CONTRACT NUMBER 508830

DEISA

DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR SUPERCOMPUTING APPLICATIONS

European Community Sixth Framework Programme
RESEARCH INFRASTRUCTURES
Integrated Infrastructure Initiative

First release of DCRMP

Deliverable ID: DEISA-SA3-3
Due date: October, 31st, 2005
Actual delivery date: November 25, 2005
Lead contractor for this deliverable: CINECA, Italy

Project start date: May 1st, 2004
Duration: 4 years

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	X
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document Keywords and Abstract

Keywords:	Deployment, Grid, Middleware, HPC
Abstract:	This document describes the architecture and the implementation details of the DEISA Cluster Resource Management Package (DCRMP), a tool to help system administrators to automate the installation process and to perform the required cross-checking of dependencies among the different software components of the DEISA production middleware

Table of Content

Table of Content 3
List of Figures 4
List of Tables 4
1. Introduction 5
1.1 Executive Summary 5
1.2 Document structure 5
1.3 References and Applicable Documents 6
1.4 Document Amendment Procedure 6
1.5 List of Acronyms and Abbreviations 6
2. DCRMP overview 8
2.1 DCRMP taxonomy 8
2.2 Main use case 9
3. Packaging technology 12
3.1 RPM Package Manager 12
3.2 Grid Packaging Tools 13
3.3 MKPKG 14
3.4 Debian 15
3.5 PACMAN 15
3.6 Conclusions 16
4. Common environment definition 18
4.1 UNICORE 18
4.1.1 Architectural constrains and software requirements 19
4.1.2 Installation schema 20
4.1.3 Connection topology 22
4.1.4 Configuration files 23
4.2 RMIS 23
4.2.1 Architectural constrains and software requirements 25
4.2.2 Installation schema 26
4.2.3 Connection topology 27
4.2.4 Configuration files 28
5. Implementation details 29
5.1 Gridconfig 29
5.2 Package Creator 29
5.3 Distribution system and software requirements 30
5.4 Future directions 31
6. Roadmap and future direction of the DCRMP task 32

List of Figures

Figure 1 - DCRMP distribution configuration....9
Figure 2 - Main use case diagram....10
Figure 3 - UNICORE components diagram....19
Figure 4 - Installation use cases....21
Figure 5 - UNICORE connection schema....23

List of Tables

Table 1 - Acronym list....7
Table 2 - Install DCRMP use case....10
Table 3 - Application install use case....11
Table 4 - Component install use case....11
Table 5 - UNICORE sub-components....19
Table 6 - UNICORE configuration files....23
Table 7 - RMIS sub-component description....25
Table 8 - RMIS configuration files....28
Table 9 - Distribution system directory structure....30

1. Introduction

1.1 Executive Summary

The DEISA Service Activity “Resource Management” (SA3) has achieved one of its milestones, the Intermediate Grid Service production status through the adoption of a common set of software applications like UNICORE and the GT2 MDS [3]. The deployment of such a software infrastructure involves the distribution of software packages across several machines available at multiple sites and requires the check of numerous dependencies. It follows that this task can be extremely cumbersome and complex.

Therefore SA3 planned to provide the DEISA Cluster Resource Management Package (from now on DCRMP) that will help system administrators to automate the installation process and to perform the necessary cross-checking of dependencies between the different software components. The DCRMP will use existing tools that are available to install the individual components and will provide an efficient checking mechanism that is able to highlight missing components (e.g. libraries as part of the package). The final aim of the DCRMP activity is to reduce the deployment time for the DEISA environment in order to maximize the availability of the infrastructure. This deliverable presents the current status of DCRMP.

1.2 Document structure

The second section describes the architecture of the DRCMP and its main components.

The third section presents a detailed comparison of currently available technology to manage packages.

The fourth section provides a description of the middleware architecture (UNICORE and RMIS) distributed by means of the DCRMP.

The fifth section describes the system implementation details.

The sixth section offers the DCRMP roadmap.

1.3 References and Applicable Documents

[1] http://www.unicore.org

[2] http://sourgeforge.net/projects/unicore

[3] DEISA Service Activity 3 – Deliverable D-SA3-2a

[4] DEISA Service Activity 3 – Deliverable D-SA3-1b

[5] http://www.rpm.org

[6] http://en.wikipedia.org/wiki/RPM_Package_Manager

[7] http://www.hpl.hp.com/personal/Carl_Staelin/mkpkg/help/mkpkg.html

[8] http://en.wikipedia.org/wiki/Deb_%28file_format%29

[9] http://www-unix.grids-center.org/r6/ecosystem/packaging/pacman.php

[10] http://physics.bu.edu/~youssef/pacman/

[11] http://www.ncsa.uiuc.edu/Divisions/ACES/GPT/

[12] http://www-unix.grids-center.org/r6/ecosystem/packaging/gpt.php

[13] http://www.nsf-middleware.org

[14] http://rocks.npaci.edu/gridconfig/

[15] http://www-03.ibm.com/servers/aix/products/aixos/linux/download.html

[16] http://www.uml.org/

[17] Fowler M., “UML Distilled” Addison-Wesley, 2004

1.4 Document Amendment Procedure

The initial document amendment procedure is via communication between members of DEISA SA3 team. The document is then submitted for review to the DEISA Executive and an Executive appointed DEISA reviewer. The document is then amended according to comments received from the Executive and the DEISA appointed reviewer. It is subsequently re-submitted to the DEISA Executive for submission to the EU.

1.5 List of Acronyms and Abbreviations

Term/Acronym	Definition
AJO	Abstract Job Object
CA	Certificate Authority
CPAN	Comprehensive Perl Archive Network

CPE	Common Production Environment
DCRMP	DEISA Cluster Resource Management
DMZ	DeMilitrarised Zone
GUI	Graphical User Interface
GW	Gateway
HPC	High Performance Computing
IDB	Incarnation Data Base
JKS	Java Key Store
JRE	Java Runtime Edition
MDS	Monitoring & Discovery System
NJS	Network Job Supervisor
NMI	NSF (National Science Foundation) Middleware Initiative
OS	Operating System
PERL	Practical Extraction and Report Language
PHP	PHP: Hypertext Preprocessor
PKCS	Public Key Cryptography Standards
POSIX	Portable Operating System Interface
RMIS	Resource Management Information System
SA	Service Activity
TSI	Target System Interface
UNICORE	UNiform Interface to COmputing REsources
Uspace	UNICORE space
UUDB	UNICORE User Data Base
VO	Virtual Organization
Vsite	(UNICORE) Virtual site
XML	eXtensible Markup Language

Table 1 - Acronym list

2. DCRMP overview

The DEISA Cluster Resource Management Package (DCRMP) is able to simplify the installation process of multiple software packages on different machines at a site through the collection of aggregated information. One of its main feature is the capability to store all the configuration information for multiple applications in a common repository. This knowledge is used to create the multiple configuration files that are necessary for the applications deployment. Additionally, software checks are performed to identify dependencies and pre-requisites.

This section presents a description of the DCRMP structure followed by the explanation of the main use case diagram that shows the functionalities of the software.

2.1 DCRMP taxonomy

The DCRMP is based on the following taxonomy that will be adopted throughout the rest of the document, to explain DCRMP's functionality.

A DCRMP distribution is composed of several *Applications* based on different technologies.

An *Application* in turn can be composed of one or more *Components*.

A *Component* is a set of executables, configuration files, documentation, and license files etc. that need to be deployed and executed on the same host.

A *Component* can be composed of one or more *Sub-Components* depending on the structure of the application.

In Figure 1 an example of a DCRMP distribution configuration is presented.

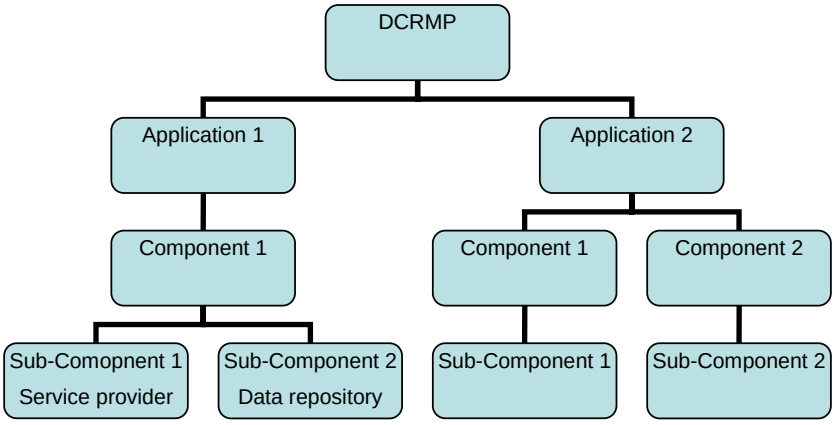

Figure 1 - DCRMP distribution configuration

The example in Figure 1 is composed only of two applications. It is important to underline that Application 1 needs to be deployed only on one machine even though it is composed of two Sub-components whereas application 2 may require two machines, due to its two Component structure.

From now on we are going to use UML Standard naming conventions [16], [17].

2.2 Main use case

In Figure 2 a typical use case where system administrators carry out the installation of one Application by means of the DCRMP is presented. In this scenario the system administrators perform 'Install DCRMP' once, when they use the DCRMP for the first time. During this phase they supply all the input data describing the general site configuration. Subsequently, the system administrators first choose the application that they want to install and provide all the information necessary to set it up ('Install Application'). At this point the DCRMP will extract the set of packages for each application component. Thereupon these packages will be executed on the hosts selected by the system administrators in the previous phase. In most cases some final configuration action will be performed by the system administrators to finalize the installation process.

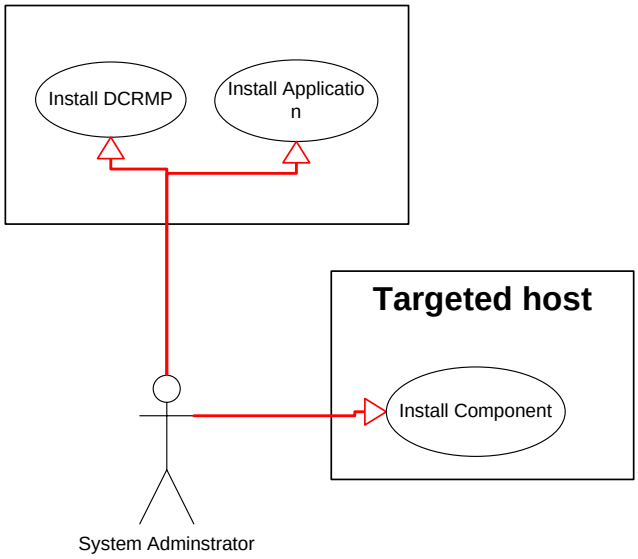

Figure 2 - Main use case diagram

In the following tables a summary of the use cases is outlined.

Install DCRMP	
When is this case performed?	This use case is performed only once, to initialize the DCRMP data repository with common information about the site.
Input:	The system administrator enters global parameters describing the deployment infrastructure. This information will be used during the installation of the DCRMP applications.
Output:	The initialization of the DCRMP data repository.

Table 2 - Install DCRMP use case

Application Install	
When is this step performed?	Every time the System Administrator wants to perform the installation (or the reinstallation) of an application that is distributed through the DCRMP.
Input:	The information that is global to the application to be installed.

Output:	A package for each application component of the Application. Each package is composed of: • All the files needed to deploy the component. • The part of the information, coming from the DCRMP data repository that is needed for configuration of this particular component.

Table 3 - Application install use case

Component Install	
When is this step performed?	After the Application Install. This task has to be performed for each application component. A system administrator's manual intervention might be needed on the target host to accomplish the installation process.
Input:	The configuration parameters relative only to this component.
Output:	The actual installation of the component on the targeted host.

Table 4 - Component install use case

3. Packaging technology

The DEISA grid infrastructure consists of various software tools, which in turn are composed of sub-components and configuration files and which may depend on other software. Because of this complexity, in order to distribute, install, and upgrade groups of software, the adoption of the most adequate technology for package management can result in a significant improvement in deployment time. The packaging tool for the DEISA Grid Infrastructure should satisfy the following requirements:

- portability: the DEISA grid is a heterogeneous environment so it is necessary that the tool is compatible with as many different Operating Systems as possible;
- simplicity: the tool should be user friendly, so that each DEISA system administrator can produce customized distribution packages, concentrating on the contents, with a minimal effort on packaging;
- availability of information about installed packages: because of the potentially high number of packages it is indispensable that the tool allows for the maintenance of a repository containing information about the packages that have been installed. It should also be possible to query this repository;
- dependency checking: this is the capability to verify that all the software dependencies associated with a particular package are satisfied before its installation.

In the following paragraphs an overview of some of the most relevant packaging technologies currently available is presented.

3.1 RPM Package Manager

The RPM Package Manager (RPM) is a powerful command line driven package management system capable of installing, uninstalling, verifying, querying, and updating computer software. Moreover several graphical front-ends to the command line are available on different platforms. Each software package consists of an archive of files along with information about the package such as its version, prerequisites, and a short description [5] [6].

Its main characteristics are:

- portability: it works on most Linux and Unix systems, including AIX 5.x.

- simplicity: there is a uniform way for the user or administrator to install and uninstall programs. It only performs non-interactive installation, making it easy to automate the installation. It allows to use DeltaRPMs, which are RPM's equivalent of a patch file. They combine themselves with installed RPMs to perform updates on software that was installed by RPM. This is a much more convenient way to update RPM-installed software, since DeltaRPM doesn't require the original package to perform the update.
- installed package information availability: working behind the scenes of the package manager is the RPM database. It consists of a doubly linked list that contains all information for all installed RPMs. The database keeps track of all files that are changed and created when a user installs a program and can therefore very easily remove the same files. If the database gets corrupted (which happens easily if the RPM client is killed), the double links ensure that it can usually be rebuilt without any trouble.
- dependency checking: it is able to perform a dependencies check, but only relating to other RPM packages.
- diffusion: RPM supports a broad diffusion on Linux distributions as well as several ports to other operating system (e.g., AIX [15]).

3.2 Grid Packaging Tools

GPT, Grid Packaging Tools, is a portable software packaging system that keeps track of the software installed on a system and manages dependencies between packages. GPT adds metadata to standard tar.gz files, putting more "intelligence" into the build/install/configure/uninstall process [11] [12].

GPT can install binary distributions (precompiled for a specific processor and operating system) or source distributions (where the software is compiled from source code when it is installed). GPT also supports dependency management, re-locatable installations (multiple installs), setup (configuration) awareness, and support for bundles (aggregation of packages).

Its main characteristics are:

- portability: GPT uses the Perl language, so is portable to a wide range of systems.
- simplicity: all packaging data is specified from a source perspective. Packaging data is strongly typed.

- installed package information availability: the data about installed packages is stored in XML files.
- dependency checking: it is able to perform a dependencies check, but only related to other GPT packages.

3.3 MKPKG

Mkpkg is a tool to help software publishers automatically create binary software distribution packages. It automates as much of the process as possible, and it provides functionality that catches most packaging mistakes that cause installed software to fail on user's machines.

Mkpkg is meant to be used after the software is debugged and can install itself on the packager's machine. Given software that is ready for distribution, *mkpkg* helps the publisher develop a description of the software package, including manifests, dependencies, and control scripts [7].

Its main characteristics are:

- portability: it is only compatible with HP-UX systems.
- simplicity: during software packaging the publisher must prepare all the elements needed by the installation tool. For many small packages, this is a very simple process, but for larger packages it can be quite difficult to ensure that all the elements have been properly and completely prepared.
- installed package information availability: its greatest weakness is its data structures for storing package configuration information. Mkpkg uses TCL arrays as the basic data structure container. There are two global arrays: database() and product(). database() contains the system information that is used by all packages created on that system, while product() contains the information relevant to a particular product.
- dependency checking: *Mkpkg* automatically detects all shared library dependencies. It checks every file in the product to discover which shared libraries are used by the product. It has a list of all shared libraries on the system, with the fileset that contains each library. *Mkpkg* then automatically lists each fileset containing the linked shared libraries as a codependency.

3.4 Debian

Debian packages are standard Unix archives that include two gzipped tar archives: one that holds the control information and another that contains the data. The canonical program for handling these packages is dpkg [8].

Its main characteristics are:

- portability: Debian packages managers only work on Linux platforms with a Debian based OS
- simplicity: approximately the same level than RPMs
- installed package information availability: dpkg maintains information about the installed packages
- dependency checking: dpkg is able to perform dependencies checking

3.5 PACMAN

PACMAN (PACkage MANager) is a software distribution tool that makes it considerably easier to deploy a custom distribution of one or many packages onto many systems with consistent installation and configuration settings. PACMAN offers features for obtaining, installing, and updating software distributions from one or more "master" sites.

PACMAN uses a "caching" paradigm, in which the architects of a deployment install and configure software on a set of "caches." Once a cache is established, others can replicate the installation on their own systems by issuing a PACMAN client command. Caches include configuration information and aiding in maintaining common configuration settings. PACMAN also provides an easy mechanism to update sites when changes are made in the cache. PACMAN supports a number of popular packaging systems, including the venerable "tar.gz" format, GPT, and RedHat's RPM [9] [10]. To achieve a higher security level it is also possible to use a secure protocol to transfer file like SCP.

Its main characteristics are:

- portability: FreeBSD, IRIX, Sun, Rocks, SL, SLC, Fermi-SL, MacOS, RedHat, Fedora, Cygwin, RHEL, AIX, SuSE and Debian systems are supported.
- simplicity: PACMAN is able to download arbitrary files and run a set of commands to install the software. The commands can be different for each set of software.

- installed package information availability: there is no repository containing data related to installed packages.
- dependency checking: PACMAN can recognize natively installed software and recursive dependencies are automatically resolved.

3.6 Conclusions

As a result of this initial analysis, RPM and GPT seem the most suitable tools that are able to satisfy the DEISA needs. Mkpkg and Debian packages managers lack portability, while PACMAN can be considered for the distribution, but not for the package building.
In order to limit the effort needed, only one package manager should be selected. For this reason a deeper comparison between RPM and GPT was performed and the following additional technical and practical aspects were taken into account:

1. Package manager dependencies: RPM requires zlib, bzip2, Berkeley DB and gettext libraries. GPT needs a version of perl >= 5.005, zlib, and uses a special set of GNU autotools (automake, autoconf, libtool) for package development. Other versions of these tools can be used, but the macros and scripts for GPT may not be compatible. Therefore both have to satisfy a certain number of constraints.
2. Compilation flavors: only the GPT seems to allow the use of compilation flavors.
3. Backward compatibility: RPM often has backwardly incompatible changes in package format, while GPT grants a better compatibility.
4. Packages signature: RPM is able to use cryptographic verification with GPG and md5, while GPT is not.
5. Dependency checking with packages of different types: neither RPM nor GPT can perform checks against software already installed, native or packaged using different formats.
6. Pre-install, post-install scripts execution capability: RPM manages pre-install and post-install scripts, while GPT only post-install ones.
7. Relocatable packages: both can relocate packages to a specified directory.
8. Last but not least, feedback from the intended users (some representative system administrators of the different DEISA sites) has been the other important factor to determine the package manager. In fact during one of the DEISA SA3

technical meetings[1] it has been clear that the adoption of an only recently developed technology was not well accepted (i.e. GPT), but at the same time the distribution of the software through different package managers would not have been a sustainable effort.

For the above listed reasons it has been decided that the software will be distributed using RPM, because it is by far the most widely used and available distribution system. On the other hand, to allow the usage of systems that do not support RPM, we also distribute the software through compressed archives. In this way it is possible to limit the number of different packages without excluding particular platforms.

[1] Held in Barcelona (Spain) on September 12th and 13th, 2005

4. Common environment definition

At the current development stage the DEISA Grid Service is composed of the following Applications that will be distributed with the DCRMP:

- UNICORE
- RMIS

Obviously they can be installed individually. A detailed description is provided in the next paragraphs accordingly to the following pattern. First of all, the application components mapping within the DCRMP structure is illustrated along with a brief description of their functionalities; then a list of software requirements for the installation and a set of architectural constraints for the deployment is presented; hence it is shown a schema with all the steps that have to be performed by the system administrator in order to deploy the application, followed by the schema containing all the connections that must be implemented among the different sub-components of the applications; finally, a list of configuration files that are involved in the installation process of the application is given.

4.1 UNICORE

UNICORE (UNiform Interface to COmputer REsources) is a vertically integrated Grid middleware for accessing distributed resources in a seamless and secure way [1] [2]. It provides several functionalities like a graphical user interface for job creation and submission, data management, workflow control, and single sign-on.

A UNICORE diagram is shown in Figure 3 where all the middleware components have been collocated within the DCRMP architecture described in the paragraph 2.1.

For completeness a short description of each Sub-Component is provided in Table 5 below, for a more detailed description refer to [3].

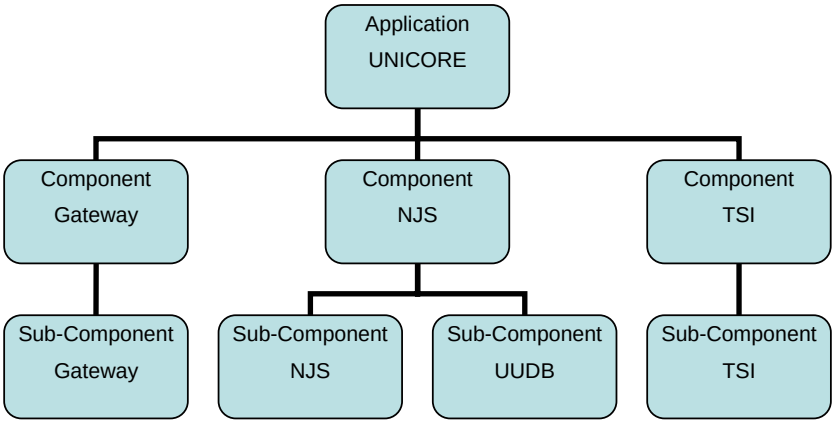

Figure 3 - UNICORE components diagram

Sub-Component	Description
Gateway	The gateway is the single entry point for all the clients' connections to the UNICORE application.
NJS	The Network Job Supervisor is in charge of: • user authorization (via UUDB) • incarnation of the submitted job (it translates AJO into the appropriate command sequence) • job scheduling according to the dependencies
UUDB	The UNICORE User Data Base defines which NJSs are allowed to submit jobs and it keeps track of the *UNICORE-users* to *system-users* mapping
TSI	The Target System Interface is responsible for the interaction with the batch scheduler interface

Table 5 - UNICORE sub-components

4.1.1 *Architectural constrains and software requirements*

Gateway and NJS components have to be deployed on a machine that belongs to some front-end resources of the DEISA site. On the other side the TSI needs to be installed on an HPC machine, in particular, on a node where the job scheduler facility commands are executable. It is possible to aggregate the Gateway and the NJS on the same machine, whereas for security reasons it is recommended to keep NJS and TSI on different

systems. Finally, to achieve a higher security level, the communication between NJS and TSI should be over a private network since it is not encrypted. For a more detailed description of the deployment model for UNICORE refer to [3].

For the current release of the DCRMP, the TSI installation is supported only on AIX clusters where a running LoadLeveler job scheduling facility is available. Following is a list of software requirements for its installation:

- Perl >= 5.004, 5.8.x recommended[2]
- Perl modules (from CPAN):
    - Socket
    - Posi
    - FindBin
    - File::Path
- LoadLeveler [4]

For the Gateway and NJS the only software requirement for the installation is a Java Run-time Edition 1.4.x.

- Gateway
    - Sun JRE 1.4.X
    - an x509 certificate signed by a trusted CA (supported formats are PKCS12 or JKS)
    - a list of x509 certificates containing all the trusted CAs (supported formats are PEM)
- NJS
    - Sun JRE 1.4.X
    - an x509 certificate signed by a trusted CA (supported formats are PKCS12 or JKS)
    - the signer certificate (supported formats are PEM)

4.1.2 *Installation schema*

In order to install UNICORE it is necessary to follow the steps reported in Figure 4. This diagram is the specialization of the general case presented in Figure 1 regarding the deployment of this particular product.

[2] it is needed a recompiled version, with the setruid/seteuid function enabled

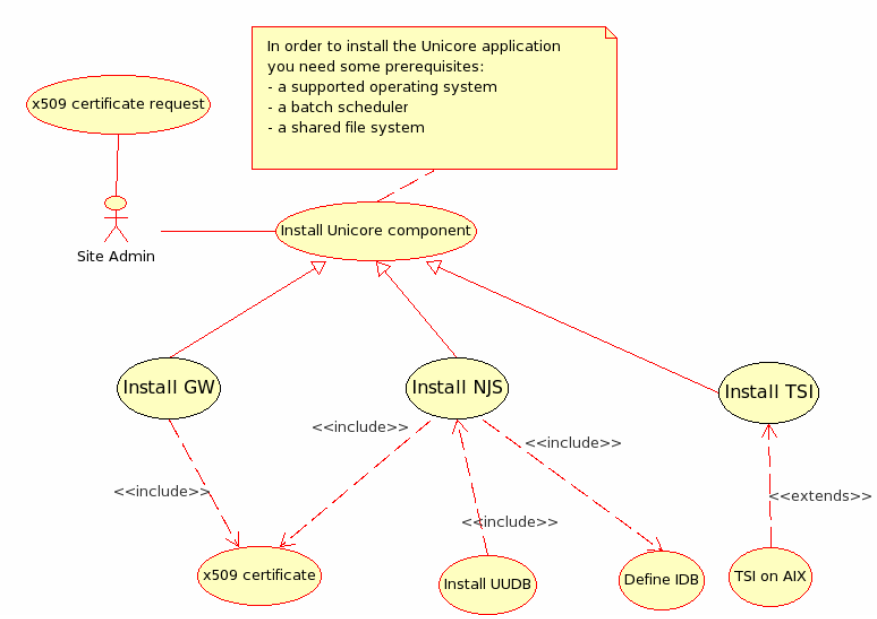

Figure 4 - Installation use cases

Below a short description of the objects in Figure 4 is given.

- x509 certificate request: this use case is performed by the site administrator to request a valid certificate from a trusted CA. Due to security issues this operation must be done necessarily by a human and can not be automated.
- Install UNICORE component: this is the main action to be performed by the system administrator. It is a generalization of the three most important use cases: “Install GW”, “Install NJS” and “Install TSI”.
- Install GW: represents the installation of the UNICORE Gateway component.
- Install NJS: represents the installation of the UNICORE NJS component.
- Install TSI: represents the installation of the UNICORE TSI component.
- x509 certificate: both Gateway and NJS component requires a valid x509 certificate issued by a trusted CA. Moreover for the Gateway and NJS the list of the trusted CAs used to accept external encrypted connections must be defined.
- Install UUDB: this use case represents the installation process of the UNICORE User Data Base, required for the user authorization and the mapping from UNICORE users into UNIX users. This is a sub-component needed by the NJS.

- Define IDB: this database is used by the NJS to map UNICORE high-level commands into target system specific command invocations.
- TSI on AIX: The installation of the TSI component on AIX systems requires some additional steps. The BecomeUser.pm Perl module uses the system calls setruid/seteuid to swap temporarily the real and effective user IDs to that of the consignor of a job. In the standard distribution of Perl (which is part of AIX) the functions setruid/seteuid are disabled by default. The solution is to recompile Perl, enabling the setreuid/setregid system calls in the configuration process.

4.1.3 *Connection topology*

The connection topology of the UNICORE components can be summarized in the following way:

- From client to Gateway
- From Gateway to NJS (AJO consign)
- From NJS to Gateway (NJS registration)
- From NJS to TSI (job submission)
- From TSI to NJS (job submission)

In Figure 5 a schema summarizing the connections involved among the UNICORE components is shown.

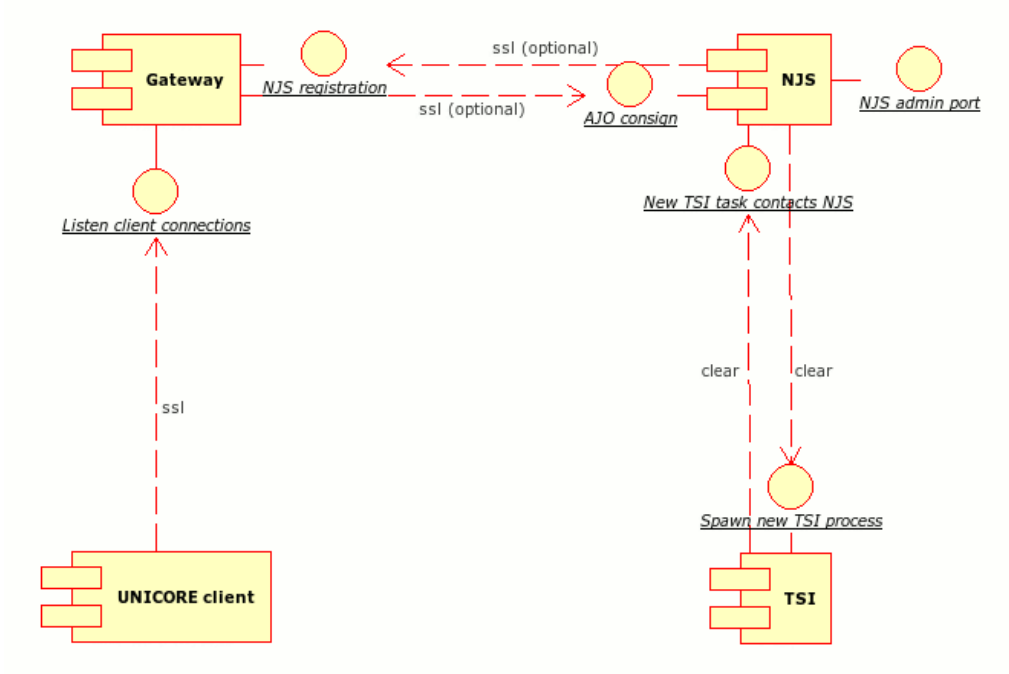

Figure 5 - UNICORE connection schema

4.1.4 *Configuration files*

The installation of the UNICORE Application involves the configuration of the files described in the table below.

File name	Component	Description
njs.properties	NJS	They define the principal configuration parameters of the respective components
gateway.properties	Gateway	
tsi.properties	TSI	
example.idb	NJS	It contains the instruction to incarnate a Job on a particular platform

Table 6 - UNICORE configuration files

4.2 RMIS

RMIS (Resource Management Information System) is a system able to deliver up to date and complete resource management information over the Grid. It should provide relevant information to system administrators.

An RMIS diagram is shown in Figure 7 where all the application components have been allocated within the DCRMP architecture described in the paragraph 2.1.

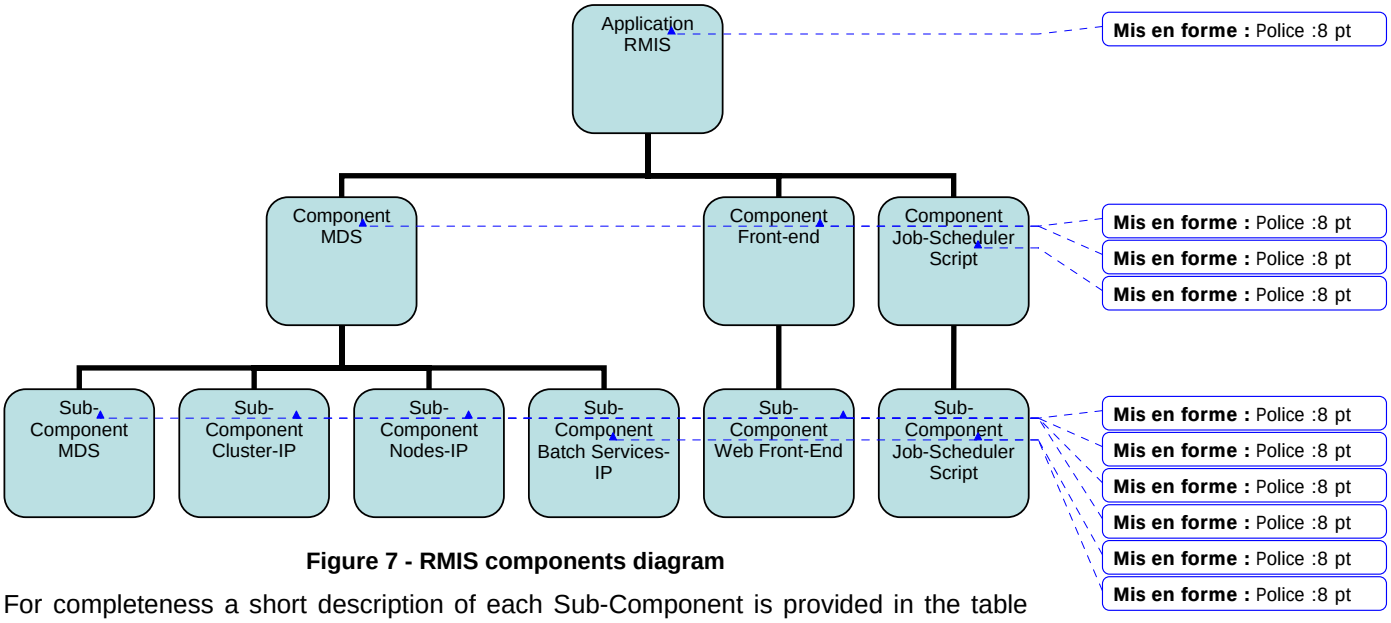

Figure 7 - RMIS components diagram

For completeness a short description of each Sub-Component is provided in the table below, for a more detailed description refer to [3].

Sub-Component	Description
MDS	The Monitoring and Discovery System is a LDAP-based implementation of information services
Cluster IP	The Cluster Information Provider is in charge of supplying information about the whole cluster to the MDS
Nodes IP	The Nodes Information Provider is in charge of supplying information about each node of the cluster to the MDS
BatchService IP	The BatchService Information Provider is in charge of supplying information about the job scheduler to the MDS
Web Front-End	The Web Front-End is a PHP Web interface to the data provided by the RMIS.

Job Scheduler script	This is a single Perl script that is in charge of providing job scheduler information to the RMIS.

Table 7 - RMIS sub-component description

4.2.1 *Architectural constraints and software requirements*

The Web Front-End and MDS components have to be deployed on a machine that belongs to the front-end resources of a DEISA site. The Job Scheduler script needs to be installed on an HPC machine, in particular on a node where the job scheduler commands are executable. It is possible to aggregate the Web Front-End and the MDS on the same machine.

Finally, to achieve a higher security level, the communication between Web Front-End and Job Scheduler script and between MDS and Job Scheduler script should be over a private network since they are not encrypted.

At the current release of the DCRMP, the Job Scheduler script installation is supported only on AIX clusters where a running LoadLeveler job scheduling facility is available. The following is a list of software requirements for its installation :

- Perl >= 5.8.x
- Perl modules (from CPAN):
    - Sys::Hostname;
    - Getopt::Long;
- LoadLeveler Multi Cluster [4]

For the Web Front-End the software requirements for the installation on a front-end machine are a PHP enabled Web Server, while MDS requires only PHP.

Both need the Ganglia monitoring toolkit (>= 3.0.0) installed on a HPC machine.

- MDS
    - an x509 certificate signed by a trusted CA (supported format is PEM)
    - a list of x509 certificates containing all the trusted CAs (supported formats are PEM)
    - PHP >= 4.3.x
- Web Front-End
    - a list of x509 certificates containing all the trusted CAs (supported formats are PEM)

- a x509 certificate signed by a trusted CA (supported formats are PEM)
- Web Server
- PHP >= 4.3.x

4.2.2 *Installation schema*

In order to install RMIS it is necessary to follow the steps reported in Figure 8. This use case diagram is the specialization of the general case presented in Figure 1 regarding the deployment of this particular product.

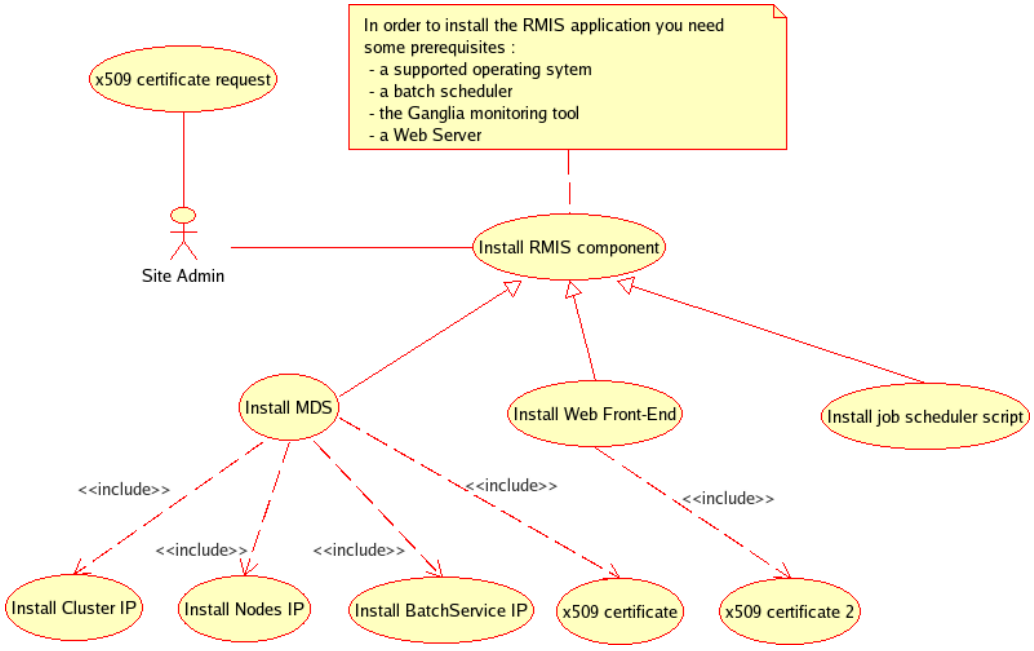

Figure 8 - Installation use cases

Below a short description of the objects in the Figure is given.

- x509 certificate request: this use case is performed by the site administrator actor to request a valid certificate to a trusted CA. Due to security issues this operation must be done necessarily by a human and can not be automated.
- Install RMIS component: this is the main action to be performed by the system administrator. It is a generalization of the three most important use cases: “Install MDS”, “Install Web Front-End” and “Install Jobs Scheduler script”.
- Install MDS: represents the installation of the RMIS MDS component.

- Install Web Front-End: represents the installation of the RMIS Web interface component.
- Install Jobs Scheduler script: represents the installation of the RMIS Jobs Scheduler script component.
- x509 certificate: MDS component requires a valid x509 certificate issued by a trusted CA. Moreover for the MDS the list of trusted CAs used to accept external encrypted connections must be defined.
- x509 certificate 2: Web Front-End component requires a valid x509 certificate issued by a trusted CA. Moreover for the Web Front-End the list of trusted CAs used to accept external encrypted connections must be defined.
- Install Cluster IP: represents the installation of the cluster Information Provider sub-component, needed by MDS. It is required in order to collect the cluster's workload and administrative data and provide them to the MDS LDAP data base.
- Install Nodes IP: represents the installation of the nodes Information Provider sub-component, needed by MDS. It is required in order to collect a single node's workload and configuration data and provide them to the MDS LDAP data base.
- Install BatchService IP: represents the installation of the batch scheduler Information Provider sub-component, needed by MDS. It is required in order to collect the batch scheduler's load and configuration data and provide them to the MDS LDAP data base.

4.2.3 *Connection topology*

The connection topology of the RMIS components can be summarized in the following way:

- From MDS client to MDS
- From Web browser to Web Front-End
- From MDS to job scheduler script
- From Web Front-End to job scheduler script

In Figure 9 a schema that summarizes the connections involved among the RMIS components is shown.

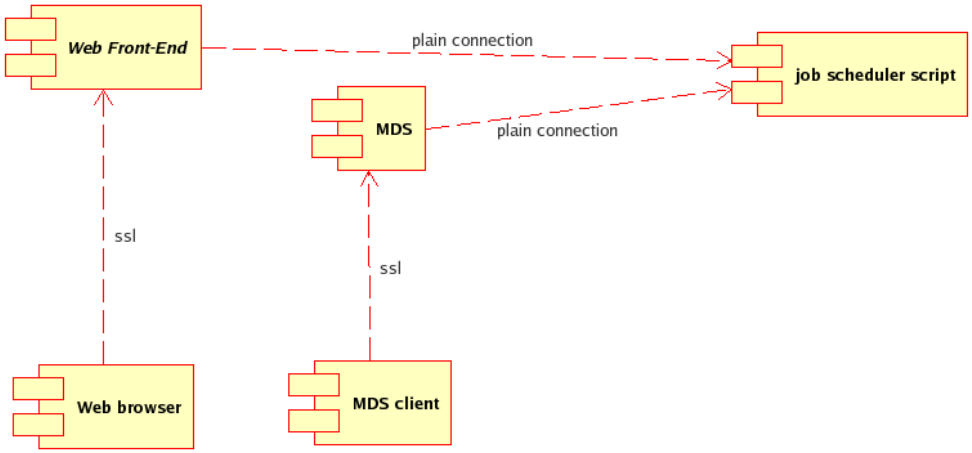

Figure 9 - RMIS connection schema

4.2.4 *Configuration files*

The installation of the RMIS Application involves the configuration of the files described in the table below.

File name	Component	Description
conf.php	Web Front-End	They define the principal configuration parameters of the respective components
grid-info-resources-ldif.conf	MDS	
grid-info-slapd.conf	MDS	
deisa.schema	MDS	
deisa.main.conf	MDS	It defines the static configuration of the cluster
deisa.cluster.conf	MDS	It defines the static configuration of a group of nodes

Table 8 - RMIS configuration files

5. Implementation details

This chapter describes the current DCRMP implementation. Today DCRMP has a quite stable version, but its implementation will be probably enhanced.

The DCRMP four main components are:

- Graphical User Interface: a user interface that is used to set the various configuration parameters needed to deploy the application.
- Configuration Engine: an engine that creates all the configuration files according to the parameters previously provided to the GUI.
- Package Repository: a repository that contains all the files that need to be deployed through the DCRMP.
- Package Creator: a set of bash shell scripts that are used to create the deployable packages with the output produced by the Configuration Engine.

The GUI and the Configuration Engine are implemented through the usage of a software tool called Gridconfig that is described in the next paragraph.

The Package Repository is just a directory containing the original packages of the current middleware software distribution (RMIS, UNICORE).

Finally the implementation of the Package Creator is performed using a set of bash scripts that are described in the paragraph 5.2.

5.1 Gridconfig

Gridconfig is a software tool that is publicly available as part of the NMI Distribution [13]. It is written in Python and can create configuration files in their native formats. It helps to ensure consistency of configuration parameters among different components. Moreover it is shipped with a GUI that is used to retrieve the input from the user.

At the current development stage Gridconfig is used as a GUI and as a configuration engine. It has been customized to configure all the files listed in chapters 4.1.4 and 4.2.4.

5.2 Package Creator

The Package Creator component is responsible for wrapping up all the previous components into one single script. Its main duties are:

1. Starting Gridconfig

2. When Gridconfig has produced the configuration files it starts sequentially a script for every application (i.e., one for UNICORE and one for RMIS). These scripts, in turn, perform the following steps for every component:
    a. copy the configuration files into the component directory structure
    b. merge the archive of more sub-components if necessary
    c. create an output package for every component that can be deployed on the targeted host.

At the current development stage these scripts are implemented as shell scripts, using only the most standard commands in order to achieve the maximum degree of portability.

Actually a 'dot bin' package is produced and can be executed on the targeted host directly. It is a unique file composed of a shell script and a tar.gz archive. A shell script is in charge of uncompressing and installing all the files in the chosen directory.

5.3 Distribution system and software requirements

The DCRMP is distributed as a tar.gz file that, once extracted, can be used directly without further configuration. This archive already contains Gridconfig preconfigured for the DEISA middleware, plus all the files needed to install the middleware. Its directory structure is described in the table below:

Directory	Contents
Bin	All the executable script of the DCRMP
Deployment	The output of the DCRMP (a set of packages that has to be executed on the target host)
Etc	Configuration files
Gridconfig	All Gridconfig files
Products	The packages of the applications that are distributed with the DCRMP
Temp	Temporary working directory

Table 9 - Distribution system directory structure

An archive containing only the Gridconfig application configured for the DEISA middleware without the software files will also be made available. In this way system administrators will be able to use the DCRMP with their preferred 'flavour'.

To execute the DCRMP, a host is required to have installed:

- Python
- Shell with the following commands: chmod, mkdir, rm, mktemp, tail, cut, awk, cd, and echo.

5.4 Gridconfig: next steps

The DCRMP is still under development and for this reason feedback from users is used to drive the implementation of new features. Possible extensions are:

1. Add some features to Gridconfig so that it will be able to save and restore a previous configuration in a more flexible way (so that users will be able to keep a history of previous configurations)
2. Improve the quality of the scripts that generate the package (Package Creator) possibly integrating them with Gridconfig
3. Modify Gridconfig so that it will also produce RPM packages
4. Generate documentation to explain how to distribute an application with DCRMP

The feasibility study of these features will be undertaken during the following months in order to come up with a consistent roadmap. Moreover, the requirements will be analysed and the ones that have highest priority will be implemented in the next releases.

6. Roadmap and future direction of the DCRMP task

In the next months the DCRMP activity will concentrate its effort on the development of additional software tools to further simplify the deployment and maintenance of the DEISA middleware.

The software will be divided into two categories in future release:

- 'site-level' tools: programs able to ease the deployment and maintenance of software that manipulates data relative to the site configuration
- 'DEISA-level' tools: programs that perform software deployment and maintenance using data that belongs to an inter-site level in other words to the DEISA infrastructure

For instance, the DCRMP tool described in this document creates a data base of the configuration and deployment status at site level. In one of the next releases, other tools able to perform configuration checking at DEISA level will be included. In fact several repositories already exist while others may be created, and with the help of DEISA specific scripts these data sources can be used to automate configuration and deployment steps in a DEISA site. For example the LDAP infrastructure is already available at each site and can be used to generate automatically the UUDB for UNICORE. Some configuration parameters of UNICORE components (e.g. trusted NJS and their certificates) can be automatically downloaded from a central data source into the configuration files of a site.

The future execution plan is divided into two phases:

1. the first phase extends until month 24 when several software tools will be released and when feedback from users will be collected and analysed
2. the second phase starts in month 24 and extends until month 30 when only the most useful and successful software tools will be selected and moved to a production level with the addition of extra features and documentation

Within this context, the DCRMP will play a central role in promoting and distributing these tools, and in gathering user's feedback and requirements. To this extent a set of web pages will be used to present DCRMP software tools while email and technical meetings will be the official means for collecting user requirements.

DCRMP will become the main repository of scripts and software tools able to ease the deployment and maintenance of DEISA middleware.