



CONTRACT NUMBER 508830

DEISA
**DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR
SUPERCOMPUTING APPLICATIONS**

European Community Sixth Framework Programme
RESEARCH INFRASTRUCTURES
Integrated Infrastructure Initiative

Basic DEISA Infrastructure Documentation

Deliverable ID: DEISA-DSA4-2

Due date : January, 31, 2005

Actual delivery date: February, 21, 2005

Lead contractor for this deliverable: IDRIS-CNRS, France

Project start date : May 1st, 2004

Duration: 5 years

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

Executive summary

The main objective of this service activity is to deploy all the actions required to enable the scientific users adoption and utilisation of the DEISA supercomputing infrastructure. This is mainly done by providing and maintaining a *Common Production Environment* on all the platforms of the infrastructure, and providing also documentations of its usage, training sessions to help the port and the optimisation of the applications and an *Help Desk* service.

The content of this document is the *DEISA Primer*, which constitutes the basic DEISA infrastructure documentation. It includes all the informations useful to the users to know which procedures must be followed to access the infrastructure, how to obtain an account, how the files can be stored and handled, how to set up the software environment using the *Common Production Environment* and how to submit jobs.

The *DEISA Primer* is available on the DEISA Web site, both as a single printable file and as an on-line document, with can be consulted interactively.

This document is publicly available.

Table of Content

1.	Introduction	1
1.1	Grid architecture	1
1.2	Services	2
2.	Access to the DEISA infrastructure	3
2.1	Projects	3
2.2	Accounts	4
2.3	Certificates	4
2.4	Access procedures	5
2.4.1	SSH	5
2.4.2	Batch managers	5
2.4.3	UNICORE	5
2.5	Passwords	5
3.	File Systems	5
3.1	Organization	6
3.1.1	Home Directory (\$HOME)	6
3.1.2	\$DEISA_HOME	6
3.1.3	\$DEISA_DATA	6
3.1.4	\$DEISA_SCRATCH	7
3.2	Usage	7
3.3	Services	7
4.	Common Production Environment	7
4.1	Definition	7
4.2	Components	8
4.3	Modules framework	9
4.4	Characteristics of libraries, tools and applications	10
4.4.1	Multibyte numeric representation	11
4.4.2	Addressing mode	11
4.4.3	Size of Fortran float numbers	12
4.4.4	Size of integers	13
4.5	Modules usage	14
4.5.1	Initialization	14
4.5.2	Help	15
4.5.3	List of the modulefiles currently loaded	15
4.5.4	List of the available modulefiles	16
4.5.5	Help on a specific modulefile	16
4.5.6	Location of a modulefile and its brief description	17
4.5.7	Detailed description of a modulefile	17
4.5.8	Loading a modulefile	18
4.5.9	Unloading a modulefile	19
4.5.10	Switching between two versions of a modulefile	19
4.6	Compiling and linking	20
4.6.1	Compiler wrappers	20
4.6.2	Compiling and linking with libraries	20
4.6.3	Complete example	21
4.6.4	Error messages, known problems, and side effects	22
5.	Job submission	23

5.1	Job submission to a batch subsystem	23
5.1.1	Job submission using LoadLeveler on the DEISA IBM sites	23
5.1.2	Batch jobs for compilation and pre and post-processing.....	25
5.2	Job submission through UNICORE.....	25
5.2.1	UNICORE client configuration and certificate management.....	27
5.2.2	Example of a UNICORE job	28
5.3	Job submission through Web portals	37
6.	Access to the documentations.....	37
7.	Access to the User Support	37
8.	Quick Reference.....	38
9.	Glossary	38
10.	References.....	39
11.	Index.....	39

1. Introduction

The DEISA project is fully described in the document *The DEISA Supercomputing Grid Infrastructure* available on the DEISA Web site to which you must refer for a complete general overview of its objectives, organization, infrastructure and operational model.

1.1 Grid architecture

The architecture of the DEISA supercomputing grid integrates the national resources at two levels:

- ? **An inner level, providing the deep integration and strongly coupled operation of similar, homogeneous platforms.** Here, national IBM AIX clusters are glued together to constitute a *distributed European supercomputer*, called “*the AIX super-cluster*”

The initial phase involves four IBM Power4 platforms:

- FZJ-Jülich (Germany): P690 (32 processor nodes) architecture, incorporating 1312 processors. Peak performance is 8.9 Teraflops.
- IDRIS-CNRS (France): Mixed P690 and P655+ (4 processor nodes) architecture, incorporating 1024 processors. Peak performance is 6.7 Teraflops.
- RZG–Garching (Germany): P690 architecture incorporating 896 processors. Peak performance is 4.6 Teraflops.
- CINECA (Italy): P690 architecture incorporating 512 processors. Peak performance is 2.6 Teraflops.

Thereafter, a fifth system will be added to the super-cluster:

- CSC (Finland). P690 architecture incorporating 512 processors. Peak performance is 2.2 Teraflops.

- ? **An outer level, providing a looser federation of heterogeneous supercomputing resources.** This constitutes a heterogeneous grid of supercomputers and super-clusters. Indeed, this heterogeneous supercomputing grid includes all the leading platforms in Europe exhibiting different technologies from different vendors like IBM, SGI, NEC. In this context, the AIX super-cluster is seen as a single platform.

- The AIX super-cluster (FZJ, RZG, IDRIS, CINECA, CSC) , with an aggregated computing power close to 25 Teraflops.

- BSC (Barcelona, Spain): 4564 Power PC 970 processor IBM Linux system, 9 TB memory space, 233 TB disk space. Peak performance is 40 Teraflops.
- HLRS (Germany): NEC SX8 vector supercomputer, 576 processors, 9.2 TB memory space, 180 TB disk space. Peak performance is 12.67 Teraflops.
- LRZ (Germany): Linux cluster with more than 500 CPUs (above 2.7 Teraflops peak performance) (*see below for the planned evolution of LRZ's participating systems*)
- SARA (The Netherlands): SGI Altix 3700 Linux system, 416 Itanium-2 processors, 832 GB main memory, peak performance 2.2 Teraflops.
- ECMWF (International organization): Two P690+ clusters with 68 32-way nodes each, with an aggregated performance of 33 Teraflops peak.

The planned evolution of LRZ's supercomputing platforms participating in the DEISA pool is:

- Mid 2006: SGI supercomputer with 5120 processor cores, 20 TByte main memory 350 TB disk space. Peak performance is 33 Teraflops.
- Mid 2007: SGI supercomputer with 6656 processor cores, 40 TByte main memory, 680 TBytes disk space. Peak performance is close to 70 Teraflops.

Other leading systems (at EPCC, UK) are not directly involved in the European resource pool, but can be engaged in some specific DEISA operations.

1.2 Services

Each of these national supercomputing centres is primarily focused on scientific research, even if some have also industrial users. All offer both basic and extended services to their users:

- ? help desk,
- ? local documentation,
- ? Web sites with site-specific information,
- ? training classes,
- ? technical and scientific workshops,
- ? help of computer specialists to port, parallelize and optimise codes, specially on a large number of processors.

Even though all the sites have a long and profound experience in all the topics of High Performance Computing, each of them has additional special competencies in selected application areas:

-
- ? CINECA: computational fluid dynamics, oil and gas applications, industry support in general,
 - ? CSC: coupled applications, scientific support by scientists in various fields,
 - ? EPCC: tera-scaling (optimisation of applications on a large number of processors), astrophysics, quantum chromo dynamics,
 - ? FZJ: computational chemistry, computational physics, methods for long range interactions, quantum chromo dynamics, virtual reality, steering, performance tools,
 - ? IDRIS: climate modelling, computational fluid dynamics, bio-informatics, coupled applications,
 - ? LRZ: computational fluid dynamics, astrophysics, computational chemistry, quantum chromo dynamics, high performance networks, virtual reality, computational steering,
 - ? RZG: materials science, plasma physics, astrophysics, computational biology, computational steering,
 - ? SARA: life sciences, bioASP (application service provider in bioinformatics), virtual reality, high performance networks.

2. Access to the DEISA infrastructure

This chapter explains how to access to the DEISA Supercomputing Grid Infrastructure. The process required to obtain an account and the access procedures to the sites are also defined.

2.1 Projects

The DEISA resources are provided by the DEISA sites. Each of the DEISA partners participating to the DEISA Grid provides a fraction of its computational resources (in general, more than 10%) to a DEISA resource pool which is globally managed.

All the national supercomputing centres have evaluation committees that allocate the national resources on the basis of scientific relevance and excellence, and computer science competence of the teams. For the DEISA resources, the basic idea is the following:

- ? The national evaluation committees decide **which** research projects acquire the DEISA label, on the basis of scientific relevance and excellence.
- ? The DEISA Consortium decides **how and when** the DEISA resources are used. It establishes priorities among the project validated by the national evaluation committees, and decides on the way the resources will be managed in each case.

2.2 Accounts

If you attend to access the DEISA infrastructure, you must request an account via the user administration at your home site. It is not possible to use an existing account, because DEISA accounts are unique and designed for the DEISA environment. This DEISA account will consist of 8 digits beginning with an abbreviation of the site which created it. This account will be available at all other DEISA sites automatically. Examples are: cne0cin2, fzj901zm, idr007is, rzg00ifw.

2.3 Certificates

A Public Key Infrastructure (PKI) has been set up in DEISA in order to provide authentication and confidentiality in the communication with entities like servers or users.

A PKI is based on two different types of keys which can be applied for data encryption, digital signatures and non-repudiation of messages. A public key is used to encrypt data that can be decrypted with the corresponding private key. A private key can also be used to sign a message and its signature can be checked by its corresponding public key.

The binding between the identity of the key owner and the public key is done in a certificate where the public key is combined with information about the owner of the certificate. In order to trust the content of this certificate, it will be signed with the private key of a trusted party, called the Certification Authority (CA). The validation of the signed certificate is accomplished using the public key of the CA, which is contained in the self-signed certificate of the CA.

It is important that the signing of the certificates is trustworthy because the certificates are used for authentication and authorisation services within the DEISA infrastructure. Certificates can be considered as the passports for access to the infrastructure. They authenticate the user or server if the user or server presents a message signed with the private key.

In order to use the DEISA infrastructure, you will need a user certificate. To get such a certificate, you have to request it at the Registration Authority related to your site. The details concerning the generation and submission of a certificate request depend also on the procedures that the national CA of your country requires. You can find more information on these procedures at the Web sites of the CAs which you can find in Table 1: E-mail and Web sites of the Certification Authorities of the DEISA sites or at the Web site of your local DEISA site. The private key linked to your certificate must be securely stored and is protected by a password, which should be known solely to yourself.

Table 1: E-mail and Web sites of the Certification Authorities of the DEISA sites

Site	E-mail of the CA	Website of the CA
CINECA	ca-grid@cineca.it	http://grid.cineca.it/docs/d66a2cb7.0
CSC	ca@nbi.dk	http://hep.nbi.dk/CA/
ECMWF	calldesk@ecmwf.int	https://w3cert.ecmwf.int
EPCC	ca-operator@grid-support.ac.uk	http://www.grid-support.ac.uk/ca/
FZJ	projects-ca@fz-juelich.de	https://projects-ca.fz-juelich.de
IDRIS		http://igc.services.cnrs.fr/GRID-FR/
RZG	rzg-ra@rzg-mpg.de	http://www.rzg.mpg.de/ca/

SARA	ca@nikhef.nl	http://certificate.nikhef.nl
------	--	---

Please, note that the CA for the German sites will probably change soon to the DFN-CA: <http://www.dfn-pca.de/>.

2.4 Access procedures

There are three ways to access the DEISA resources: *ssh*, batch managers, and UNICORE.

2.4.1 SSH

You can login interactively to your DEISA home site by secure shell (*ssh*) using *ssh* key pairs.

2.4.2 Batch managers

After login at your home site, you can use the local batch manager to submit a batch job using, for example, the *LoadLeveler lsubmit* command on AIX systems. Depending on the choices of the DEISA job balancing mechanism, the job will be executed locally or at another DEISA site (see paragraph 5.1.1).

2.4.3 UNICORE

The third way to access the DEISA resources in order to submit or create a job is UNICORE. To use UNICORE, you need to obtain a valid certificate following the instructions in paragraph 2.3. The configuration of the UNICORE client is described in detail in paragraph 5.2.1.

2.5 Passwords

Since you can start an interactive session only from your home site, your accounts at the other sites are transparently generated using random passwords, that you will never need.

3. File Systems

In order to provide the user with a consistent view on the data at each DEISA-site, beside \$HOME three additional environment variables have been defined to properly access DEISA file systems in a functional way with respect to possibly different policies. These variables may point to different file systems at some sites, while they may point to just one file system at another site. But in order to allow each user to use the same syntax in the job-definition, the environment variables should be used as defined below.

Although this might look complicated at first sight, it is in fact a simple, transparent and flexible way to consistently handle jobs across all DEISA sites.

In general, four different types of file systems have been identified. The file systems can be either local or non-local. The non-local file systems may be available only on a specific subset of hardware or accessible from all computers participating in DEISA.

There are also different possible policies on the file systems. There may be a quota limiting the available file space to something less than the physically possible maximum on some file systems. On some of the file systems there may be a backup, while on other parts of the file system space may be available only during a single job and be cleared on exit.

Despite all these differences there are well defined ways on how you as a DEISA user can access the different file systems.

3.1 Organization

3.1.1 Home Directory (\$HOME)

On each DEISA system you will have a home-directory, which is the normal starting place after login. You will not be allowed to login on another site than your home site, but you will always have a home directory. The location of the home directory may vary from site to site and even from machine to machine on each site. Usually, the home directory at your home site will be backed up and have quotas. But as you see, this location, addressed via the environment variable `$HOME`, is not a place where programs or data should reside.

3.1.2 \$DEISA_HOME

Since the home directory may vary from computer to computer, DEISA has introduced an environment variable `$DEISA_HOME`, which is guaranteed to point to the same physical location at least on each homogeneous hardware. Please, be aware that the path to this location may vary from site to site, so always use this environment variable to address data in this place.

The data located on `$DEISA_HOME` is normally physically located at your home site but accessible on each site. On the AIX machines, this file system is implemented using the Multi-Cluster GPFS of IBM. `$DEISA_HOME` is usually quite restricted in size by having defined quotas, because it is guaranteed to be backed up often.

3.1.3 \$DEISA_DATA

Since the file space in `$DEISA_HOME` may be very limited, there is another file system, which can be accessed using the environment variable `$DEISA_DATA`. This file system is also available on each computer and the location defined with this environment variable is also guaranteed to point to the same physical location at least on each

homogeneous hardware. But, as for `$DEISA_HOME`, it is also true for `$DEISA_DATA` that the path may vary from site to site.

The data is also physically located on your home site but accessible on each site. Whether the file system is backed up or not is depending on the policy of your home site. There may be also some site dependent policy on cleaning up by removing older files.

3.1.4 `$DEISA_SCRATCH`

Since the files in `$DEISA_DATA` are accessed over the DEISA-network, if your job does not run at your home site, jobs producing large amounts of output require some fast (local) disk space. This disk space can be found using the `$DEISA_SCRATCH` environment variable. Please, note that it may be possible that each job has a different `$DEISA_SCRATCH` and that the data therein may be deleted after a job has ended.

Therefore, this file system is only useful for fast I/O operations from within a job. Data located in this file system is not persistent and has to be copied to `$DEISA_DATA` or `$DEISA_HOME` at the end of the job.

3.2 Usage

As described in the previous section of this chapter, there are four different DEISA file systems. Accessing data in these file systems consistently over all sites will work only if you do not use absolute path names but always use the environment variables provided above to address the location of data or programs.

If you compile your own programs, never put absolute path names into your code. Create output in and read input from the current directory, or use the values of the three environment variables from within your executable or Makefile.

3.3 Services

The quota and backup rules may vary from site to site. It is guaranteed that the data located in `$DEISA_HOME` is backed up regularly. Details of the policies for the single file systems can be found on the DEISA Web site.

4. Common Production Environment

4.1 Definition

To ensure a high level of coherence of the software environment in the DEISA supercomputing infrastructure, especially within the subgroups of homogeneous computers which share the same hardware and software architecture, a *Common*

Production Environment (CPE) has been defined and deployed on each computer integrated in the platform.

The Common Production Environment includes:

- ? *shells* (Bash and Tcsh),
- ? *compilers* (C, C++, Fortran and Java),
- ? *libraries* (for communication, data formatting, numerical analysis, etc.),
- ? *tools* (debuggers, profilers, editors, batch and workflow managers, etc.),
- ? *and applications*.

The components currently supported are listed in Table 2.

4.2 Components

The following table lists all the components currently available, specifying those which belong only to a subgroup of computers, and also the commercial and licensed software, which may be available only on one or a few sites. In such cases, the re-routing of batch jobs will obviously be limited to the sites where the required softwares are available.

Naturally, this list of components is evolving and is enriched according to the requirements of the users and the decisions of the administrators. The supported applications are up to now restricted to the public domain ones used by the scientific Joint Research Activities.

Table 2: Current software components of the Common Production Environment

Software	Class	Description	Restrictions
SHELLS			
bash	shell	Bourne Again Shell (Bash)	
tcsh	shell	Extended C Shell (Tcsh)	
COMPILERS			
c	compiler	C compiler	
c++	compiler	C++ compiler	
fortran	compiler	Fortran compiler	
java	compiler	Java compiler	
LIBRARIES			
essl	numerical	IBM Engineering and Scientific Subroutine Library	IBM only
fftw	numerical	Fast Fourier Transform library	
hdf5	data format	Hierarchical Data Format (version 5)	
lapack	numerical	Linear Algebra PACKage	
mass	numerical	IBM basic numerical library	IBM only
mpichg2	MPI	MPI library (from the Globus toolkit)	
nag	numerical	Numerical library from the Num. Algorithms Group	commercial
netcdf	data format	Network Common Data Format	
pacxmpi	MPI	Parallel Computer eXtension to MPI	
pessl	numerical	IBM Parallel ESSL	IBM only
scalapack	numerical	SCAlable Linear Algebra PACKage	

wsmpp	numerical	IBM Watson Sparse Matrix Package	IBM only commercial
TOOLS			
emacs	editor	Text editor	
globus hpm	grid toolkit profiler	Environment for the Globus toolkit IBM Hardware Profiler Monitor	IBM only
loadleveler	batch	IBM batch system	IBM only
nedit	editor	Text editor	
omniorb	utilities	CORBA implementation	
openssh	utilities	Secure shell	
openssl	utilities	Secure Sockets Layer toolkit	
perl	utilities	Practical Extraction and Report Language	
poe	utilities	IBM Parallel Operating Environment	IBM only
python	utilities	Development language	
tcl	utilities	Tool Command Language	
tk	utilities	GUI toolkit for Tcl	
totalview	debugger	Etnus parallel debugger	commercial
unicore	workflow	Uniform Interface to COmputing REsources	
xgrafx	graphics	Interactive display of results	
APPLICATIONS			
cpmd	chemistry	Molecular dynamics (Car-Parrinello)	
orb	plasma phy	Turbulence	
wien2k	chemistry	Electronic structure calculations of solids	

For each component the current *default* version is available in this way. In fact, several other versions of each component (either older or newer) are generally accessible. The default version is the one without a number, and the other ones are suffixed by a number (like *fortran-9.1.0.0*). We will describe below how to find out which ones are available and how to use a version other than the default one.

Please, note that for each subgroup of homogeneous computers, the versions are the same. This is of course essential to allow the user's applications to have the same behaviour on the different homogeneous platforms. To guarantee this characteristic, the upgrade of a component (that is to say changing the default version for a newer one) or the removal of an old one, is always done in a coordinated and synchronized way on all the relevant sites. Please, note also that the default versions of a component can be different between the subgroups of homogeneous computers (for instance the default version of the NAG library can be different between the AIX IBM and HPC Linux SGI platforms, the Netcdf library can be different between the AIX IBM and Super-UX NEC platforms, etc.).

4.3 Modules framework

The Common Production Environment offers:

- ? a common interface to access different software components on different subgroups of homogeneous computers, hiding the possibility that different files of a component would be named or installed in different locations depending on local policies of the sites, and on software packaging for different computer types,

- ? a common interface, for each group of computers, which is available from all the shells, allows people using different shells, or even a user who wants to change his or her shell temporarily or permanently, to use exactly the same commands to access a certain software,
- ? an individual management of each component allows people to load into the user environment only the required software and also to be able to change the version of each independently of the others,
- ? an easy way, for each user independently, to switch between the current default version of a software to an other one. It could be an older one, if for instance some updates are not yet done in a user code after some incompatible changes in the interface of a library used, or if special bugs are discovered in the current version of a software. Or it could also be a new one, to test and validate it, or just to benefit from its new features, before it becomes the current version.

The *Modules* tool is used to offer these features and to define the portable interfaces to the components of the CPE, to set it up according to your needs and to allow you to dynamically enrich your environment during a session.

The *module* command is mainly used to list, load and unload the special files which define each component (see paragraph 4.5 for details). These files are called *modulefiles*, which are written by the system administrators. From the information in a *modulefile*, the *module* command configures the user's environment, allowing you to run the corresponding software. In effect, the *module* command alters or sets common Unix shell environment variables such as \$PATH, \$LD_LIBRARY_PATH or \$LIBRARY, \$MANPATH, etc. and the specific variables of the application. All the required definitions for an application are modified in a coordinated way, which prevents from forgetting some required changes or making incompatible ones.

The syntax rules for the *modulefiles* are:

modulename[-*version*]

where *modulename* is the name of the *modulefile* and *version* is the version number,

- ? all names are in lowercase,
- ? the names are alphanumeric characters (*mode64*, *fortran*, *cpmd*, etc.),
- ? when a version number exists, it is placed after the name of the default version with a minus sign as separator. The version number can have several components, each being a set of alphanumeric characters separated by a dot, to specify the major and minor version numbers, the patches level, etc. (*fortran-9.1.0.0*, *cpmd-3.9.1*, *gaussian-03.b04*, *openssl-0.9.7d*, etc.).

Only one version of a software can be loaded at any given time. If you try to load two versions of a component, a conflict will occur (an error message will be printed and the version previously loaded will stay active).

Please, note that for each software a default version is provided. It is expected to be used in all normal cases.

4.4 Characteristics of libraries, tools and applications

To set the user's environment correctly, the version number is often not sufficient. This has been taken into account, especially for a library:

- ? the default numeric representation (AIX IBM: big endian),
- ? the default addressing mode (AIX IBM: OBJECT_MODE = 64 or 32),
- ? the size of the default Fortran integer, real and double precision variables (IBM: *-qintsize*, *-qrealsize*, etc.),
- ? the type of arguments transmitted to Fortran subprograms (real or double precision).

These characteristics are set by the *environment modulefiles*, which define some behaviour of the programming environment.

4.4.1 Multibyte numeric representation

Depending on which computing system you use, you will have to consider the byte order in which multibyte numbers are stored, particularly when you are writing them to a file. The two orders are called *little endian* and *big endian*. This is mostly intended for Fortran unformatted input/output operations. If a code reads input files written in a different numeric representation, input errors will occur. Some systems provide support (compiler options or environment variables) to enable the development and processing of files with *big endian* data organization on processors which use *little endian* (or vice versa).

On Power 4 and 5 IBM systems, for instance, there is only one sort of numeric representation: *big endian*.

Two modules have been defined to set the multibyte numeric representation:

- ? *bigendian* (default on AIX IBM, HPC Linux SGI and Super-UX NEC),
- ? *littleendian*.

On the AIX IBM platforms, the *littleendian* modulefile generates an error message, because it is not possible to change the default numeric representation. These two modules are available for portability reasons, as the same set of *environment modulefiles* should be available on all platforms (see the next section for explanations about the syntax of the *module* command).

4.4.2 Addressing mode

The default addressing mode is set to 64 bits on most platforms, but some libraries are provided in both modes on some machines. On AIX IBM, we can compile an application or a library in two addressing bit modes, 32 or 64 bits (this is not the case for the vector platform NEC-SX, for instance). Libraries can share these two modes (by mixing 32 and 64 bits object modules) but Fortran 90 libraries (IMSL, NAG, LAPACK 90, Netcdf, etc.) come often with Fortran 90 object modules which cannot be mixed, so you must say which addressing mode you would like to use. For this reason, two *modulefiles* are defined:

- ? *mode64* (default),
- ? *mode32*.

The first one is loaded by default in the *deisa modulefile*, but if you ever switch to 32-bit mode, some libraries or applications may be unavailable in this mode.

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode32 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe

> module load netcdf
WARNING: netcdf cannot be loaded due to missing prereq.
HINT: Load one of the following modules first: mode64
The library netcdf 3.5.0 does not exist in 32-bit mode!
```

Please, note also that these two modes are conflicting, so you must load only one and switch it to another if necessary:

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe

> module load mode32
WARNING: mode32 cannot be loaded due to a conflict.
HINT: You might try "module unload mode64" first.
```

4.4.3 Size of Fortran float numbers

A Fortran float type declaration must include either the REAL or DOUBLE PRECISION keyword. The term REAL, which stands to define *single precision* floating point numbers, refers to the IEEE 4-byte representation. The second one, which stands to define *double precision* floating point numbers, refers to the IEEE 8-byte representation. The KIND type parameter is generally equivalent to the byte size for 4- and 8-byte float values. For instance, AIX IBM Fortran sets the default kind parameter to 4.

Differences between data representations across systems and hardware platforms usually generate the most significant portability problems. Nearly all the computers today support the IEEE data format, but the default KIND type parameter may vary from one machine to another. Furthermore, on some computers, REAL variables are themselves represented as 64-bit floating point numbers, so porting codes developed on such platforms to IBM Power or SGI Altix systems would require the use of special compiler options (on IBM, `-qautodbl=dbl4` or `-qautodbl=dbl8`), in order to maintain the same accuracy.

There is also the case of old legacy codes developed for 32-bit architecture platforms many years ago, which used REAL as declaration for the floating numbers. Today, it is very often desirable to increase the internal storage of these numbers from 4 to 8-byte, to lower the rounding errors and to improve the accuracy of the computations.

Therefore, the version and the addressing mode are not sufficient attributes to distinguish Fortran libraries, which can be developed in both single and double precisions. On most systems, only one precision of the libraries is available (double precision on most systems). Nevertheless, on some systems both precisions are provided. In this case, the double precision routines use DOUBLE PRECISION for arguments and have different names compared to the single precision routines.

So, in order to help you to associate the corresponding precision Fortran libraries to the corresponding size of Fortran float type precision used in your codes, four *modulefiles* are available. They can be separated in two groups (because the size of REAL and DOUBLE PRECISION variables is not always linked), one for floats with single precision declarations (REAL or REAL(4)) and the other one for floats with double precision declaration (DOUBLE PRECISION or REAL(8)):

- ? *fortranreal4* and *fortranreal8* (with *fortranreal4* as default and the two modes being exclusive, that is to say that you must use the *switch* subcommand to change this characteristic),
- ? *fortrandouble8* and *fortrandouble16* (with *fortrandouble8* as default and the two modes being exclusive too).

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++   8) fortran      9) loadleveler 10) poe

> module switch fortranreal4 fortranreal8
unload real variables in single precision in Fortran on 4 bytes (default)
load real variables in single precision in Fortran on 8 bytes
(DEISA_FFLAGS)

> module switch fortrandouble8 fortrandouble16
unload real variables in double precision in Fortran on 8 bytes (default)
load real variables in double precision in Fortran on 16 bytes
(DEISA_FFLAGS)
```

Never try to set such a compiler option by hand, as using “*f90 -c -qautodbl=dbl4 pg.f90*” on IBM systems, because if you load a library later, the *Modules* framework will not be able to load the correct one, and will load the one using 4-byte single precision reals.

If later you will want to use a library which is not available in the selected mode, an error message will be displayed (please, note that only one error message is printed, the first encountered in the corresponding *modulefile*, even if there are several reasons for which the library does not exist in the requested modes):

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal8 5) fortrandouble16
6) c      7) c++   8) fortran      9) loadleveler 10) poe

> module load nag
The library NAG 20 does not exist for single precision reals on 8 bytes!
```

The size of COMPLEX and DOUBLE COMPLEX variables is linked to the size of REAL and DOUBLE PRECISION variables, which explains that no additional *modulefiles* for COMPLEX variables are needed.

4.4.4 Size of integers

For portability issues, Fortran compilers often provide an option to change the default kind type of the INTEGER variables (for instance “*-qintsize=8*” with the *IBM Fortran compiler*, “*-i8*” with the *Intel Fortran compiler* on Altix SGI systems, etc.).

The two following *modulefiles* set the default size of integers:

- ? *fortraninteger4* (default),
- ? *fortraninteger8*.

For instance, the AIX IBM Fortran compiler sets the default kind type parameter for integers to 4-byte. So, loading the *fortraninteger4 modulefile* will have no effect, while the *fortraninteger8 modulefile* will set the compiler option “*-qintsize=8*”.

Never try to set this compiler option by hand (“*f90 -c -qintsize=8 pg.f90*” with the IBM Fortran compiler), because if the compilation will be successful, the *Modules* framework will not be able to load the correct library if you will need one later, but still the library with 4-byte integers.

Please, note that these two modules are conflicting and that you cannot load them together, but you must switch from one to the other:

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran      9) loadleveler 10) poe

> module switch fortraninteger4 fortraninteger8
unload integer variables in Fortran on 4 bytes (default)
load integer variables in Fortran on 8 bytes (DEISA_FFLAGS)
```

In this environment, only libraries with default integer size on 8 bytes will be available and if they do not exist, an error message will be displayed:

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger8 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran      9) loadleveler 10) poe

> module load nag
The library NAG 20 does not exists for integers on 8 bytes!
```

4.5 Modules usage

4.5.1 Initialization

The access to the *Modules* software itself is automatically initialized for you when you log in or when you submit a batch job, according to the shell that you use. Nevertheless, you must initialize the DEISA environment yourself at the beginning of your interactive sessions and your submitted jobs, using the *deisa modulefile*.

For this action, the command is:

```
> module load deisa
load big endian multibytes numeric representation (default)
load 64 bits mode (OBJECT_MODE, DEISA_FFLAGS) (default)
load integer variables in Fortran on 4 bytes (default)
load real variables in single precision in Fortran on 4 bytes (default)
load real variables in double precision in Fortran on 8 bytes (default)
IBM C compiler version 6.0.0.7 load (PATH, LIBPATH, MANPATH)
IBM C++ compiler version 6.0.0.8 load (PATH, LIBPATH, MANPATH)
IBM Fortran compiler version 8.1.1.4 load (PATH, LIBPATH, NLSPATH, MANPATH)
IBM LoadLeveler batch manager version 3.2.0.8 load
IBM Parallel Environment version 4.1.1.3 load
```

which defines the special DEISA variables to reference the file systems in a portable way (\$DEISA_HOME, etc. – see section 3.1) and loads all the default *modulefiles*.

For each subgroup of systems, these default *modulefiles* are:

```
AIX IBM systems:      deisa, mode64, fortraninteger4, fortranreal4, fortrandouble8,
                      bigendian, c, c++, fortran, loadleveler, poe
HPC Linux SGI systems: deisa, mode64, fortraninteger4, fortranreal4, fortrandouble8,
                      bigendian, c, c++, fortran, lsf
Super-UX NEC systems: deisa, mode64, fortraninteger4, fortranreal4, fortrandouble8,
                      bigendian, c, c++, fortran, nqs
```

4.5.2 Help

The *help* subcommand gives you the syntax of the main subcommands of *module*:

```
> module help

Available Commands and Usage:

+ list
+ avail
+ help      modulefile [modulefile ...]
+ whatis   modulefile
+ display  modulefile
+ load     modulefile [modulefile ...]
+ unload   modulefile [modulefile ...]
+ switch   modulefile1 modulefile2
. . .
```

4.5.3 List of the modulefiles currently loaded

The *list* subcommand lists all the *modulefiles* which are currently loaded in your environment. For instance, after the initialization step done by the *deisa modulefile*, you must have on AIX IBM systems:

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c 7) c++ 8) fortran 9) loadleveler 10) poe
```

4.5.4 List of the available modulefiles

The *avail* subcommand gives the complete list of all the *modulefiles* available on the system on which it is executed, that is to say all the components of the Common Production Environment, including all versions. As previously said, the *modulefiles* without any version number are the default ones. Due to these various versions for each component, this list is rather long, as you may see in the example below (of course, the list that you will get depends on the computer on which you execute this command):

```
> module avail

----- /usr/local/pub/Modules/modulefiles/compilers -----
c                c++-6.0.0.8          fortran-8.1.1.4   java-1.3.1.13
c-6.0.0.7        c++-7.1.0.0          fortran-8.1.1.5
c-7.1.0.0        fortran                fortran-9.1.0.0
c++              fortran-8.1.1.2     java

----- /usr/local/pub/Modules/modulefiles/environment -----
bigendian        deisa                  littleendian      fortranreal4
fortrandouble8   fortraninteger4       mode32            fortranreal8
fortrandouble16  fortraninteger8       mode64

----- /usr/local/pub/Modules/modulefiles/libraries -----
essl             lapack                 nag-19            scalapack
essl-4.1.0.0     lapack-3.0            nag-20            scalapack-1.7
fftw             mass                   netcdf            wsmmp
fftw-2.1.5       mass-3.2.1            netcdf-3.5.0     wsmmp-4.08.05
fftw-3.0.1       mass-4.1              pacxmpi
hdf5             mpichg2               pssl
hdf5-1.6.3       nag                   pssl-3.1.0.0

----- /usr/local/pub/Modules/modulefiles/tools -----
emacs            nedit-5.3             poe               totalview
emacs-21.2       omniorb               poe-4.1.1.3      totalview-6.5.0.2
globus           omniorb-4.0.3        python            unicore
globus-2.2.4     openssh               python-2.2.3     unicore-4.1
hpm              openssh-3.9p1        python-2.3.3     xgrafx
hpm-2.5.2        openssl              tcl               xgrafx-2.6.0
loadleveler      openssl-0.9.7d       tcl-8.4.7
loadleveler-3.2.0.8 perl                tk
nedit            perl-5.8.0           tk-8.4.7

----- /usr/local/pub/Modules/modulefiles/applications -----
cpmd             cpmd-3.9.1           orb-1.2.2         wien2k
cpmd-3.7.2       orb
```

4.5.5 Help on a specific modulefile

The *help* subcommand, followed by the name of a *modulefile*, gives more information about its purpose and effect, as shown in this example for the Fortran IBM compiler on AIX systems:

```

> module help fortran
-----
Module Specific Help for
/usr/local/pub/Modules/modulefiles/compilers/fortran:

Fortran compiler version 8.1.1.4 (PATH, LIBPATH, NLSPATH, MANPATH)

**** DEISA Common Production Environment ****

modulefile "fortran" - Version 1.1

Set environment variables to use the IBM Fortran compiler version 8.1.1.4.
-----

```

4.5.6 Location of a modulefile and its brief description

The *whatis* subcommand gives the location of a *modulefile* and its brief description:

```

> module whatis fftw
----- /usr/local/pub/Modules/modulefiles/compilers -----
----- /usr/local/pub/Modules/modulefiles/environment -----
----- /usr/local/pub/Modules/modulefiles/libraries -----
      fftw: Set environment variables to enable the usage of the FFTW 3.0.1
numerical library.
----- /usr/local/pub/Modules/modulefiles/tools -----
----- /usr/local/pub/Modules/modulefiles/applications -----

```

4.5.7 Detailed description of a modulefile

The *display* subcommand describes what would be the effect of the specified *modulefile* in your environment, showing which changes it would make in the content of the environment variables *PATH*, *MANPATH*, etc. and which new variables it would define. Below, an example for the *fortran* and *fortran-9.1.0.0 modulefiles* on AIX IBM systems:

```

> module display fortran
-----
/usr/local/pub/Modules/modulefiles/compilers/fortran:

module-whatis   Set environment variables to use the IBM Fortran compiler
version 8.1.1.4.
prepend-path    PATH      /usr/local/pub/Modules/wrappers/fortran
Fortran compiler version 8.1.1.4 display (PATH, LIBPATH, NLSPATH, MANPATH)
-----

> module display fortran-9.1.1.0
-----
/usr/local/pub/Modules/modulefiles/compilers/fortran-9.1.1.0:

```

```

module-whatis  Set environment variables to use the IBM Fortran compiler
version 9.1.1.0.
prepend-path   PATH
/usr/local/pub/Modules/wrappers/fortran:/usr/local/prod/xlf9110/usr/bin
prepend-path   LIBPATH /usr/local/prod/xlf9100/usr/lib
prepend-path   NLSPATH
/usr/local/prod/xlf9110/usr/lib/nls/msg/%L/%N:/usr/local/prod/xlf9110/usr/
lpp/xlf/bin/default_msg/%N
prepend-path   MANPATH /usr/local/prod/xlf9100/usr/share/man
conflict       fortran
Fortran compiler version 9.1.1.0 display (PATH, LIBPATH, NLSPATH, MANPATH)
-----

```

4.5.8 Loading a modulefile

The *load* subcommand adds a *modulefile* in your environment, executing all the commands defined inside. As previously said, appending a version number to the name of the component will load the *modulefile* of this specific version, which can be different from the default one.

```

> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe

> module load lapack

LAPACK version 3.0 "load" (DEISA_FLIBS)
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe
11) lapack

> echo $DEISA_FLIBS
-L/usr/local/pub -llapack -lessl

>module load cpmd-3.9.1

CPMD version 3.9.1 "load" (PATH, LIBPATH, MANPATH)

> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe
11) lapack 12) cpmd-3.9.1

> type cpmd
cpmd is /usr/local/prod/cpmd/cpmd-3.9.1/bin/cpmd

```

4.5.9 Unloading a modulefile

The *unload* subcommand allows you to remove all the definitions set up by a previous *load* command. The *modulefiles* can be removed in any order.

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe
11) lapack 12) cpmd-3.9.1

> module unload lapack

LAPACK version 3.0 "unload" (DEISA_FLIBS)

> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe
11) cpmd-3.9.1

> echo $DEISA_FLIBS
```

4.5.10 Switching between two versions of a modulefile

It is easy to switch between two different versions of a software. This subcommand shows the power of the *Modules* concept, as different versions of huge and complex packages, like compilers or complete applications, can be replaced using a single simple command, and this can be done independently of the shell that is used.

```
> module switch fortran fortran-9.1.0.0

Fortran compiler version 8.1.1.4 "unload" (PATH, LIBPATH, NLSPATH, MANPATH)
Fortran compiler version 9.1.0.0 "load" (PATH, LIBPATH, NLSPATH, MANPATH)

> echo $PATH
/usr/local/pub/Modules/wrappers/fortran:/usr/local/prod/xlf9110/usr/bin:/usr/local/pub/bin:/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:...

> echo $LIBPATH
/usr/local/prod/xlf9110/usr/lib:/usr/lib
```

```
> module load cpmd

CPMD version 3.7.2 "load" (PATH, LIBPATH, MANPATH)

> type cpmd
cpmd is /usr/local/prod/cpmd/cpmd-3.7.2/bin/cpmd

> module switch cpmd cpmd-3.9.1

CPMD version 3.7.2 "unload" (PATH, LIBPATH, MANPATH)
```

```
CPMD version 3.9.1 "load" (PATH, LIBPATH, MANPATH)
> type cpmd
cpmd is /usr/local/prod/cpmd/cpmd-3.9.1/bin/cpmd
```

4.6 Compiling and linking

4.6.1 Compiler wrappers

In the DEISA environment, the C, C++ and Fortran compilers are not directly invoked; compiler wrappers are called instead.

For instance, in the AIX IBM environment, the wrapper script for the Fortran compiler (xlf, xlf_r, f90, etc.) uses two environment variables called \$DEISA_FFLAGS (for the compilation steps) and \$DEISA_FLIBS (for the link steps). With the C and C++ compilers, these variables are called \$DEISA_CFLAGS and \$DEISA_CLIBS. You are advised not to change these variables, as it can generate unwanted side effects.

4.6.2 Compiling and linking with libraries

With this approach, compiling and linking your codes is very simple: you will just need to load the *modulefiles* of the desired libraries and call the compiler. For instance, if you want to compile a code with the NAG Fortran library:

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe

> module load nag

NAG Library mark 20 "load" in 64 bits mode (DEISA_FFLAGS, DEISA_FLIBS)

> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe
11) nag

> f90 -o test-nag test-nag.f90
Fortran command: /usr/bin/f90 -q64 -I/usr/local/prod/nag/nag20/include/q64
test-nag.f90
-L/usr/local/prod/nag/nag20/lib/q64 -lnag_use_essl -lessl
/usr/bin/f90:
** test_nag   === End of Compilation 1 ===
1501-510  Compilation successful for file test-nag.f90.

> ./test-nag
Test of Fortran 90 NAG interfaces correct
```

As explained in the previous paragraph, the Fortran compiler wrapper (which replaces the *real* compiler with the same name, just allowing to invoke it with special options) uses the two environment variables `$DEISA_FFLAGS` and `$DEISA_FLIBS` for the compilation and linking. A similar approach is used for the C/C++ compiler wrapper, except that the two environment variables are called `$DEISA_CFLAGS` and `$DEISA_CLIBS`. These environment variables are updated each time a *modulefile* is loaded/unloaded:

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++   8) fortran          9) loadleveler 10) poe

> module load nag
NAG Library mark 20 "load" in 64 bits mode (DEISA_FFLAGS, DEISA_FLIBS)

> echo $DEISA_FFLAGS
:-q64:-I/usr/local/prod/nag/nag20/include/q64

>echo $DEISA_FLIBS
:-L/usr/local/prod/nag/nag20/lib/q64 -lnag_use_essl -lessl
```

As previously said, never modify these environment variables by yourself circumventing the *module* command. Please, note that in these environment variables the character ":" appears each time a new module is loaded (`-q64` corresponds to the module *mode64* and `-I/usr/local/prod/nag/nag20/include/q64` to the module *nag*). These special characters are absolutely necessary to the management of *modulefiles*. Therefore, you cannot use these environment variables directly in your compilation lines or in your Makefiles!

```
> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++   8) fortran          9) loadleveler 10) poe
11) nag

> /usr/bin/f90 test-nag.f90 $DEISA_FFLAGS $DEISA_FLIBS
/usr/bin/f90: 1501-228 input file :-q64:-I/usr/local/prod/nag/nag20/
include/q64 not found
/usr/bin/f90: 1501-228 input file :-L/usr/local/prod/nag/nag20/lib/q64 not
found
/usr/bin/f90:
"test-nag.f90", line 10.7: 1514-219 (S) Unable to access module symbol
file for module nag_f77_d_chapter. Check path and file permissions of
file. Use association not done for this module.
1501-511 Compilation failed for file test-nag.f90.
```

4.6.3 Complete example

```
> module load essl fftw

> module list
```

```

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe
11) essl 12) fftw

> f90 -L $DEISA_HOME/model/libraries -llib1 -llib2 -o model model.f90

> ./model
Segmentation fault.
core dump.

> module load totalview

> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe
11) essl 12) fftw 13) totalview

> totalview ./model &

```

4.6.4 Error messages, known problems, and side effects

If you try to load two versions of the same software, an error message is printed and the corresponding *modulefile* is not loaded. The only possibility is to change one version by another one, using the *switch* version.

```

> module list

Currently Loaded Modulefiles:
1) deisa 2) mode64 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe

> module load fortran-8.1.1.2
WARNING: fortran-8.1.1.2 cannot be loaded due to a conflict.
HINT: You might try "module unload fortran" first.

```

Some modules require other modules to be loaded. If they are not present, an error message is printed:

```

> module switch mode64 mode32

"unload" 64 bits mode (OBJECT_MODE, DEISA_FFLAGS) (default)
"load" 32 bits mode (OBJECT_MODE, DEISA_FFLAGS)

> module list

Currently Loaded Modulefiles:
1) deisa 2) mode32 3) fortraninteger4 4) fortranreal4 5) fortrandouble8
6) c      7) c++    8) fortran          9) loadleveler 10) poe

> module load fftw
The library FFTW 2.1.5 does not exist in 32 bits mode!

```

Also, please, be aware that strong dependencies may be enforced on some platforms. For instance, on the NEC-SX, if the *fortrandouble16 modulefile* is loaded then *fortranreal8* is automatically loaded. In such cases, a warning message is printed.

5. Job submission

Job can be submitted in three different ways:

- ? by direct submission to the batch subsystem,
- ? through the UNICORE software, and
- ? through Web portals (not yet available).

5.1 Job submission to a batch subsystem

You will interact with the resource manager or the job scheduler that is available on your home site. This is the site where you log in and submit your jobs. Table 3 shows the job schedulers which are available at each site:

Table 3: Job schedulers available at each site

Site	Available job scheduler
CINECA	LoadLeveler
CSC	LoadLeveler
ECMWF	LoadLeveler
EPCC	LoadLeveler
FZJ	LoadLeveler
IDRIS	LoadLeveler
RZG	LoadLeveler
SARA	LSF

Each job scheduler has specific commands to submit jobs and to monitor the job execution. Table 4 shows the main commands which are available for two different job schedulers. These commands can be used to submit a new job, to cancel a submitted job, and to query the status of a submitted job.

Table 4: Main job scheduler commands

Job scheduler	Submit command	Delete command	Job status command
LoadLeveler	llsubmit	llcancel	llq
LSF	bsub	bkill	bjobs

For detailed information about the usage of these and other job scheduler commands, please see the documentation at your home site.

5.1.1 Job submission using LoadLeveler on the DEISA IBM sites

It is crucial to understand how the LoadLeveler job schedulers interact with the distributed DEISA infrastructure. From DEISA sites using AIX IBM systems (which all have the LoadLeveler job scheduler, share the \$DEISA_HOME and \$DEISA_DATA filesystems, and have a local \$DEISA_SCRATCH filesystem), your LoadLeveler jobs can run across all these sites (it is called a Multi-Cluster job).

Table 5: Job submission example with LoadLeveler

```
# @ job_name = MyJob
# @ input    = /dev/null
# @ output   = MyModel/output.%(jobid).log_ll
# @ error    = MyModel/output.%(jobid).log_ll
# @ notify_user = s.campagna@cineca.it
# @ notification = error
# @ shell    = /bin/bash
# @ requirements = (Feature == "DEISA")
# @ job_type = parallel
# @ total_tasks = 128
# @ cpu_limit = 06:00:00,05:50:00
# @ data_limit = 512mb
# @ queue

module load deisa

exe=$DEISA_HOME/MyModel/exe/MyModel.x

cd $DEISA_SCRATCH/MyModel/work

$exe
```

Table 5 shows an example of a LoadLeveler job submission file. This file has been produced at CINECA by user cne0cin3. All the lines starting with “# @” are parsed by the job scheduler submission command, which is *lsubmit* in this case; they state how the job must be run, and how many resources it does need. All other lines will be ignored by *lsubmit*.

Let’s have a closer look at the example in Table 5:

- ? this job is named “*MyJob*”,
- ? it takes no standard input,
- ? it puts its standard output and standard error into the same file *MyModel/output.%(jobid).log_ll* in the *\$DEISA_HOME* directory,
- ? it notifies events to the s.campagna@cineca.it mail address,
- ? it notifies only error events,
- ? it will be run in a */bin/bash* shell,
- ? it requires the “*DEISA*” feature, which implies that this job can be re-routed to any DEISA host of the AIX IBM kind,
- ? it’s an MPI job;
- ? it requires 128 processors;
- ? it will use no more than 6 hours of CPU time (the process will receive a notification after 5 hours and 50 minutes of CPU time);
- ? it will use no more than 512MB of memory.

The “# @ *queue*” line states that the job definition is complete, and that the job can be queued.

When submitted (using the LoadLeveler command *lsubmit*), this job will run on the local host, if resources are available, but it can also be run on a different host, depending on the choices of the DEISA inter-site job balancing mechanism. This mechanism allows a job to be re-routed to a different host.

During execution on a DEISA host, the same script shown in Table 5 is executed, using a */bin/bash* shell. During this phase, all the “# @” lines are obviously comments for the shell interpreter, so they are ignored.

5.1.2 Batch jobs for compilation and pre and post-processing

The compilation can be done interactively, on the home site, as well as pre and post-processing steps. If they require very long CPU times, and the interactive limits do not allow that, it can also be done in batch mode. In this case, a batch job can be prepared following the instruction in paragraph 5.1.1. See Table 6 for a simple example of a serial job.

Table 6: Example of batch compilation

```
# @ job_name = MyJob
# @ input    = /dev/null
# @ output   = MyModel/comp_out.$(jobid).log_ll
# @ error    = MyModel/comp_out.$(jobid).log_ll
# @ notify_user = s.campagna@cineca.it
# @ notification = error
# @ shell    = /bin/bash
# @ requirements = (Feature == "DEISA")
# @ job_type = serial
# @ cpu_limit = 02:00:00,01:50:00
# @ data_limit = 512mb
# @ queue

module load deisa

cd $DEISA_HOME/MyModel/src

make
```

5.2 Job submission through UNICORE

UNICORE [<http://unicore.sourceforge.net/>] provides a seamless interface for preparing and submitting jobs to a wide variety of computing resources. It has a three-tier design:

- ? A UNICORE client GUI is used for the preparation, submission, monitoring, and administration of jobs.
- ? Site specific information on computing resources, including the availability of applications, is provided by a *Network Job Scheduler* (NJS). This server dispatches the jobs to a dedicated target machine or cluster, and handles dependencies and data transfers for complex workflows. It transfers the results of executed jobs from the target machine.

- ? A *Target System Interface (TSI)* which is running on the target machine is the interface to the batch scheduler (up to now in DEISA to the multi-cluster LoadLeveler at CINECA, FZJ, IDRIS and RZG) on the target machine.

From a UNICORE client, you connect to a UNICORE gateway of your home site. If this gateway is not available, and only in this case, you can also connect to the UNICORE gateway of any other DEISA site. In general, every NJS server is accessible from every UNICORE gateway. But particularly in the homogeneous IBM case (every site has the same computer platform providing multi-cluster LoadLeveler), you only need to access to the NJS of the home site; then the mechanism of job rerouting, described in the paragraph 5.1, will allow you to run on every other IBM cluster. Figure 1 depicts the UNICORE infrastructure with regard to the fact that each one of the four sites offers a cluster of IBM Power 4 machines with multi-cluster LoadLeveler which is able to reroute jobs between different sites.

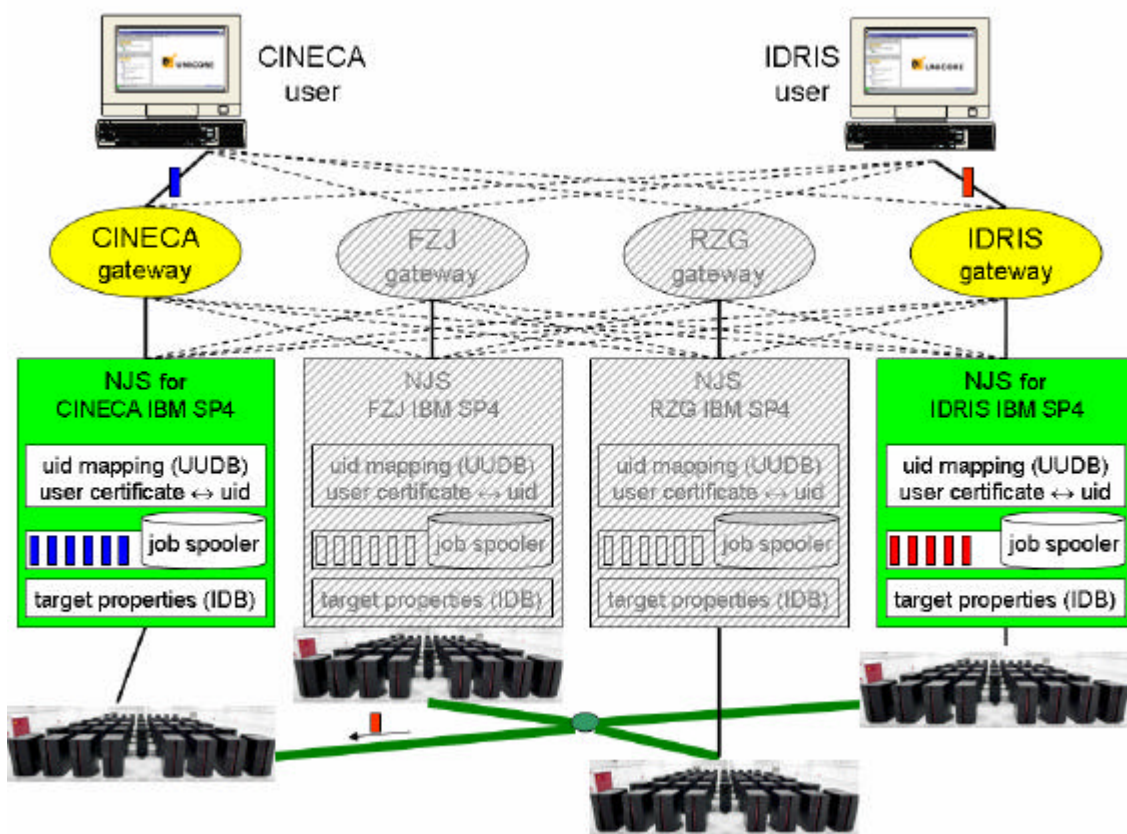


Figure 1: DEISA UNICORE infrastructure, designed in particular for the heterogeneous case. In the case of a homogeneous infrastructure jobs are typically submitted via the NJS at the home site.

The UNICORE client allows you to create a new job or modify existing ones. They can be composed of various tasks within a complex workflow including data transfers and dependencies. You can:

- ? define the workflow, in a system independent way (UNICORE creates an internal abstract representation that allows to execute the job on any platform),
- ? choose the appropriate platform for the execution of each task,

- ? assign resources to the task within the limits that are available on the selected target system,
- ? submit the job, monitor its status, and control it.

5.2.1 UNICORE client configuration and certificate management

When you run the UNICORE client for the first time, a new *.unicore* directory is created in your home directory. Additionally, an empty keystore file is created in this new directory; which will contain all the PKI keys which you will need to use UNICORE. You will be asked for a passphrase to encrypt the keystore; keep attention to remember this passphrase.

In order to use UNICORE, you first need to obtain a valid certificate, following the instructions in paragraph 2.3, and then to import it into the UNICORE keystore, by clicking on:

Settings -> Keystore Editor -> Actions -> Import Keystore

It is mandatory to perform every action through UNICORE.

You will also need to import other certificates. In order to contact a gateway, you need in fact to import the gateway signer's certificate. Since you have to use the gateway of your site, the gateway signer is the Certification Authority (CA) responsible for your site. The certificates of all CAs used by DEISA for signing can be downloaded from <http://winnetou.matrix.sara.nl/deisa/certs/CACerts>. In order to install this CA certificate, you have to put it into a file, and then import it by clicking on the menu:

Settings -> Keystore Editor -> Actions -> Import certificate

in the UNICORE client, and by choosing the certificate file. Additionally, when you install new plugins in UNICORE, you will also have to install the certificate of the plugin signer. The procedure is the same as outlined above for the case of the gateway signer's certificate.

Whenever new sites or new signers are added an update of the trusted certificates becomes necessary. Re-import the global certificate file.

You can configure your client by changing your settings. Click on:

Settings -> User defaults

and the window showed in Figure 2 will let you specify, for example, the URL of the gateway to use. This URL points to an xml file. For example, if CINECA is your home site, you will have to use the URL provided by CINECA, which is http://grid.cineca.it/docs/DEISA_GW.xml. The full list of all DEISA gateways, as displayed in Table 7, is also available at the DEISA Web site.

Table 7: URLs for DEISA UNICORE Site Servers (Usites)

Site	URL of the gateway
CINECA	http://grid.cineca.it/docs/DEISA_GW.xml
FZJ	http://www.fz-juelich.de/unicore/unicoreSites_Deisa.xml
IDRIS	http://unicore.idris.fr/grid/unicore/gateways.xml
RZG	http://www.rzg.mpg.de/unicore/gateways.xml

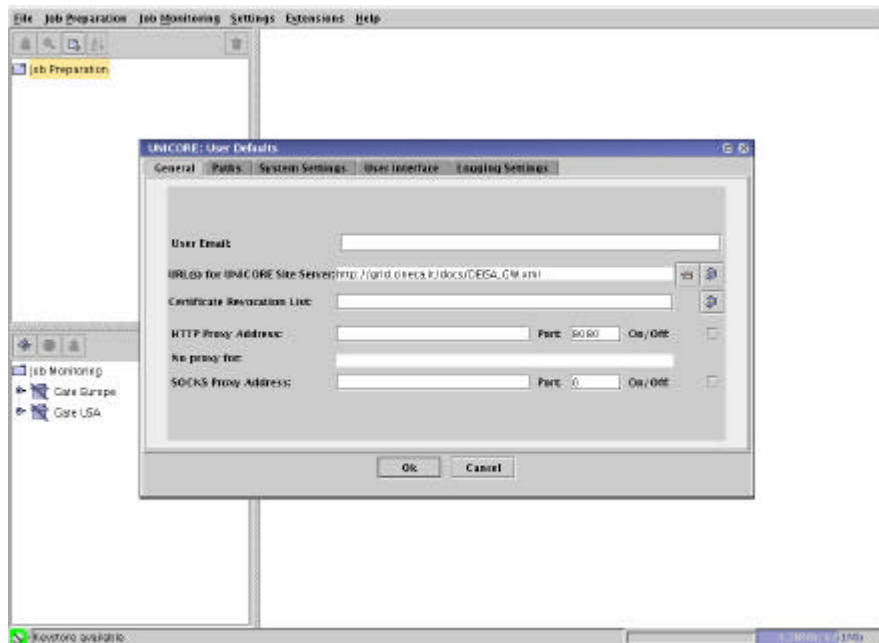


Figure 2: Setting of the DEISA gateway

You can also specify the path of the directory in which new plugins should be installed, see Figure 3.

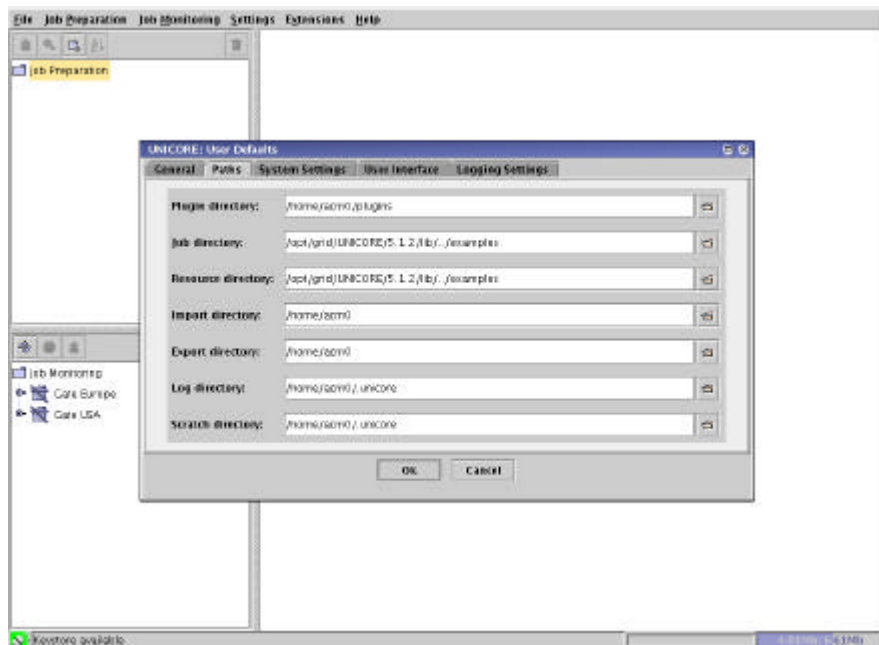


Figure 3: Configuration on the UNICORE client. This window allows you to change default paths used by UNICORE.

5.2.2 Example of a UNICORE job

As an example of a job definition through UNICORE, we will consider a job consisting of two main tasks. The first task has to run the *MyModel.x* executable (using some input files), while the second must run the *MyPostp.x* executable on the output of the first one.

In our example, all the work is done on your personal computer or workstation, on which the UNICORE client was installed and configured. The input files used in the run reside locally on this computer, so they will have to be transferred to the target host at the beginning; this is also done through UNICORE.

After having started the UNICORE client, you can create a new job named "MyModel_job", as shown in Figure 4. The new job must be assigned to a specific host to be run. Suppose that your home site is CINECA; you will first have to choose the "DEISA – Usite" Usite, and then the "SP4" Vsite inside this Usite (SP4 is the name of the CINECA IBM cluster).

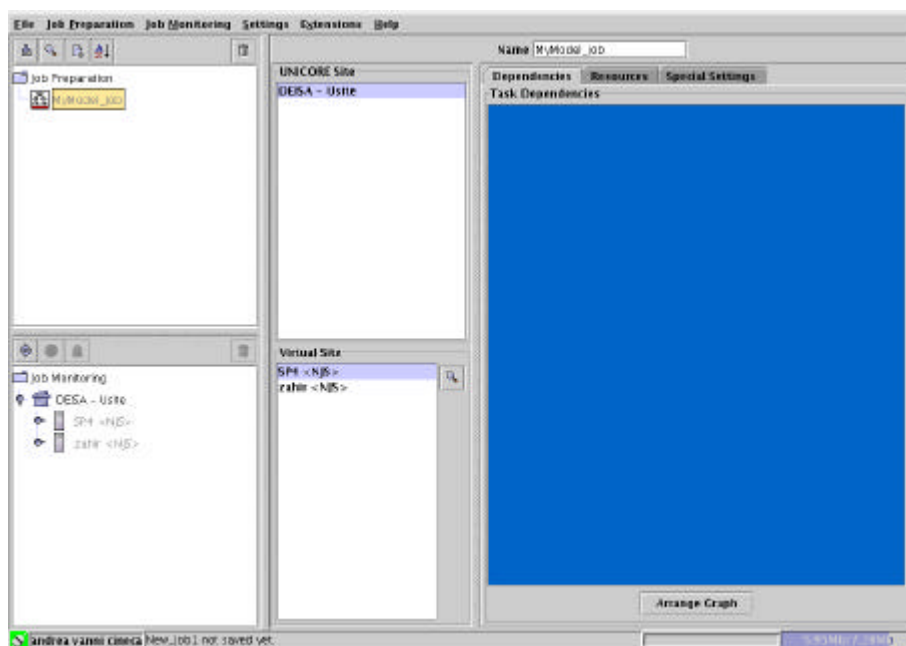


Figure 4: Definition of a new job, named "MyModel_job", through the UNICORE client

Having chosen the "SP4" site, all the tasks inside the "MyModel_job" job will be run on that platform, inside the same USPACE, which is a new, empty scratch directory.

Now, a new task must be added to run the *MyModel.x* program. You can use a *script* task, i.e. a task which consists of a script to be run on the target site. This is shown in Figure 5. Other tasks are available. For example, the *command task* allows you to run a single command on the remote platform; the command can be chosen by browsing the remote file systems. The *script task* that we will use in this case allows you to write and execute a shell script (in general consisting of more than one command) on the remote platform. This is a convenient way to run existing jobs under UNICORE. You can also install other plugins, which offer you the possibility to run tasks using an application specific GUI.

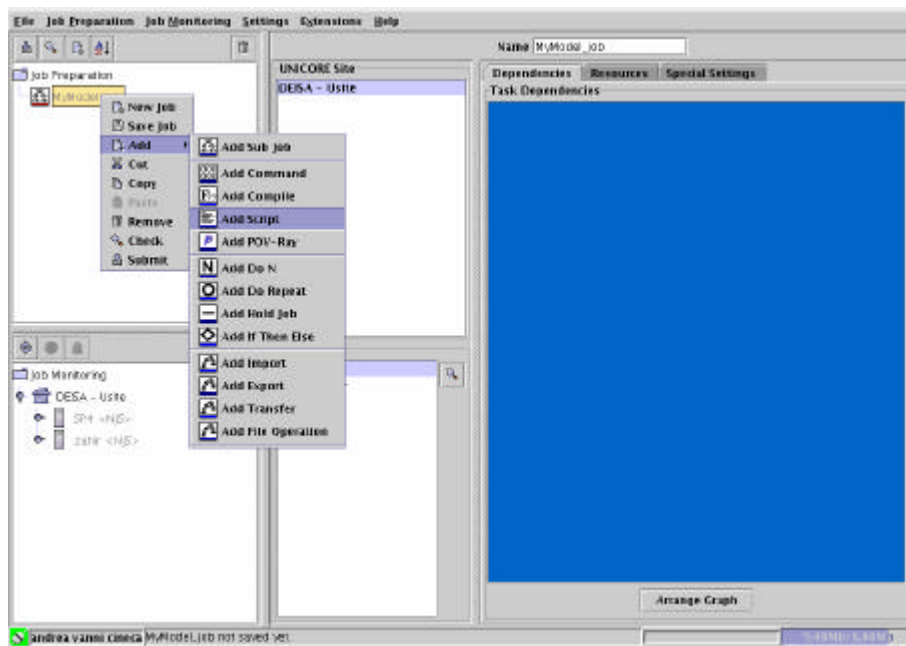


Figure 5: Creation of a new task named “MyModel_task” inside “MyModel_job”

The script content can now be defined as shown in Figure 6.

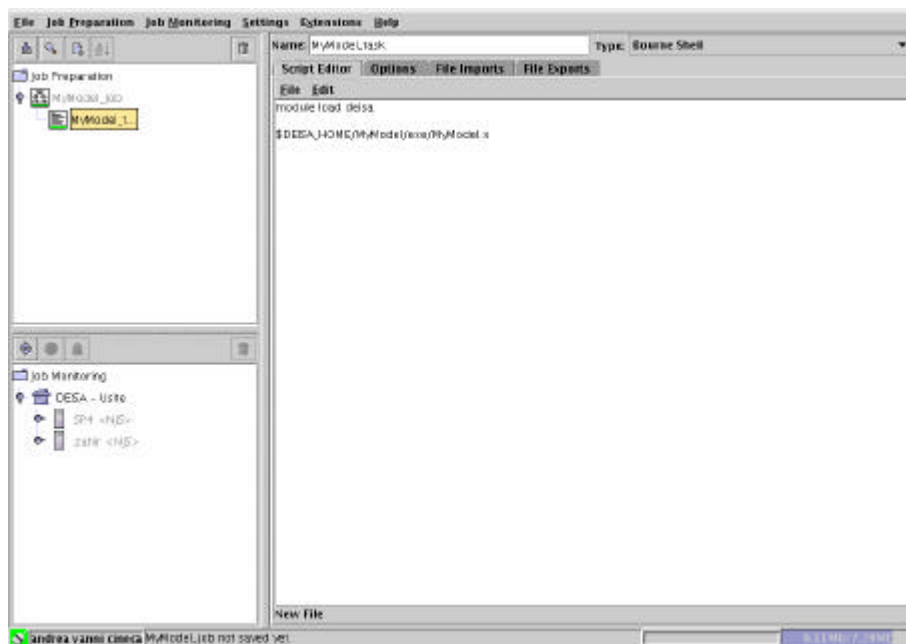


Figure 6: Content of the “MyModel_task” script

Notice the use of the “module load deisa” command at the beginning of the script. This is mandatory to set up a correct environment. Since some input files are needed by this task, they have to be imported in the job’s USPACE; this can be done using the “File imports” panel of the task, as shown in Figure 7: the local files *run128.conf* and *run128.input* are copied in the USPACE, respectively as *MyModel.conf* and *MyModel.input*.

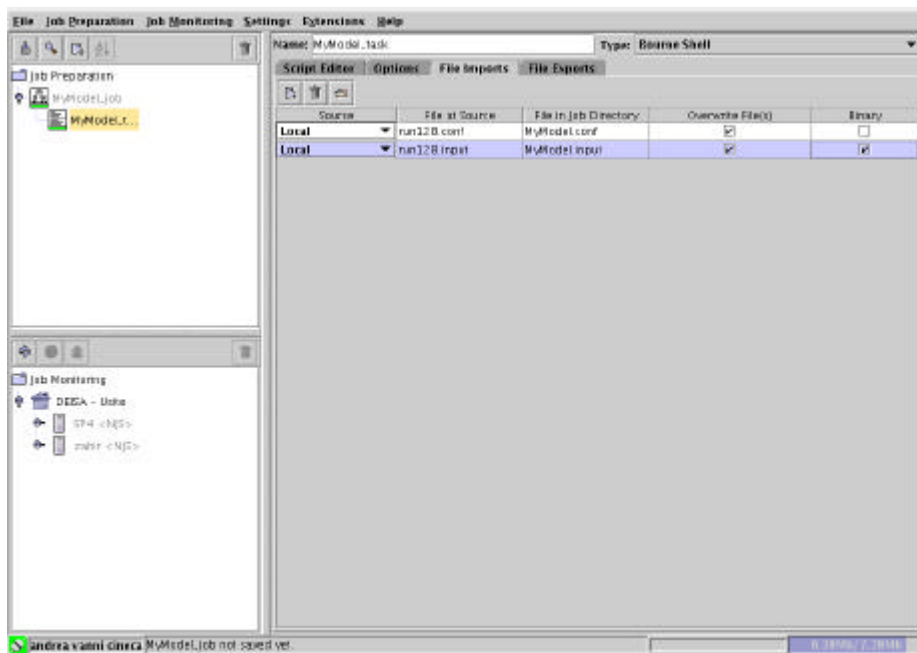


Figure 7: File imports for "MyModel_task"

Now, we must assign to the task a set of resources available on SP4. In the *resources* panel of "MyModel_job" we add a new resource set, named "MyModel_task", requiring 128 processors. This resource set must then be assigned to "MyModel_task" (see Figure 8 and Figure 9).

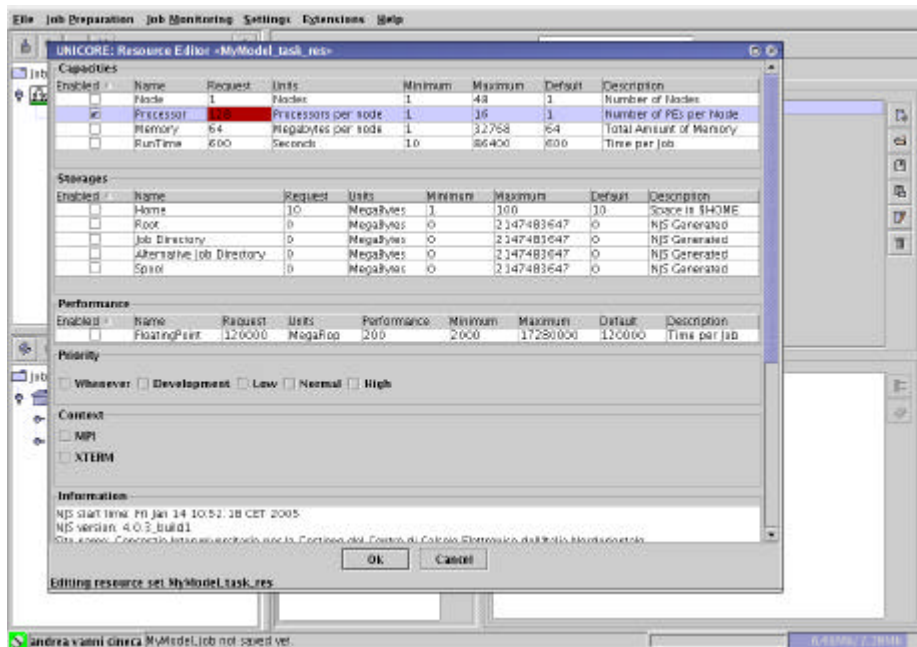


Figure 8: Resource set for "MyModel_task"

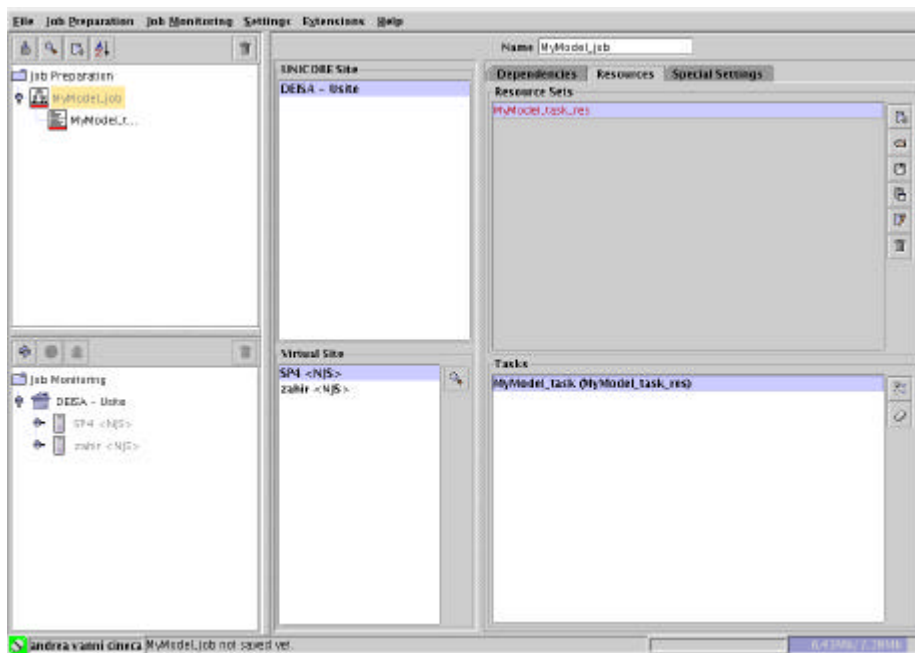


Figure 9: Assignment of the "MyModel_task_res" resource set to "MyModel_task"

The definition of this task is now complete. Now we will have to add a new task for the postprocessing phase; in our example this phase is also run on the same Vsite (the SP4 platform inside the "CINECA – Usite"), However, this is done in another job because the postprocessing phase is serial, so it should not hold the huge number of processors reserved by the previous phase. Thus, we have to add a new subjob referring to that platform. This new job is named "MyPostp_subjob", see Figure 10.

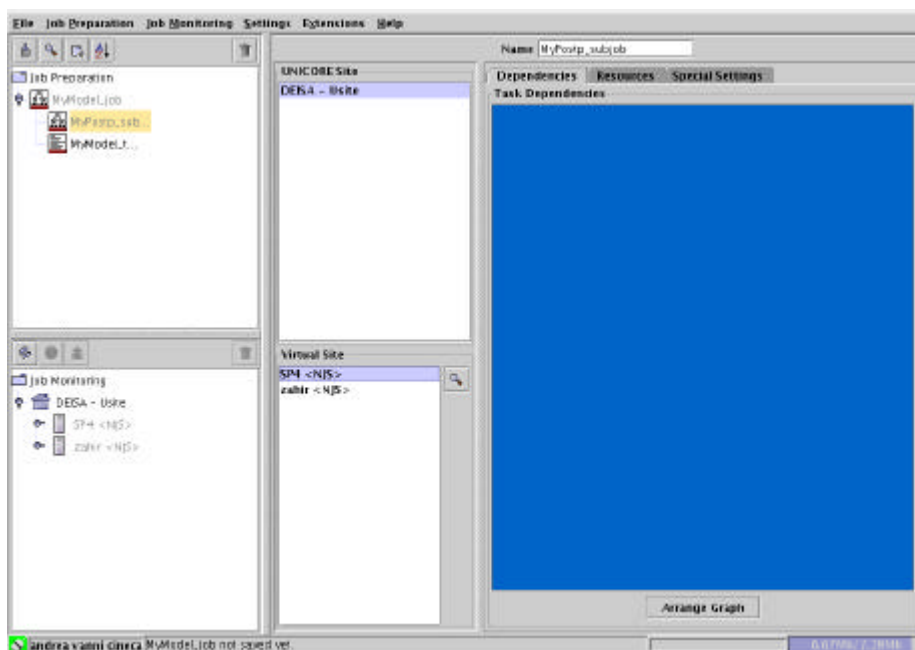


Figure 10: Definition of a "MyPostp_subjob" subjob

The new task, named “MyPostp_task”, is also a script task. Its content is shown in Figure 11.

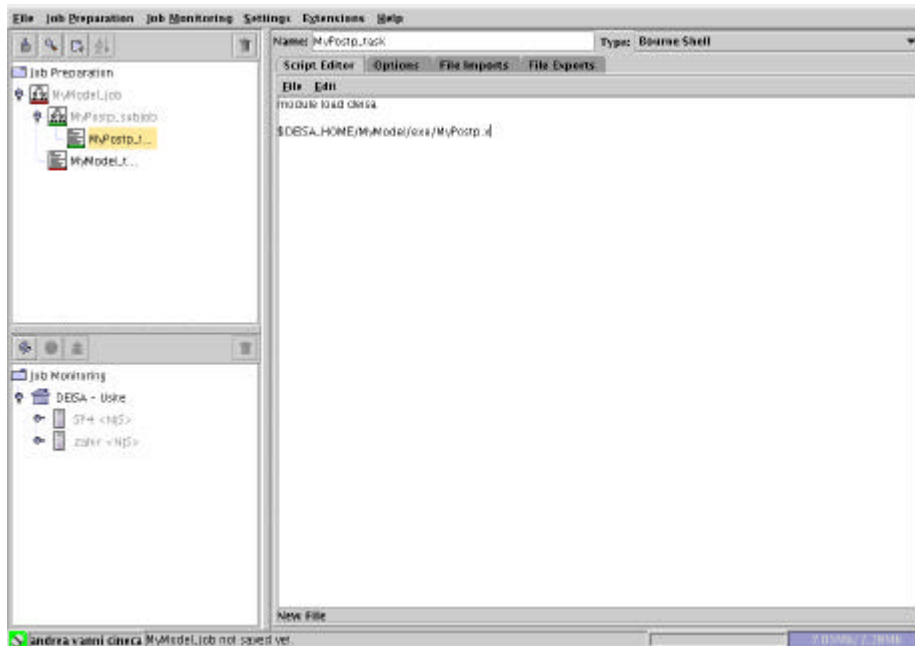


Figure 11: Definition of a new script task named "MyPostp_task" inside "MyPostp_subjob"

Again, we will have to define a set of resources on SP4 for this task. This is shown in Figure 12. We will need only one processor for this phase.

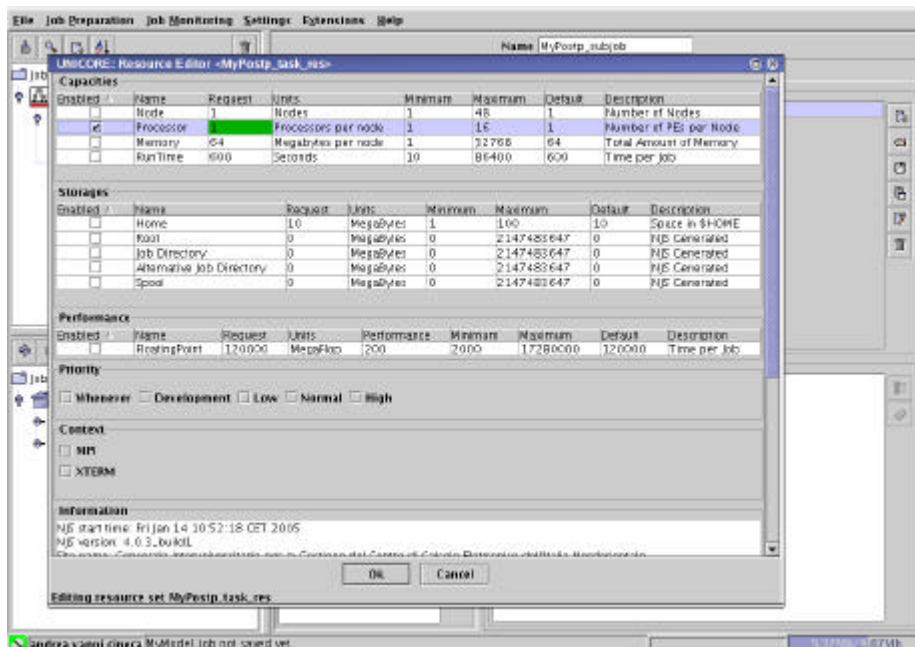


Figure 12: Definition of a resource set named "MyPostp_task_res" for "MyPostp_task"

This resource set has then to be assigned to the “MyPostp_task”, as previously done for “MyModel_task”. The output of “MyPostp_task” must be transferred to your personal

computer; this can be done using the “File exports” panel of the task, as shown in Figure 13.

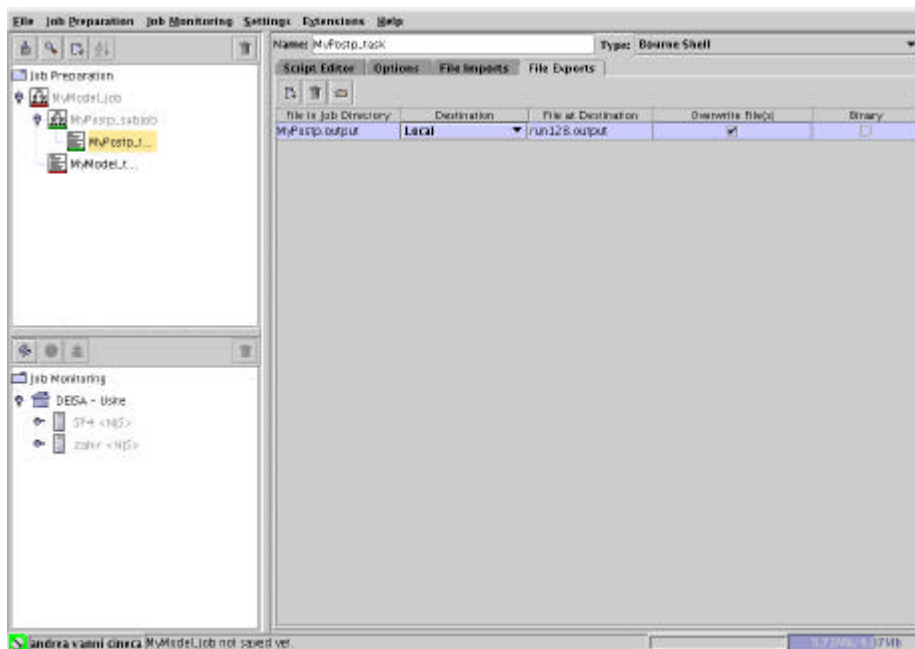


Figure 13: File exports for "MyPostp_task"

Now, you will have to link the output of “MyModel_task” with the input of “MyPostp_task”; this can be accomplished by defining a “Transfer” task inside “MyModel_job” (see Figure 14).

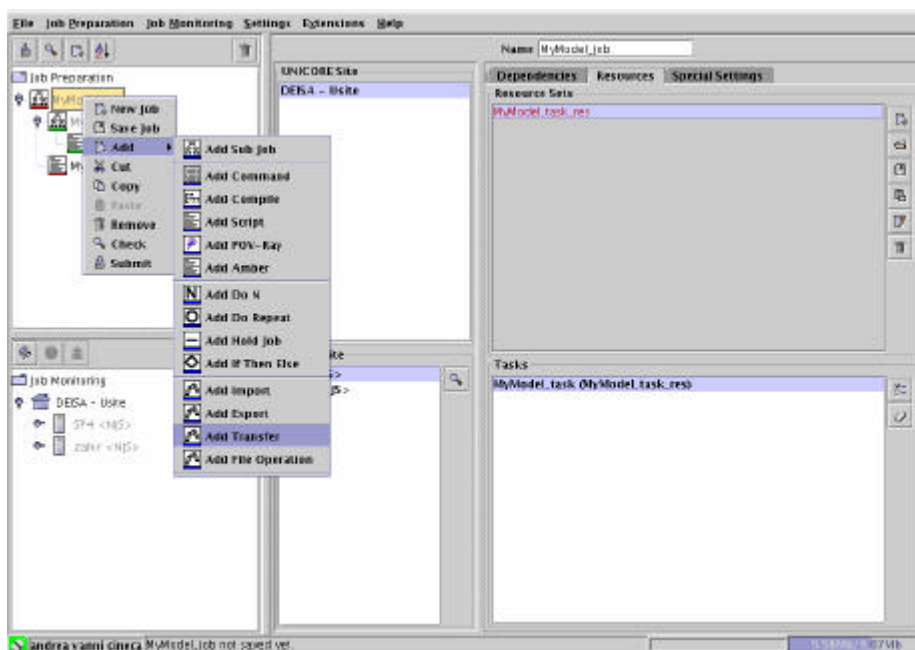


Figure 14: Definition of a transfer task between "Mymodel_task" and "MyPostp_task"

This task copies the two files *MyModel.conf* and *MyModel.output* from the USPACE of “MyModel_job” to the USPACE on “MyPostp_subjob”, as shown in Figure 15; these files have also to be renamed *MyPostp.conf* and *MyPostp.input* respectively.

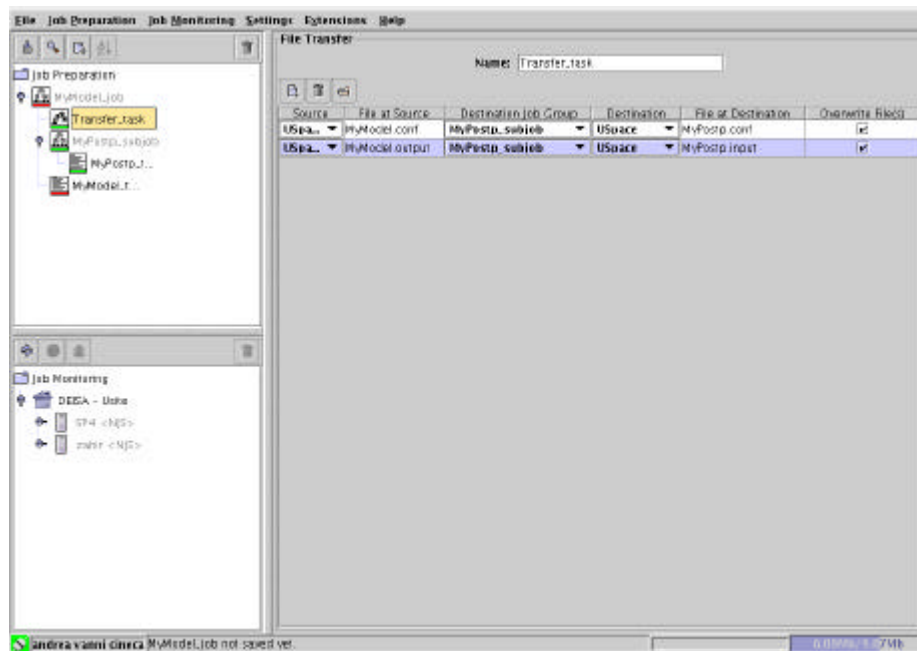


Figure 15: Content of the "Transfer" task

Now you have three tasks:

- ? MyModel_task,
- ? Transfer,
- ? MyPostp_task.

You need to define the workflow between them: indeed, “Transfer” must be run after that “MyModel_task” has completed, and “MyPostp_task” has to run after that “Transfer” has completed. This can be done inside the dependencies panel of “MyModel_job”, as shown in Figure 16. The arrows are created by right-clicking on the icon of the predecessor and dropping it to the successor.

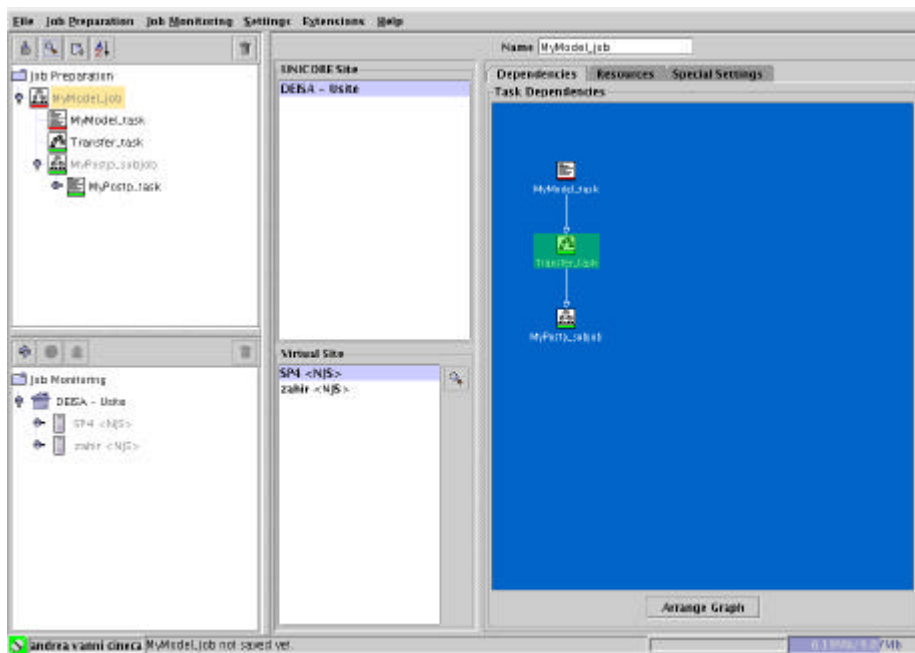


Figure 16: Definition of the dependencies between the three tasks

Now the job is completely defined, and can be submitted through the UNICORE client, as shown in Figure 17.

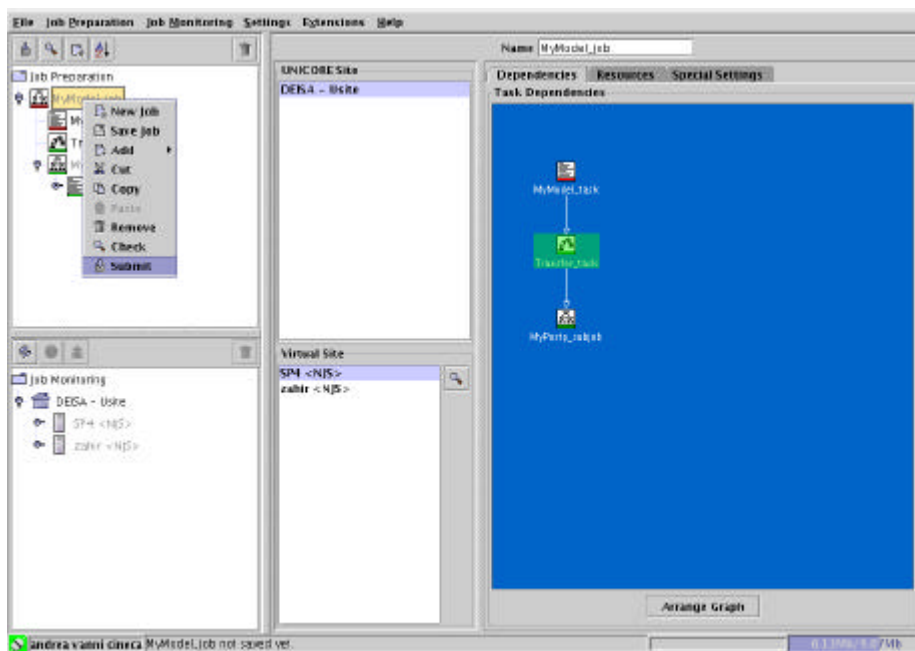


Figure 17: Submission of the job

The job can then be monitored using the lower-left “job monitoring” panel. The abstract job can also be saved as a local file, for future submission.

5.3 Job submission through Web portals

This will be available in the future. Portals and Web interfaces are critical to enhance the user adoption of sophisticated supercomputing infrastructures, by hiding from them the complexities of the computational environment.

6. Access to the documentations

Documentation about compilers, tools and libraries is installed at each DEISA site, and is locally maintained. You must refer to their home site in order to find specific documentation.

Local documentation may consist of man pages, Web pages, user manuals, etc...

7. Access to the User Support

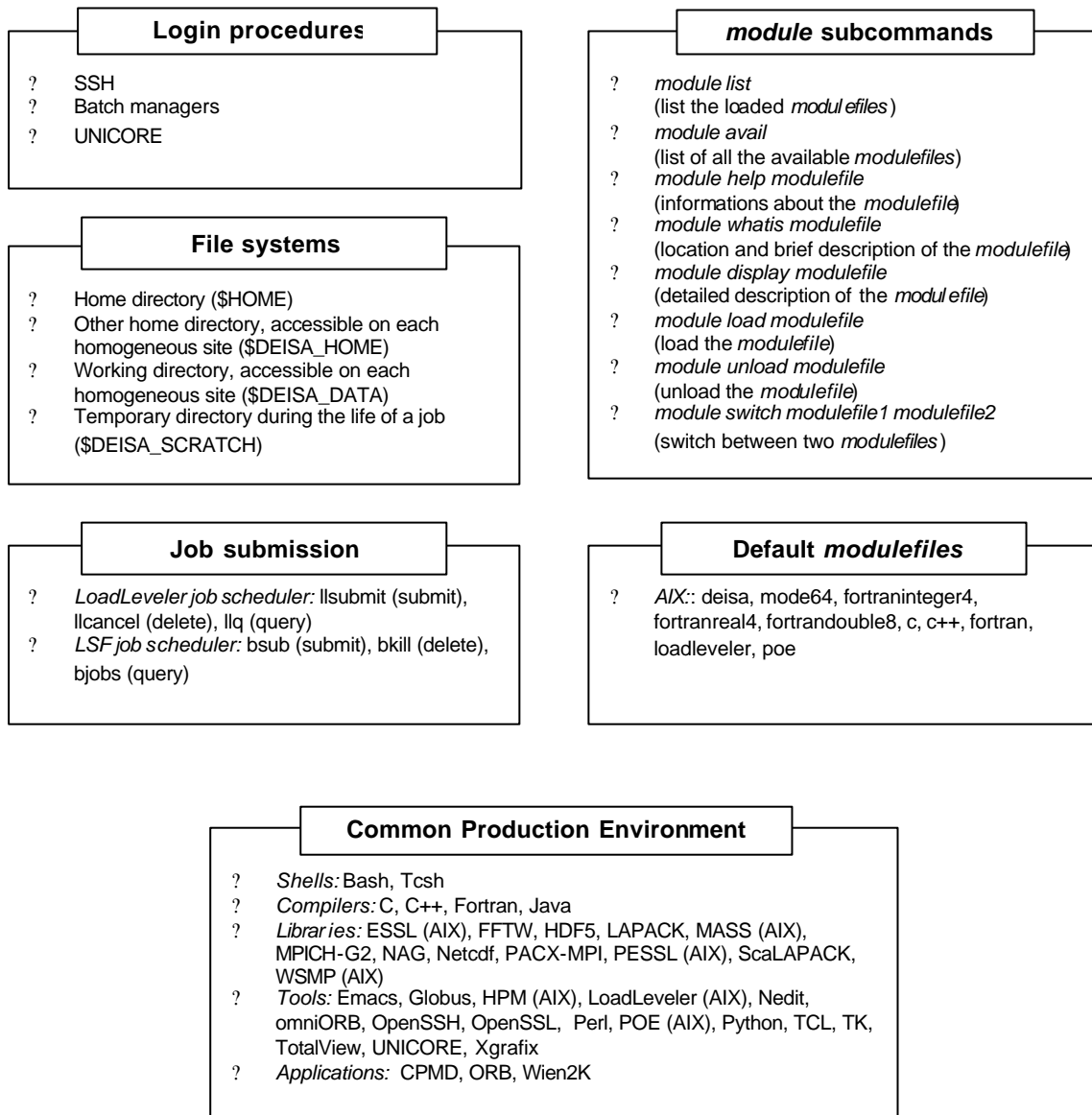
You must refer to your home site for every question regarding the usage of the infrastructure. Every DEISA site has a local user support, which can be contacted by the usual way or by sending an e-mail to a specific mailing list:

Table 8: Local user support mailing list

Site	Mailing list
CINECA	deisa-support@cinca.it
CSC	deisa-support@csc.fi
ECMWF	deisa-support@ecmwf.int
EPCC	deisa-support@epcc.ad.ec.uk
FZJ	deisa-support@fz-juelich.de
IDRIS	deisa-support@idris.fr
LRZ	deisa-support@lrz-muenchen.de
RZG	deisa-support@rzg.mpg.de
SARA	deisa-support@sara.nl

So, if you have an account at IDRIS, for any problem or question you have to write to: deisa-support@idris.fr. If the question involves also other sites, it will be due to the IDRIS support to contact these sites.

8. Quick Reference



9. Glossary

CPE (Common Production Environment): The softwares available on all the sites of the DEISA infrastructure (Some softwares may be installed only in some subgroups of homogeneous computers, and some commercial and licensed softwares may be available on some platforms only).

GEANT: Multi-gigabit pan-European data communication network. This is a collaboration between 26 National Research and Education Networks representing 30 countries across Europe.

GUI (Graphical User Interface): A graphical interface which allows users to execute some programs, specifying interactively the required parameters

Home site: The site where you usually work, log in and submit jobs.

JRA (Joint Research Activities): The activities of the DEISA project dedicated to adapting specific scientific applications to the distributed infrastructure.

NREN (National Research and Educational Network): A national academic network, used by people working in public research laboratories or in universities.

PKI (Public Key Infrastructure): A collection of services and components to manage trust relationships using public keys. This is required for certificates used by some applications in order to provide authentication and confidentiality of the resources and communications.

UNICORE Usite: A site offering a UNICORE server and target system(s).

UNICORE Vsite: Execution system, or a cluster of execution systems, at a UNICORE Usite.

10. References

GEANT: Multi-gigabit pan-European data communication network
(<http://www.geant.net/>)

Modules: Environment Modules package, for the dynamic modification of a user's environment (<http://modules.sourceforge.net/>)

UNICORE: UNiform Interface to COmputing REsources (<http://unicore.sourceforge.net/>)

11. Index

account.....	3, 4, 37	DEISA_HOME.....	6, 7, 15, 21, 24, 25
certificate.....	4, 27	DEISA_SCRATCH.....	7, 24
<i>Common Production Environment</i>	7, 8, 9, 16, 17, 38	LoadLeveler	5, 15, 23, 24, 25, 26
DEISA_CFLAGS	20, 21	<i>modulefile</i> .	10, 11, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23
DEISA_CLIBS	20, 21	<i>Modules</i>	9, 10, 13, 14, 16, 17, 19, 39
DEISA_DATA.....	6, 7, 24	UNICORE... ..	5, 23, 25, 26, 27, 28, 29, 36, 39
DEISA_FFLAGS ...	13, 14, 15, 20, 21, 22		
DEISA_FLIBS	18, 19, 20, 21		