("Can't get config!\n");
$node = $cfg->getNode(GRAPHICS . "/server");
$data = $node->getStringValue;

$path = Path->initPath(NETINTF);
$node = $cfg->getNode($path);
while ($name = $node->nextNode) {
    $ipath = $path->down($name);
    $node = $cfg->getNode($ipath->down("address"));
    $addr = $node->getStringValue;
}
```

The complete API can be found in [R11].

13.3.2. Dependencies

The Library communicates with CCM.

13.3.3. Interfaces

- Communication interface to the CCM, implements the client side of the CMP.
- NVA API, applications use this interface to communicate with the CCM via the library.

13.3.4. Internal Data

The Library may maintain some protocol-specific data e.g. describing the state of the communication with CCM.

14. APPENDIX: INSTALLATION MANAGEMENT COMPONENTS

14.1. COMPONENT: NODE MANAGEMENT AGENT (NMA)

14.1.1. Functionality

The Node Management Agent runs on all nodes on a computing fabric managed by WP4. The NMA is the core component of the installation management subsystem. It provides a framework for installing and managing the software packages (SP's). The NMA is run on request and fetches, installs, configures, upgrades and verifies the SP's that are configured in the Configuration Management subsystem to be available on that computer node.

The NMA obtains the desired *node configuration* from the Configuration Management subsystem. This configuration includes:

- Overall configuration information concerning the node as a whole (eg. Network configuration information, disk partition layouts)
- The list and the state of the SP's to be installed and/or configured on the node, together with their SR address, with node specific configuration if required.

The NMA provides two major modes of operation, *update* and *verification*:

In update mode, the NMA consults the complete list and states of the SP's in the Configuration Management subsystem and compares them to the currently installed ones (*full update*). Alternatively, a list of SP's to be checked can be provided as a parameter (*partial update*). The execution of an NMA update is also called a *maintenance task*.

To bring the node to the desired state, a sequence of actions has to be computed, which includes steps for downloading, installing, removing, updating, starting or stopping SP's. All these actions are *method calls* issued to the affected SP's from the NMA framework.

The NMA has to take the inter-SP *dependency information* (both from already-installed SP's and from SP's in the SR) into account. Once the list of actions has been computed and ordered, the NMA executes them. New or updated SP's are downloaded from the SR specified in the node configuration. There may be more than one SR serving a NMA.

In verification mode, the NMA proceeds as above, but only comparing the current node status to the one stored in the Configuration Management subsystem, without actually performing any operation. The NMA reports the result of the verification. Different levels of verification are available. The NMA can be called in verification mode by the Monitoring and Fault Tolerance subsystem, acting as a monitoring sensor. NMA verifications can be executed for all configured SP's (full verification) or for a subset of SP's (*partial verification*).

The NMA supports also the following additional operations via control functions:

- *Shutdown* the node: switches the node off.
- *Reboot* the node: power cycles the node.
- *Install* the node: performs an initial installation (or reinstallation) of the node (disk partitioning and setup, installation of initial core SP's on the node)

14.1.1.1. Validation of NMA Configuration Information

The NMA does not modify node information stored in the Configuration Management subsystem. It does not make any assumption about what is the correct state of the system. It assumes that the Configuration Management subsystem information is correct. In order to avoid run time errors, the NMA node configuration information is validated beforehand:

- Checking that all SP's in the node configuration are available on the SR for the node architecture.
- Checking for dependency conflicts and unresolved dependencies between SP's.
- Checking for file name space conflicts between SP's to be installed.
- Checking that the overall configurations (e.g. partitions) are valid.

How and where exactly these validation checks are to be performed is closely related with the planned high-level configuration description of the Configuration Management subsystem.

14.1.2. Dependencies

- Configuration Management subsystem: stores the node configuration
- SR: serves the SP's to be installed. In order to execute any installation/upgrade all required SP's must be present within the specified SR's.

14.1.3. Interfaces

The external interfaces to the NMA are:

- Update all SP's (or a subset)
- Restart/reconfigure all SP's (or a subset)
- Verify all SP's (or a subset)
- Reboot the node
- Shutdown the node
- Install / Re-install the node
- Set the state of the node (Production or Maintenance)

14.1.4. Internal Data

SP information:

- List of actually installed SP's on the node, their states and dependencies. This information is kept on the node.

Node configuration information in Configuration Management subsystem:

- Overall node configuration shared by all SP's, e.g. IP address, disk partition layout.
- SP's names, dependencies, source SR and node specific configurations.

14.2. COMPONENT: SOFTWARE PACKAGE (SP)

14.2.1. Functionality

Bundled software applications, modules or libraries which are to be installed on a computer node are packaged as units called Software Packages (SP's). A software package contains a set of directories/files and may contain additional installation and execution control instructions and data. An SP may also contain dependency information with respect to other SP's and the required system environment.

A Software Package for a given architecture is identified by its name, a major and minor version number, and a release number. Examples are: sendmail-8.9-1, monitoring_agent-0.45-2.

An SP is composed of: data (directories and files), meta-information (names, descriptions), dependency information and control methods (for installation, configuration, execution control).

1. **Data**: All files and directories that need to be installed on the node. Each file is classified according to its type:

- Read-only executable files (e.g. binaries, scripts)
- Configuration files
- Documentation files

2. **Meta-information**: This includes all relevant identifiers and descriptors for this SP, including:

- Name
- Version (major, minor) and release
- Architecture
- Additional descriptions (e.g. Summary, Copyright, Packager)
- Package size
- Security information (e.g. CRC's, signatures)

3. **Dependency information**: The SP's requirements on, and conflicts with, other SP's on the node and/or particular system resources (e.g. local files, available memory) that must be fulfilled:

- Installation dependencies and conflicts
- Runtime dependencies and conflicts
- De-installation dependencies and conflicts

4. **Control methods**: These methods provide the means by which SP can be manipulated by the Node Management Agent (NMA) for its installation, its configuration and the execution control:

- Installation methods: handle installation, removal, upgrades and verification of SP data components.
- configuration methods: update the configuration of an SP, verify it.
- control methods: control execution of an SP: start/stop, restart, verification of current state (this class is intended for use by daemons or administrative jobs like purging/rotating logs)

The installation and configuration methods are expected to access the Configuration Management subsystem and translate the information found there into the SP's native format.

Implementation of all the methods is not mandatory. Default methods are provided by a class hierarchy that allows to group SPs according to their functionality.

Each SP is in a *state* in relation to a particular node. The possible states include:

- Uninstalled
- Installed
- Configured
- Running

Only certain state transitions make sense. For simple SP's, these states collapse into either "uninstalled" or "installed".

For a smooth integration with the node's platform type (operating system and architecture), the SP is interfaced to platform-specific *system packagers*. Interfaces to standard system packagers like RPM [R12] and `dpkg` [R13] for Linux, and `pkg` [R14] for Solaris will be provided.

14.2.2. Dependencies

Configuration Management subsystem: SP control methods can access configuration information in the Configuration Management subsystem.

14.2.3. Interfaces

Query methods:

- Meta-information retrieval
- Information on data (files, directories)
- Information on dependencies
- Information on the state

Control methods:

- Installation: install, remove, update, and verify installation of one or more SP's.
- Configuration: configure, update, verify configuration of one or more SP's.
- Control: start, stop, and restart one or more SP's.

Defaults control methods are provided but may be overloaded by each SP.

14.2.4. Internal Data

None.

14.3. COMPONENT: SOFTWARE REPOSITORY (SR)

14.3.1. Functionality

The Software Repository (SR) manages and stores Software Packages (SP). It serves them to NMA client nodes on request.

There are two access types to the SR:

- Client access: For nodes running the NMA to retrieve software packages.
- Administrative access: Insert, remove and update software packages and run queries on the SR contents.

Client access: The NMA obtains the SP's to install from an SR. In the NMA configuration, for every SP to install, an SR address is indicated. The client access interface allows for transparently retrieving the SP from an SR using standard and scalable protocols like HTTP/HTTPS. This interface is used by the NMA for downloading the required SP's from their SR's. Client access is read-only – clients cannot change the SP's stored on the SR.

Administrative access: Access for adding, modifying or removing an SP is separated from the client access. The administrative access interface is used from within administrative scripts for managing the packages on the SR. Administrative access to the SR is subject to authentication, and authorisation based on Access Control Lists (ACL's).

When a new or updated package is to be added to the repository, the following checks apply:

- Verify the access rights of the requester
- Upload the SP
- Apply consistency checks on the SP (verify correctness and completeness of the SP)
- Declare the SP as available for the client interface.

The SR keeps in the Configuration Management subsystem a list of its available SP's. This is necessary for ensuring the consistency between the NMA configurations and the available SP's on the SR:

- The SR updates the SP list when an SP is added/removed/updated.
- The SR queries the NMA configurations stored in the Configuration Management subsystem before removing currently available SP's, making sure there will not be broken node configurations.

14.3.2. Dependencies

The SR depends on the Configuration Management subsystem as it stores and accesses information on it.

14.3.3. Interfaces

Client access interface:

- Get SP (SP identifier)

Administrative access interface:

- Add SP (Requester ID, SP identifier, SP URL)
- Remove SP (Requester ID, SP identifier)
- Update SP (Requester ID, SP identifier, SP URL)
- Queries (e.g. get list of SP's)

14.3.4. Internal Data

Data kept on the Configuration Management subsystem:

- Access rights on the SP's
- List with all available SP's

Data kept on the SR:

- SP's themselves.

14.4. COMPONENT: BOOTSTRAP SERVICE (BS)

14.4.1. Functionality

The Bootstrap Service (BS) provides the services needed for initial node system installation through the network. A BS server distributes setup data to the nodes to be installed and allows them to perform an initial boot, which triggers an initial run of the NMA component on the target node. Prior to any

installation, the NMA configuration of a target node has to be entered into the Configuration Management subsystem and validated.

A BS server stores *network boot information* and *initial system boot images* for nodes. The **network boot information** contains the required information for a node to be remotely installed. This information includes the client node's network MAC address, its IP and/or DNS address and the identifier of the system boot image to be used.

The **initial system boot image** contains the files to be installed on the machine's hard disk. Depending on the needs of each fabric, several types of initial boot images may be made available on the SR:

- The minimal requirement is to include a reduced operating system together with a working NMA on the boot image. The NMA is then run and performs all necessary operations for bringing the node to its configuration as described in the Configuration Management subsystem. This includes the setup of disk partitions and installing also a set of SP's.
- Complete system images can be built from *reference nodes*. Such a reference node is a ready to use installed and configured template node for a specific hardware and usage profile, for example a batch CPU node. The image can be obtained by cloning the data contained in the disk partitions on this reference node. In this case, the NMA only has to update node-specific information, like the network information (DNS name and IP address) for matching the node's configuration described in the Configuration Management subsystem.

For massive installations of nodes with an identical setup, having complete system images can be faster than running the installation from a minimal image. However for heterogeneous fabrics, the manual effort cost of creating and maintaining multiple complete system images may not be compensated by better installation performance.

After a node has been physically powered up, the following operations are performed as part of the node's bootstrap process:

- An initial boot program is loaded on the node, either over the network (e.g. using `bpbatch` [R20] or `pxe` [R19]) or via removable media (floppy, CDROM)
- The initial system image is loaded over the network (eg. using FTP/TFTP)
- The system is rebooted and an NMA full run is executed to make the node to match its configuration.
- This NMA configures and starts all required SP's. From this moment on, the node is ready for production.

14.4.1.1. Confirming a host identity on installation

When a host is first installed in a fabric, the NMA needs to obtain its configuration data from the Configuration Management subsystem. Since the configuration data can be privacy-enhanced, it may not be accessed in an anonymous or unencrypted form, a positive mutual identification (using X.509 certificates) between Configuration Management subsystem and the NMA may be required.

Depending on the fabric environment, installations may have to be secured:

- The Ethernet / IP addresses can be used as identifier if the network is trusted sufficiently
- If the network can't be trusted, the FLIDS component from the Gridification subsystem is used for signing certificates used as identifiers for initial installation.

14.4.2. Dependencies

The BS depends on the Configuration Management subsystem as it stores and accesses configuration information on it. It also relies on the NMA as this is started during initial installation. The FLIDS component is required for authenticated installations.

14.4.3. Interfaces

Administrative access interface:

- Register / deregister node on the BS
- Add / remove initial system image to the BS
- Assign initial system image to a node
- Activate / deactivate node for installation
- List nodes registered on the BS
- List available initial system images

14.4.4. Internal Data

The list of registered nodes, together with their network information is stored in the Configuration Management subsystem. The list of available initial system images is also kept in the Configuration Management subsystem, while the images themselves are stored on the BS server.

14.5. COMPONENT: INFORMATION PROVIDERS FOR GRIFIS

14.5.1. Functionality

In order to run specific Grid user jobs, a specific *Application Environment* may be required. An Application Environment is defined by the set of installed Software Packages that are needed for a specific job type to run, for instance analysis programs, mathematical libraries, system tools, shells, compilers. The information is obtained by querying the Configuration Management and Monitoring subsystems for the set of SP's installed on Grid-accessible clusters. This is provided via information providers, which are integrated in the GriFIS component of the Gridification subsystem. The GriFIS publishes the information about the installed application environment to the Grid.

14.5.2. Dependencies

Information providers are integrated in the GriFIS component.

14.5.3. Interfaces

To be defined.

14.5.4. Internal Data

None.

15. APPENDIX: MONITORING AND FAULT TOLERANCE COMPONENTS

15.1. BASIC DEFINITIONS

15.1.1. Metric

A *Metric* is a unique identifier of a monitored element. Examples of monitored elements are: “CPU load”, “daemon dead”. Different MSs on the same node may measure the same *Metric*.

15.1.2. Sample

A *Sample* is an instantiation of *{nodeName:String, metric:Metric, localTimeStamp:Time, measuredValue:String}*. An example of a *Sample* could be: {"lplus042", uptime, 90923483, "12345"}.

15.1.3. Notify

Notify is an interface with one method:

- **newData** (data:Object[]):void – this method is called when new data is available. The new data is passed in the *data* parameter, with an appropriate typecast.

15.2. COMPONENT: MONITORING SENSOR AGENT

15.2.1. Functionality

The MSA is a local agent that controls and reads out the MSs and inserts the data into the MR. The MSA is driven by configuration files that are produced from information stored in the Configuration Management subsystem. Because the configuration of the MSA largely depends on what other software packages are installed and running on a node, the update of the MSA configuration files is a potentially intrusive action that normally needs to be synchronized with the updates of those other packages. Therefore a special FTCE rule is assigned to check if the MSA configuration files need to be updated. In case the configuration files need to be updated a request is sent to the AD to update the installation of the monitoring SP. The AD queues the request together with other NMA requests so that the running of the actuators is synchronized once the node state permits it. Some configuration changes, e.g. the change of a sampling frequency, are non-intrusive and can execute immediately. Once the configuration files are updated the MSA dynamically reads any new configuration from them.

Another special FTCE rule assures that the average resource utilisation of the controlled sensors is kept within the required limits.

15.2.2. Dependencies

The MSA configuration is maintained by the Configuration Management subsystem. The MSA reads the following information from the Configuration Management subsystem:

- Sensor modules (executable or library function name)
- For each module, the identifiers of the metrics measured by that module
- For each metric, the sampling offset and frequency for that metric. The sampling offset is optional and can be specified either as relative to the starting of the sensor (default) or in absolute time (e.g. date and time or “every Sunday at noon”). Several metrics can be grouped to use the same sampling definition.

15.2.3. Interfaces

The MSA is driven by configuration files and has no externalised interfaces. However, one could also consider the MSA as a “Producer” in the Grid Monitoring Architecture [R15] of the Global Grid Forum [R16] and as such instantiate a Producer object for certain metrics. WP3 has adopted the GMA with relational implementation (R-GMA), which among other things will be used for application monitoring. The GMA proposes a dynamic framework where producers publish themselves via a directory service and consumers query that directory service to find appropriate producers. This would not primarily be used by the fabric monitoring infrastructure, which is rather statically structured around its measurement repository (MR) and the central Configuration Management subsystem, but for special cases of non-fabric type of monitoring such as application monitoring. The client user would then instantiate a consumer, which subscribes to measurements from the producer of the application monitoring. The alternative is that also application monitoring is stored as opaque objects in the MR, which the client user retrieves through queries. The latter alternative is easier for the fabric administrator to control while the former is more practical for the user. Some further investigation is needed to decide which of those two alternatives is the more appropriate.

15.2.4. Internal Data

None.

15.3. COMPONENT: MONITORING REPOSITORY

15.3.1. Functionality

All monitoring data is stored in the MR. Note that not only exceptions (alarms) are stored but also all performance data, which is sampled with regular frequency. The MR consists of a server and a client API. While the MR server is logically a central component in the WP4 architecture, it is for scalability reasons more likely that a hierarchy of MR servers is built where each hierarchy corresponds to relatively confined entities, e.g. the site hierarchy with of a set of services where each service hierarchy consists of compute clusters and storage servers and finally the compute cluster hierarchy consists of the individual nodes. However, as was mentioned previously in section 9.1, the structuring of the measurement data inside the MR, or the individual MR instances, makes no assumptions on the site hierarchy. The client API by default maintains a local cache of the latest measurements. The cache resides in persistent storage (disk) and assures that local fault tolerance correlation engines and actuators can process monitoring data even if the network is unreachable, e.g. during a reboot. The cache is also valuable for tracing of what happened after the network went down prior to a node hang or crash.

The MR client API provides methods for querying and inserting data. It also has a metric-subscription/notification mechanism intended for FTCEs to be notified every time the specified metric has been measured. With this interface the FTCEs only access the MR when new data is available and hence removes the need for polling. The metric-subscription is subject to authorization and regulated through a prior declaration in the Configuration Management subsystem.

15.3.2. Dependencies

The MR depends on the Configuration Management subsystem for its entire configuration. The configuration data includes: metric identifiers, MR client host, and metrics-subscription bindings.

15.3.3. Interfaces

- **getSamples**(nodeName:String[],metric:Metric[],startTime:Time, endTime:Time):Sample[] – get the samples matching the specified nodes, metrics and times.
- **getLastSamples**(nodeName:String[],metric:Metric[]):Sample[] – get the latest samples matching the specified nodes and metrics.
- **insertSample**(sample:Sample):void – insert sample in the MR.
- **subscribe**(nodeName:String,metric:Metric,notify:Notify):void – subscribe to notification via notify every time metric is measured on nodeName. The notify parameter specifies the caller's implementation of the Notify interface (see section 15.1.3). The MR calls newData(sample:Sample[]) to notify and deliver the samples to the subscriber.
- **unsubscribe**(notify:Notify):void – remove the notify notification
- **cleanUpCache**(nodeName:String[],metric:Metric[],startTime:Time, endTime:Time):void – clean-up the local MR caches. Note, this only removes data from the local MR caches, while the data in the global MR repository is not affected.

Each of the methods generates an exception if an unknown metric is specified. A metric is unknown if it has not been configured in the Configuration Management subsystem. The subscribe() method also generates an exception if the requested subscription binding has not been configured in the Configuration Management subsystem.

15.3.4. Internal Data

The MR maintains an internal list of all accepted metric-subscriptions.

15.4. COMPONENT: MONITORING USER INTERFACE

15.4.1. Functionality

The MUI is the normal way for humans to access monitoring data stored in the MR. The MUI uses the fabric layout stored in the Configuration Management subsystem to present the user with coherent views of monitoring data, e.g. services view, cluster view, node view. In very large fabrics it is important that the presentation of monitoring data is structured as a hierarchy of visualization so as not to overwhelm the operator with information. When managing tens of thousands of nodes it is useless to attempt to present the operator with a view of all of them. Even if some nodes may fail completely it may be irrelevant to report such events on the monitoring display because fault tolerance strategies can define that the node should be left down in case a recovery fails and the services moved to another node.

15.4.2. Dependencies

The MUI retrieves the fabric layout into clusters, services, etc. from the Configuration Management subsystem. It also uses a common fabric authentication/authorisation subsystem (see section 11.4 in the Gridification subsystem description) to restrict the access to the fabric monitoring.

15.4.3. Interfaces

None.

15.4.4. Internal Data

None.

15.5. COMPONENT: ACTUATOR DISPATCHER

15.5.1. Functionality

The AD is used to dispatch FTAs on the local node. The AD consists of a client API and an agent. It allows only dispatching of FTAs registered with the Configuration Management subsystem. An FTA is normally a simple maintenance task (see section 4.2) that calls externalised methods of the Node Management Agent (see NMA interface description, section 14.1). The AD agent listens to a network port for both local *and* remote requests, where the latter are normally a result of an inter-node recovery actions, for instance an administrative script launched by an operator or inter-node fault tolerance correlation engine (FTCE). The AD agent provides the following functionality:

- FIFO scheduling: the dispatch requests are queued as they arrive to the AD agent. Certain urgent actions (e.g. shutdown node because temperature is running high) may require a bypass of the normal scheduling. However, such actions imply immediate execution and hence there is no reason to maintain a special queue for them.
- Serialized execution: only one FTA can run at a time. This is necessary because the AD has no knowledge about the dependency between the FTAs, e.g. the running of an FTA may require that all FTAs queued in front of it have finished their execution.
- Maintains a persistent dispatch handle for each received request. There cannot be any persistent channel between the requestor and the AD, since the FTA can trigger a reboot of the system.
- Returns status information of any dispatched FTA upon queries with the unique dispatch handle. Status information can be retrieved at any moment. The status information of requests is kept until the timeout limit expires.
- Maintains its state (request queue, current running FTA, status of completed requests) in locally persistent storage (disk)

15.5.2. Dependencies

The AD agent uses the Configuration Management subsystem to retrieve actuator configuration. The actuator configuration includes a mapping from the actuator identifier and the associated local names of the actuator binary or script executable or library function. The AD agent must assure secure authentication/authorisation of all clients. For local clients it uses the local host security while for remote clients the AD agent relies on the common fabric authentication/authorisation component of the Gridification subsystem (see section 11.4).

15.5.3. Interfaces

The AD client API provides the following three methods:

- **dispatch**(*actuator:ActuatorId, timeout:Time*):DispatchHandle – submit a request for running of the actuator within the specified timeout period.
- **getStatus**(*handle:DispatchHandle, actuatorStatus:Integer, startTime:Time*):Integer - get the status information of the submitted dispatch request identified by handle. The arguments *actuatorStatus* and *startTime* are output parameters. The returned *actuatorStatus* can take either of the following predefined values:
 - AD_QUEUED - the request is still in the queue

- AD_RUNNING - the request is currently running
- AD_TIMEOUT - the actuator did never start because the timeout expired
- AD_FINISHED - the actuator has finished

The returned `timeStart` contains the execution start time of an actuator that has reached the status AD_RUNNING or AD_FINISHED. The `getStatus()` method returns the completion code (process exit status or library call return value) of the actuator. This value is associated the individual actuator. The AD merely delivers the actuator status to the client without any interpretation. It is not responsible for the completion of the actuator.

- **waitForCompletion**(handle:DispatchHandle):void - wait for the specified dispatch request to finish (or timeout).

Both the `ActuatorId` and `DispatchHandle` classes contain enough information for allowing for remote execution of actuators.

15.5.4. Internal Data

The AD agent maintains the dispatch queues, information on running actuator and a repository of finished actuator status in persistent local storage (disk).

Information about actuators and their associated identifiers is maintained in the Configuration Management subsystem. The configuration information contains a field for flagging an actuator as *intrusive* or *non-intrusive*. Intrusive actuators normally change the user environment on a node and should therefore only be executed when the node has been put in *maintenance* state (no user jobs are running, see chapter 4).

15.6. COMPONENT: MONITORING SENSOR

15.6.1. Functionality

A *sensor* is defined here to be a component (software or hardware) that knows a given set of Metrics and how to sample them. Every sensor is associated with at least one metric.

In the FMFT framework the MS is an (software) implementation of the *MonitoringSensor* interface. The MS constitutes the lowest level information producer. The MSs that are configured to run on a node are called by the MSA using the *MonitoringSensor* API methods. The API allows for periodic samplings triggered by the MSA as well as asynchronous samplings triggered by the MS itself.

15.6.2. Dependencies

The MS constitutes the plug-in layer for any information producer to the monitoring system. The dependency of an MS on other WP4 subsystems or other external components thus depends on the provider of that particular MS implementation.

15.6.3. Interfaces

The MS must implement the *MonitoringSensor* interface, which has the following methods:

- **start**(metric:Metric, args:Object[], notify:Notify):void - initialises the MS to start sampling metric. The `notify` parameter specifies the caller's implementation of the `Notify` interface (see section 15.1.3). The MS calls `newData(sample:Sample[])` to notify and deliver samples to the MSA. The `args` argument is used to pass parameters associated with the `metric` (e.g. uid/gid of a daemon process).

- `stop(metric:Metric):void` - removes the specified metrics from the ones sampled by the MS.
- `getSample(metric:Metric[]):void` - request a new sample of `metric`. The sample is returned via the `notify` callback object specified in the call to the `start()` method.

An exception is generated if a specified `metric` is unknown to the MS.

The reason for using the `notify` callback mechanism to pass back measurement values is to avoid blocking `getSample` calls.

15.6.4. Internal Data

None.

15.7. COMPONENT: FAULT TOLERANCE ACTUATOR

15.7.1. Functionality

The FTA is the lowest level fault tolerance component. Every fault tolerance actuator is associated with at least one actuator identifier. An actuator identifier defines a unique combination of a loadable actuator module (executable or library function name) and an administrative task.

15.7.2. Dependencies

The FTA uses the Configuration Management subsystem for retrieving its specific configuration.

15.7.3. Interfaces

The FTA must implement the *FaultToleranceActuator* interface, which has the following methods:

- `run(task:MaintenanceTask,location:StatusLocation,notify:Notify):ErrorStatus` - run specified `task` and notify its completion. The `notify` parameter specifies the caller's implementation of the `Notify` interface (see section 15.1.3). The `location` contains the location address that the AD agent should use to retrieve the actuator return status. The FTA calls `newData(null)` to notify the caller (AD) of its status.

The `run()` method is non-blocking. Note that the return value of `run()` only reports the success/failure of the starting of the maintenance task and not the success/failure of the task itself.

15.7.4. Internal Data

None.

15.8. COMPONENT: FAULT TOLERANCE CORRELATION ENGINE

15.8.1. Functionality

The FTCE is the central component of the fault tolerance framework that allows for automation of fault detection and execution of recovery or preventive actions. The FTCE is a special case of an MS in the sense that it implements the *MonitoringSensor* interface and it is sampled either periodically or asynchronously by the MSA. However, the FTCE does not collect any external data but rather reads in monitoring data from the MR and processes that data to produce its own output metric value. A FTCE is typically implemented as an administrative script application procedure (see 4.1) and may as such call components of other WP4 subsystems. An FTCE can execute either locally on the node in a tight local monitoring – recovery action loop (e.g. check daemon and restart it if it is dead) without any remote interaction, or remotely if the FTCE correlates data from many nodes and executes collective recovery actions on all nodes in parallel (e.g. check the status of a central server and reconfigure all clients to use a standby server if the central server is down).

15.8.2. Dependencies

The FTCE is able to interact with most other WP4 subsystems.

15.8.3. Interfaces

The FTCE implements the *MonitoringSensor* interface (see section 15.6.3).

15.8.4. Internal Data

None.