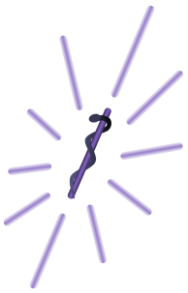
MediGRID

Networked computing resources for biomedical research



Guidelines for the Deployment of Applications in MediGRID

- Promotion number: 01AK803H -

“eScience and GRID Middleware to support the scientific working”
of the German federal ministry for education and research (BMBF)

Date	**06.12.2007**
Version	**1.2**
Type	**Public**
Status	**Internally approved**
Authors	**Andreas Hoheisel (Fraunhofer FIRST)** **Dietmar Sommerfeld (GWDG)** **Kathrin Peter (ZIB)**

Table of Content

1 Introduction

The MediGRID project, which is part of the German e-Science Framework (D-Grid), was formed to build a Grid computing environment for medicine and life sciences. Started in 2005, the MediGRID consortium's main goal is the development of a Grid middleware integration platform enabling e-Science services for biomedical life sciences.

The purpose of this document is to define guidelines for deploying applications on MediGRID resources, in order to make them available to end users. The main target audience of this document is the group of persons that is responsible for the MediGRID application development.

In the following, we first introduce the MediGRID middleware architecture and the personal roles which are involved in the deployment process. Chapter 2 then specifies a software component model, and Chapter 3 defines the software *deployment* process. The software *development* process, including (external) testing and Quality of Service procedures, is not within the scope of this document.

1.1 Middleware Architecture

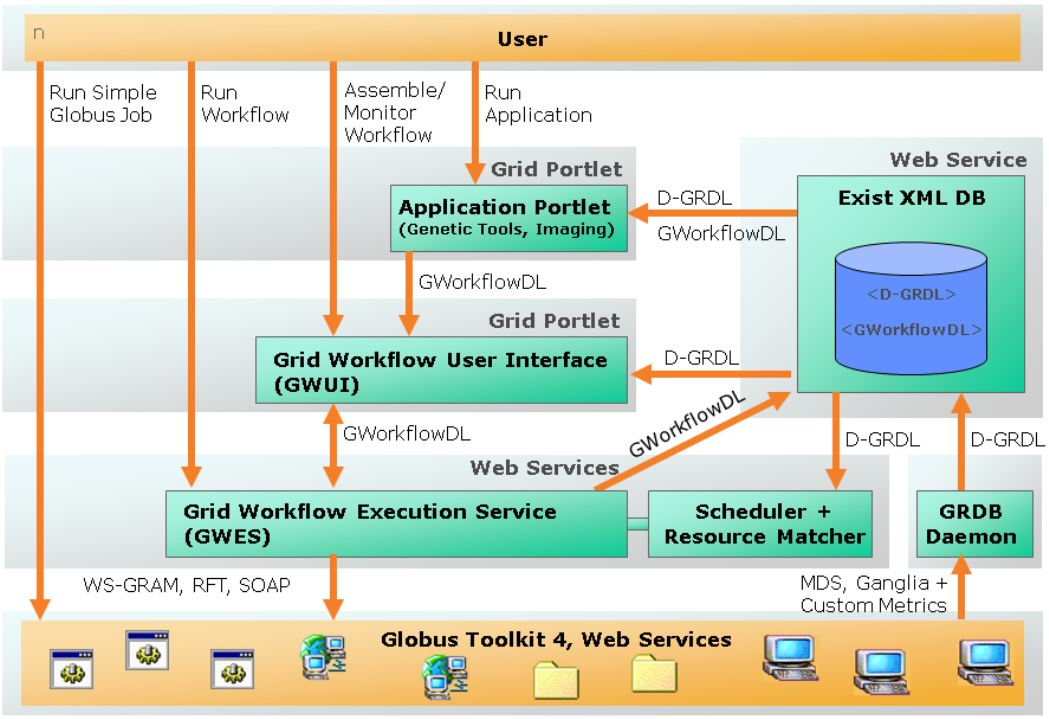


Figure 1. The layered middleware architecture of MediGRID.

The MediGRID middleware is composed of several architectural layers as shown in Figure 1. The end user normally accesses the application by means of an Application Portlet. Based on the users' input data, this portlet generates *GWorkflowDL* documents describing the

workflow of the application. The workflows can be visualized and monitored by means of the *Grid Workflow User Interface (GWUI)*. The execution of workflows is delegated to the *Grid Workflow Execution Service (GWES)*, which hides the technical details of the underlying Grid infrastructure. The *Resource Matcher* uses the resource information stored in the *eXist XML DB* in order to automatically map abstract workflow activities onto suitable and available hardware and software. The *Meta-Scheduling* is done on the basis of current information about the queue and load status of the Grid nodes. The submission of activities to hardware resources is done via WS-GRAM, RFT, or SOAP.

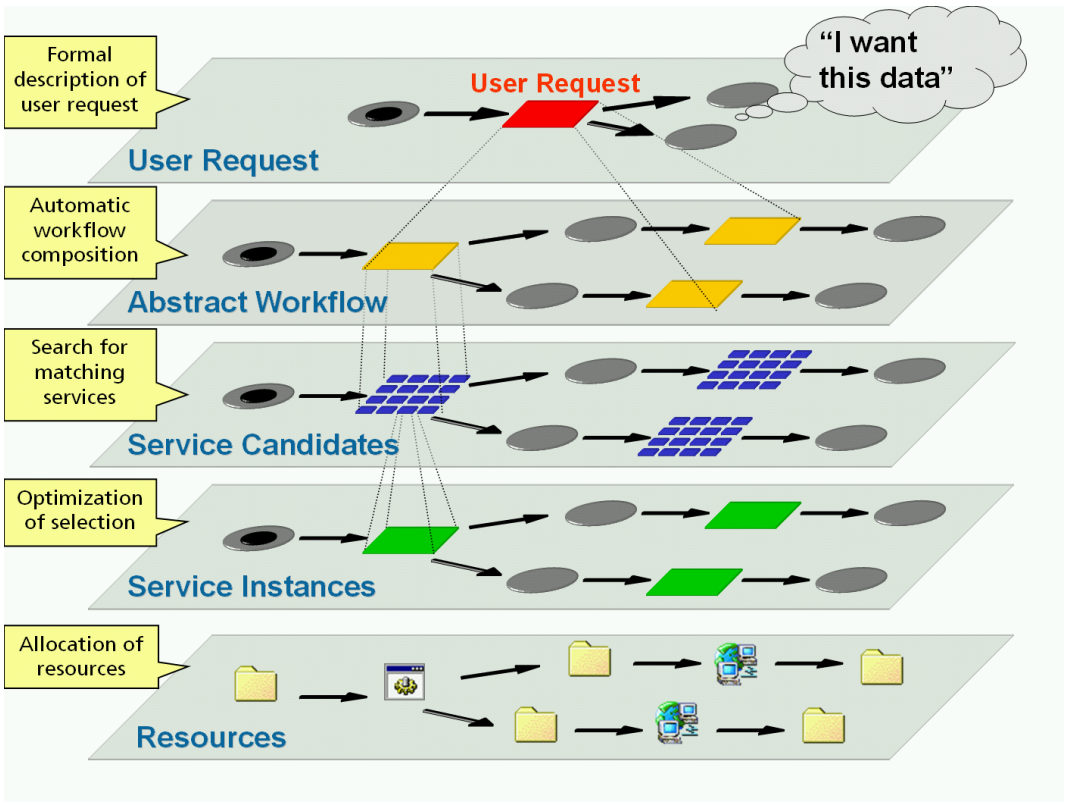


Figure 2. Automatic mapping of user requests onto available Grid resources in MediGRID.

Figure 2 represents the mapping process for workflow applications within MediGRID, beginning with an application-specific user request and ending in a workflow instance which allocates specific hardware and software resources. In the first step, the application portlet generates an abstract workflow (yellow) based on the input data as well as additional parameters and context information provided by the user. The Resource Matcher then maps each abstract workflow activity onto a set of service candidates (blue workflow). During the runtime of the workflow, the Meta-Scheduler selects one of the candidates (green workflow) and allocates the resources. The colors mentioned here are also used as border colors for transitions within the Petri Net representation of the Grid Workflow User Interface.

1.2 MediGRID roles

The following roles currently exist in MediGRID and are mapped onto specific account types which are partially described in more detail in the MediGRID resource usage policy:

1. Portal Administrator
2. Resource Administrator
3. Developer (Application Developer)
4. eXist XML DB Administrator
5. GWES Administrator
6. User
7. Guest

The roles 4 (eXist XML DB Administrator) and 5 (GWES Administrator) have no specific account types within the MediGRID resource usage policy and have been introduced here additionally as they are important for the software deployment process. The corresponding contact persons for each role are listed in Appendix A on page 21.

Regarding the deployment of applications in MediGRID, the main responsible role is the *(Application) Developer*, who may need support by the *Resource Administrator*, the *Portal Administrator*, the *eXist XML-DB Administrator,* and the *GWES Administrator* in order to deploy the software for *Users* and, optionally, for *Guests*.

2 Software Component Model

The software component model for applications within MediGRID is currently restricted to non-interactive command line programs or Web Services that work in batch mode without own graphical user interfaces. Several of these command line programs or Web Services can be coupled within a more complex application workflow and be automated using the Grid Workflow Execution Service as shown in Section 3.4.

What are Software Components?

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.“ European Conference on Object-Oriented Programming (ECOOP), 1996 [Szyperski, 1999, p. 34]

“Components are for composition.“ [Szyperski, 1999, p. 3]

Following the definitions above, we support two different kinds of software components: command line programs and Web Services. The following two sections describe the underlying component models used for the composition of workflow-based Grid applications in MediGRID.

2.1 Command Line Programs

Command line programs are software components that are usually loosely coupled with other programs using input files, output files, standard input (stdin), standard output (stdout),

and standard error (stderr) as communication channels. In order to enable the automatic coupling of command line programs within MediGRID, we specify the following restrictions:

- A command line program can be treated as "**blackbox**" with defined input and output data. During the execution, the program reads input data and writes output data.
- The command line program is **stateless**, that means that invoking the same program twice with the same input data always results in the same output data. The necessary job-specific configuration is done by input data. (A concept for considering stateful programs in workflows is available but not yet implemented for MediGRID).
- **Input data** is read from standard input (stdin), or from input files, which can be dynamically defined using command line parameters, or from additional command line parameters.
- **Output data** is written to standard output (stdout), or standard error (stderr), or to output files, which can be dynamically defined using command line parameters.
- The **exit status** of the application must be "0" on success and "unequal 0" if the program execution was not successful.
- For MediGRID, currently only **Unix or Linux** command line programs are supported
- The program should be mostly **independent from external entities**, such as system environment variables (e.g. $JAVA_HOME, $PATH), external libraries (dynamic binding), and license servers. Often, the execution environment on the Grid/Cluster nodes is very limited (no environment variables, no connection to the internet, etc.).
- The workflow system executes the program using a standardized (automatically generated) command line syntax which is specified as follows:

```
<executable> [-<portId1> <directory>/<filename>] \
             [-<portId2> <directory>/<filename>] \
             [-<portId3> <directory>/<filename>] \
             [...] \
             [-<paramNameX> <paramValueX>] \
             [...]
```

`<executable>` Executable of the command line program, e.g. `augustus.sh`

`<-portIdX>` Identifier of the input or output port, e.g. `-input1`.

`<directory/filename>`
Absolute path to the file which should be connected to the corresponding input/output port.

`<-paramNameX> <paramValueX>`
Program options, e.g. `-list all`.

There are three reserved port identifiers which are handled by the Grid middleware and the operating system and which should not be used as command line parameters within the command line program: `-stdin, -stdout, -stderr`

The command line parameters have no specific order; this means that the parameters are identified by their port identifier (parameter name) and not by their sequence.

Example of a valid command line syntax:

```
augustus.sh -input2 /data/in2.dat -input1 /data/in1.dat -list all \
            -out /tmp/out.dat
```

- If the command line program itself does not fulfill the specified command line syntax, the application developer must provide a wrapper script (using bash) which accomplishes the translation between the parameters of the legacy program and the component model. Here an example of such a wrapper script which wraps the Linux program “cat”:

```
#!/bin/bash
export PATH="/bin:/usr/bin:usr/local/bin"
CAT=`which cat`

function usage {
  echo "### USAGE: cat.sh -input1 <FILENAME1> -input2 <FILENAME2>"
}

while [ $# -gt 1 ]
do
  case $1 in
    -input1)
      input1=$2
      shift 2
      ;;
    -input2)
      input2=$2
      shift 2
      ;;
    *)
      echo "### Error: unknown argument $1 ###"
      usage
      exit 1
      shift
      ;;
  esac
done

if [ -r "$input1" -a -r "$input2" ]; then
  exec ${CAT} $input1 $input2
else
  echo "### ${CAT} $input1 $input2"
  echo "### Error: the input files are not specified or not available! ###"
  usage
  exit 1
fi
```

- If you want to use automatic installation and execution using the “generic-executor” (refer to Chapter 3.4), you need to pack the command line program and all dependent files into a “tgz” archive, e.g. using:

```
tar -czf <executable>.tgz <application-dir>/*
```

The name of the tgz archive must be the same as the executable plus the suffix “.tgz”. The executable must be in the top level directory of the tgz archive.

2.2 Web Services

As a second type of software components, the MediGRID workflow management supports standard stateless Web Services that use the SOAP protocol in RPC mode and HTTP as transport layer. The service must be described properly in a WSDL in “flat” format, available at a given URL. Currently, only basic SOAP types are supported as input and output parameters for Web Service method calls within workflows. More complex (e.g. stateful) Web Services, such as Globus Toolkit 4 Grid Services, including GSI layer, are only supported on request. Please ask the GWES Administrator for details.

3 Software Deployment Process

For the MediGRID middleware architecture, the software deployment process can be structured into several sub-processes, regarding the different architectural levels. On the bottom layer, there are the command line programs, files, and Web Services to be installed on the distributed Grid resources, which are the basic components. On top of these components, the application developers may define higher-level workflows which are provided to the users via application-specific user interfaces. One guiding principle of the software deployment process in MediGRID is that the responsible application developers themselves deploy, test, and maintain the application’s basic software components on the Grid nodes, in order to ensure the quality of service using their domain knowledge.

3.1 Deployment of Command Line Programs

In MediGRID, there are currently two main alternatives to install and invoke command line programs. For small and simple applications, you may use the “generic-executer” to automatically deploy the program during runtime of the workflow. We describe this procedure later in Chapter 3.4 “Application Workflows”. For bigger and more complex applications, the software should instead be preinstalled on the target Grid nodes as shown in Figure 3 in order to reduce the installation overhead during runtime of the workflow and to test the software.

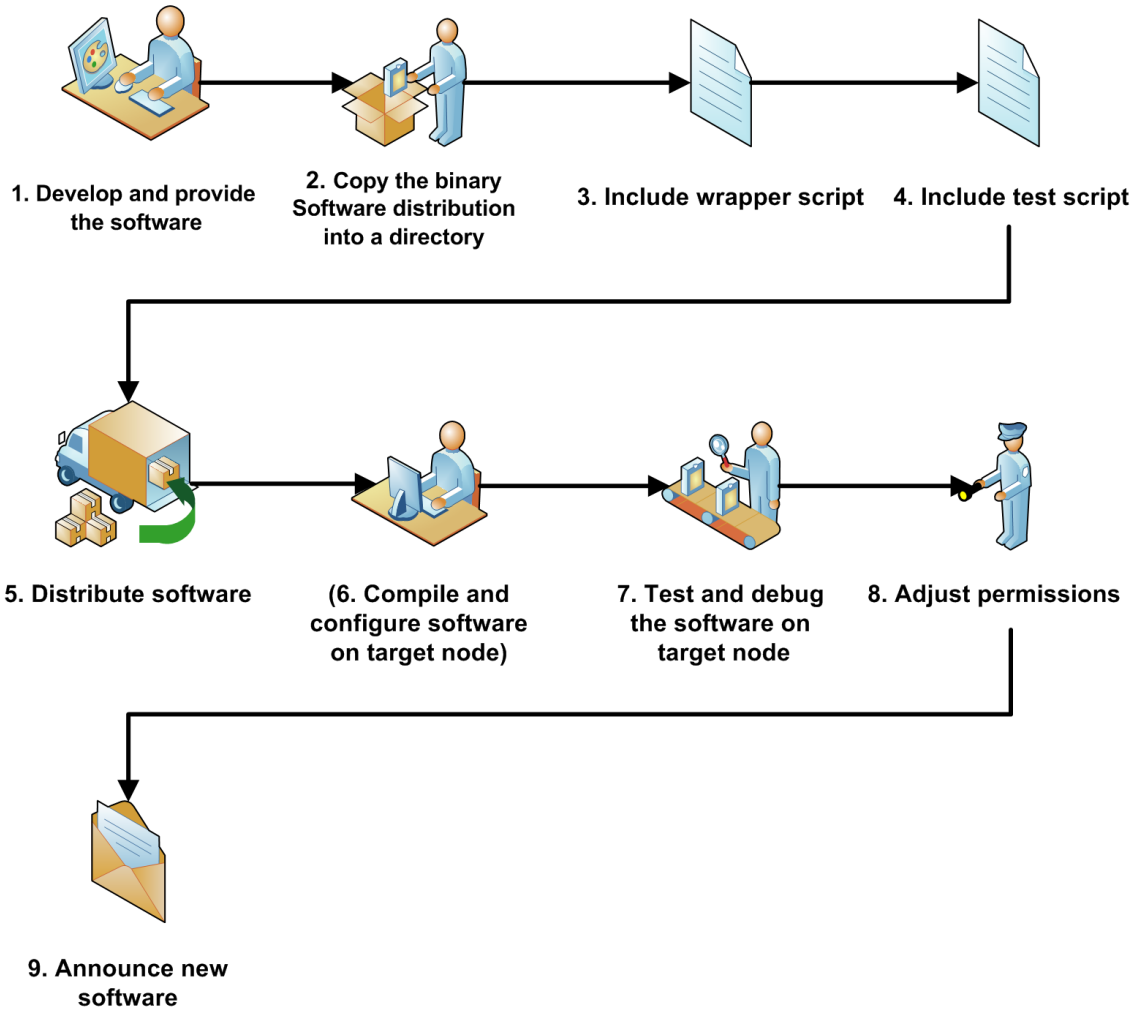


Figure 3. Diagram representing the manual software deployment process for command line programs in MediGRID.

Below, we describe the process of how to manually install software on MediGRID nodes in detail:

1. **Provide the software** in a self-contained way, ideally reducing external dependencies to other software components, such as libraries (use static linking), databases (include them in the software component), license servers, etc. The software should follow the software component model described in Chapter 2 and should be well tested before deploying it in the Grid.

2. **Directory**: Copy the binary software distribution into a directory named
   `<application>[ <submodule>]-<version>`
   using lower case letters without spaces and other special characters (such as umlauts), e.g.,
   `augustus-0.9.1`

   `<application>` Name of the application

`<submodule>`	(Optional) If the application is structured into submodules either provide the corresponding submodule name here or include all submodules in subdirectories within the main application.
`<version>`	Version of the application. Please do not modify or delete old versions of your software, as they may be needed for backward compatibility.

Within this document, this application directory is referred to as `<application-dir>`.
The directory must be located on a file system which can be accessed from a Globus Toolkit 4 Grid node (including gsiscp and gsissh) with a valid D-Grid user certificate. The Grid node is required to trust the CA of the host certificates of the MediGRID nodes where the software is to be deployed (refer to Appendix B).

3. **Wrapper script**: Include a wrapper shell script (bash) regarding the software component model (refer to Chapter 2), e.g. `augustus-0.9.1/augustus.sh`

4. **Test script**: Include a test script named `test.sh` that can be invoked without any additional command line parameters and which returns:
   `exit code 0`: if installation works fine (OK)
   `exit code 1`: if installation has some non-fatal problems (WARNING)
   `exit code 2`: if installation has fatal problems (CRITICAL)
   `exit code 3`: if script was not able to perform the test (UNKNOWN)
   These exit codes are required to facilitate the integration of the test script within the monitoring system “nagios”. Ensure that all persons from the medigrid group have the permissions to invoke the test script, e.g., create temporary data only in `/tmp` or `/opt/medigrid/tmp`.
   In addition, the developer should provide a clear script named `clear.sh` which removes temporary files generated by the test script, if not done by the test script itself.

5. **Distribute software**: There are two alternatives to distribute software on remote MediGRID nodes, depending on whether the group “medigrid” and the responsible application developer (refer to Appendix A) has write permissions for the directory `/opt/medigrid` and direct access to the Grid node using gsissh/gsiscp (alternative a) or not (alternative b). A list of available MediGRID nodes is available in Appendix B:

   a. Requirements: Access via gsissh/gsiscp to target Grid node and write permissions for /opt/medigrid.
      Use `gsiscp` to copy the software distribution from a local Grid node to the target MediGRID node:

```
grid-proxy-init
gsiscp -r -P 2222 <application-dir> <grid-node>:/opt/medigrid/
```

`<application-dir>`	local application directory
`<grid-node>`	hostname of remote Grid node (Appendix B)

Repeat this command for all Grid nodes where you want to deploy the software.

b. Pack the software distribution (.tgz) and send it to the corresponding resource administrator (use e-mail signed with your personal D-Grid certificate) and ask him or her to install the package in /opt/medigrid and to change the ownership of the application directory to the responsible application developer UID.

6. **Optional: Compile and configure software** on target Grid node. If your software has strong dependencies to external libraries or other software components, it may be necessary to compile and to configure the software on the target Grid node. If this is the case, distribute the source code of your software as described in step 5 and logon to the corresponding interactive node using `gsissh` in order to compile and to configure the software:

```
grid-proxy-init
gsissh -p 2222 <grid-node>
cd /opt/medigrid/<application-dir>
# make…
# configure…
```

Please note that for Clusters, the environment of the Grid (head) node may differ from the environment of the Cluster nodes.

7. **Test and debug the software** on target Grid node:

```
# test software
grid-proxy-init
gsissh -p 2222 <grid-node>
# on Grid nodes with fork
/opt/medigrid/<application-dir>/test.sh
# on Grid nodes with PBS
qsub /opt/medigrid/<application-dir>/test.sh
```

If the Grid nodes use PBS to distribute the job on a Cluster, the environment on the Grid node may differ from the environment of the Cluster nodes. If you want more information about the environment use the following commands:

```
# show information about target environment
grid-proxy-init
gsissh -p 2222 <grid-node>
# information about head node
/opt/medigrid/utils/show-environment.sh
# information about Cluster nodes
qsub /opt/medigrid/utils/show-environment.sh
```

8. **Adjust permissions**: Only the responsible application developer should be allowed to change the code using his personal D-Grid certificate. Persons from the group “medigrid” should be able to execute the application; others should not have any permissions.

```
# adjust permissions
grid-proxy-init
```

```
gsissh -p 2222 <grid-node>
chmod -R u=rwX,g=rX,o= /opt/medigrid/<application-dir>
# chmod +x for shellscripts and other executables
chmod ug+x /opt/medigrid/<application-dir>/*.sh
# chmod ug+x /opt/medigrid/<application-dir>/…
```

9. **Announce new software**: Please send an email to the developers' mailing list (developer@medigrid.de) with the name of the new application, the name of the directory where it is installed (`<application-dir>`), the name and the email address of the responsible application developer, and a short description of the software.

Now, in principle the software is ready to be used by all users within MediGRID which have direct access to the Grid nodes. If you want to access the software from within a Workflow, you also need to follow the procedure steps described in Chapters 3.3 and 3.4.

3.2 Deployment of Web Services (SOAP Services)

Currently, most of the Web Services deployed in MediGRID are part of the Grid Middleware stack, and therefore it is in the duty of the corresponding *middleware service administrators* to deploy and to maintain these Web Services in agreement with the resource administrators, which is beyond the scope of these guidelines.

Actually, there is no established process for the deployment of *application* Web Services by *application developers* in MediGRID. The deployment should be done on a case-to-case basis in collaboration with the corresponding resource administrators. If the Web Service is able to run in user-space, the application developer may follow the process for command line programs described in Section 3.1 (see Figure 3) to deploy the Web Service together with its Web Service Container (e.g., tomcat, jetty). In addition, the application developer should contact the resource administrators directly to ensure that the Web Service container is started automatically (e.g., within the boot process of the corresponding hardware resource), and that the Web Service port (e.g. 8080) is free and the firewall is open for traffic to this port.

In case that application Web Services become more common in MediGRID, we will extend this Section for providing more specific guidelines for the deployment of Web Services.

3.3 Resource Matching

The resource matching service of MediGRID facilitates locating suitable IT resources for executing complex processes. For executing software, you just need to specify the software class, and the resource matching service returns all available software/hardware pairs within MediGRID which fulfill the requirements. The resource matching algorithm is based on a XML resource description language, named "D-GRDL", developed in the DGI project within D-Grid. More information about the D-GRDL is available at:

- D-GRDL XML schema (public):
  http://www.gridworkflow.org/kwfgrid/src/xsd/resource-d-grdl.xsd
  http://portal.medigrid.izbi.uni-leipzig.de:9081/gwes/xsd/resource-d-grdl.xsd
- Specification (D-Grid internal Web page):
  https://www.d-grid.de/index.php?id=118&no_cache=1&filename=D-GRDL-Spezifikation-2007-03.pdf&dir=FG2/Kern-D-Grid/Dokumente&task=download&mountpoint=2
- Tutorial (D-Grid internal Web page):
  https://www.d-grid.de/index.php?id=118&no_cache=1&filename=d-grdl-tutorial.pdf&dir=FG2/Kern-D-Grid/Dokumente&task=download&mountpoint=2

In order to enable the resource matching for your software, you need to provide an XML description of the software as well as a reference to this software within each hardware description where the software has been deployed. In a later phase of the MediGRID project, we plan to provide a more user-friendly D-GRDL Portlet in order to edit this information using the Grid Portal.

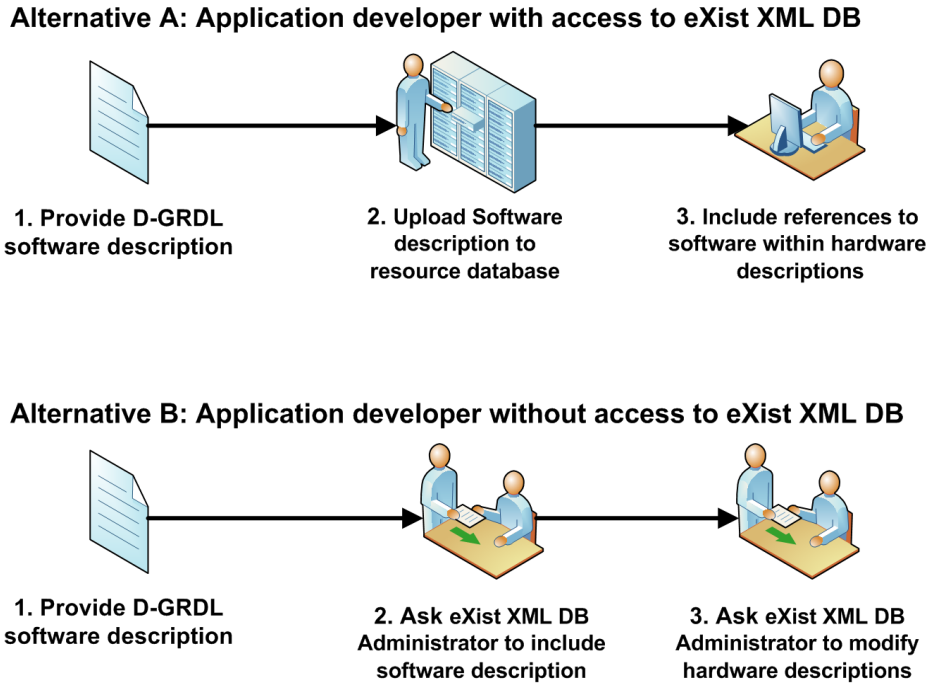


Figure 4. Diagram representing the process for enabling the automatic resource matching within MediGRID.

In the following, we describe the process for enabling the automatic resource matching within MediGRID which is also depicted in Figure 4:

1. **Provide software description in D-GRDL-Syntax** as XML file with the name
   `software medigrid-<application>[ <submodule>]-<version>.xml`
   Replace dots "." in `<version>` by hyphens " –", e.g.,

`software_medigrid-augustus-0-9-1.xml`
Example:

```
<resource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.gridworkflow.org/kwfgrid/src/xsd/resource-d-
grdl.xsd"
uri="software:medigrid-augustus-0-9-1">
    <ofClass uri="urn:dgrdl:software:medigrid-augustus-0-9"/>
    <name>Augustus</name>
    <description>AUGUSTUS gene prediction program</description>
    <simpleProperty ident="executable" type="string" unit="">
      /opt/medigrid/augustus-0.9.1/augustus.sh
    </simpleProperty>
</resource>
```

The element
`<ofClass uri="urn:dgrdl:software:medigrid-augustus-0-9"/>`
specifies the name of the software class which is used to refer to this software from within workflows. The attribute `uri="software:medigrid-augustus-0-9-1"` specifies the specific software installation, which is used in the <provides> section of hardware resources.

2. **Upload D-GRDL document to resource database** or send it to the eXist XML DB Administrator if you do not have write access to the database. Use the eXist Java WebStart Client to upload the D-GRDL document:

```
# start eXist Java client
javaws http://portal1.medigrid.izbi.uni-leipzig.de:9050/exist/webstart/exist.jnlp
# login with the username/password provided by the database administrator
# goto /db/dgrdl/resources
# upload D-GRDL software description: File -> Store files/directories
```

3. **Include references to the software within the corresponding hardware resource descriptions**, or send the list of (hardware uri/software uri) pairs to the eXist XML DB Administrator and ask him or her to modify the descriptions, if you do not have write access:

```
# start eXist Java client
javaws http://portal1.medigrid.izbi.uni-leipzig.de:9050/exist/webstart/exist.jnlp
# login with the username/password provided by the database administrator
# goto /db/dgrdl/resources
# double click on hardware resource description and edit the <provides> section
# include the resourceRef, e.g.,
<resourceRef uri="software:medigrid-augustus-0-9-1"/>
# file -> save
# close eXist Java client
```

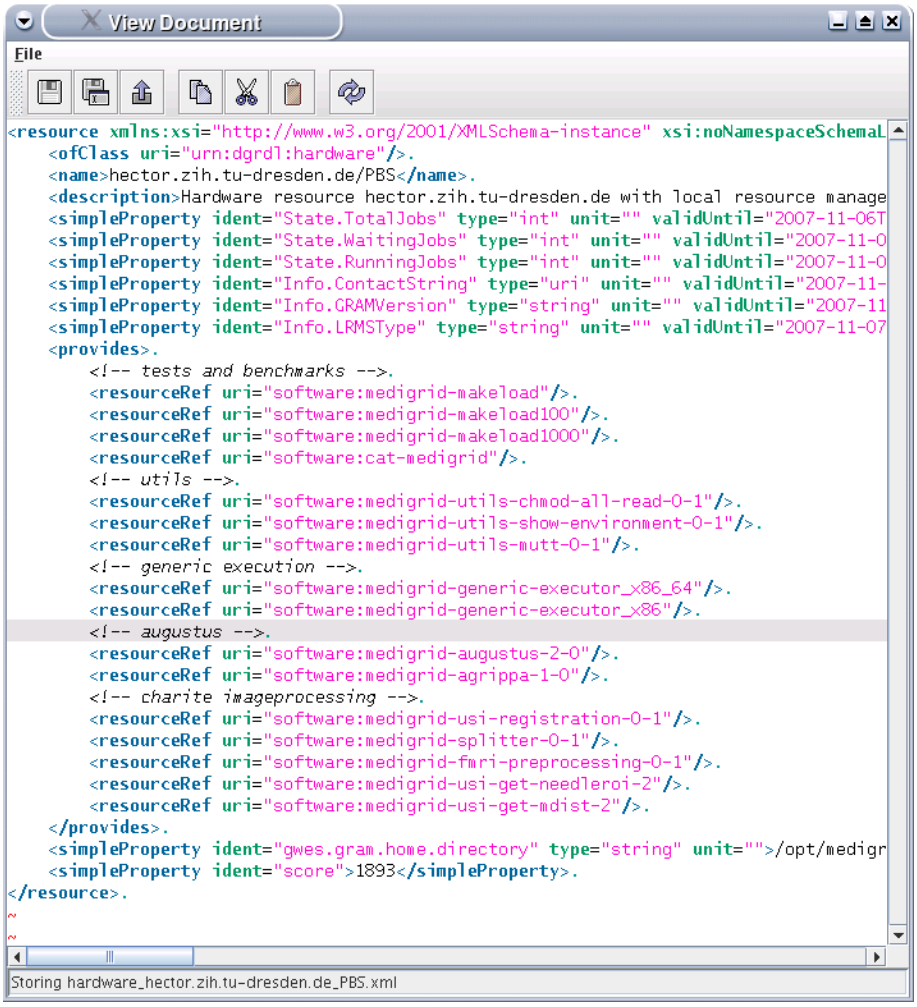


Figure 5. Screenshot of the eXist Java WebStart Client for editing D-GRDL documents.

3.4 Application Workflows

This section gives two simple examples of how to refer to software components from within workflow documents. For more information about the usage of the GWorkflowDL to model workflows, please refer to the documentation on the home page of the Grid Workflow Execution Service at http://www.gridworkflow.org/gwes.

Workflows can be used to provide a suitable virtualization and automation of IT processes within MediGRID. The following two figures show how to invoke two times the software “cat” within a workflow, using the `stdout` of the first invocation as `input1` for the next invocation. Please note that it is not required to specify the hardware resources within the workflow document:

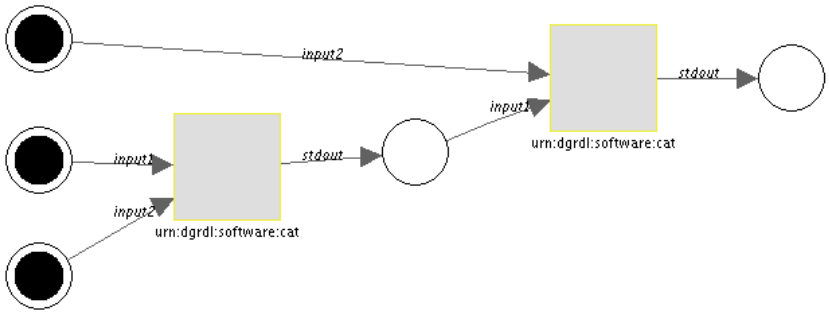


Figure 6. Graphical representation of a workflow which invokes two times the software class „urn:dgrdl:software-cat“.

```
<workflow xmlns="http://www.gridworkflow.org/gworkflowdl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.gridworkflow.org/gworkflowdl http://portal.medigrid.izbi.uni-
leipzig.de:9081/gwes/xsd/gworkflowdl_1_0.xsd" ID="No_ID">
  <description>small abstract workflow that invokes two executables with input files via WS-
GRAM</description>

  <place ID="d25">
    <token>
      <data>
        <file xsi:type="xsd:string">gsiftp://othello.zih.tu-
dresden.de//opt/medigrid/cat/d25.dat</file>
      </data>
    </token>
  </place>

  <place ID="d26">
    <token>
      <data>
        <file xsi:type="xsd:string">gsiftp://othello.zih.tu-
dresden.de//opt/medigrid/cat/d26.dat</file>
      </data>
    </token>
  </place>

  <place ID="d27">
    <token>
      <data>
        <file xsi:type="xsd:string">gsiftp://othello.zih.tu-
dresden.de//opt/medigrid/cat/d27.dat</file>
      </data>
    </token>
  </place>

  <place ID="d25-26"/>

  <place ID="d25-27"/>

  <transition ID="cat1">
    <description>concatenate two files</description>
    <inputPlace placeID="d25" edgeExpression="input1" />
```

```
    <inputPlace placeID="d26" edgeExpression="input2" />
    <outputPlace placeID="d25-26" edgeExpression="stdout" />
    <operation>
      <pe:programClassExecution
        xmlns:pe="http://www.gridworkflow.org/gworkflowdl/programclassexecution"
        softwareClass="urn:dgrdl:software:cat" />
    </operation>
  </transition>

  <transition ID="cat2">
    <description>concatenate two files</description>
    <inputPlace placeID="d25-26" edgeExpression="input1" />
    <inputPlace placeID="d27" edgeExpression="input2" />
    <outputPlace placeID="d25-27" edgeExpression="stdout" />
    <operation>
      <pe:programClassExecution
        xmlns:pe="http://www.gridworkflow.org/gworkflowdl/programclassexecution"
        softwareClass="urn:dgrdl:software:cat" />
    </operation>
  </transition>

</workflow>
```

Figure 7. The GWorkflowDL corresponding to the workflow graph represented in Figure 6. The two highlighted elements `<programClassExecution>` contain the reference to the software class `urn:dgrdl:software:cat`.

The next two figures show how to use the “generic-executor” application to automatically install and invoke software on MediGRID resources:

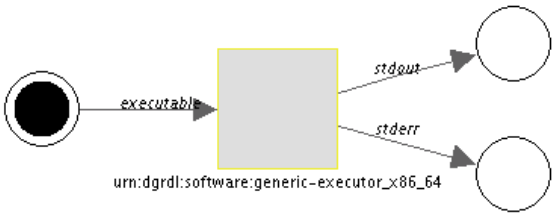


Figure 8. Graphical representation of a workflow which installs and executes a software package using the software class „urn:dgrdl:software:generic-executor_x86_64“. The application executable is modeled as input token of the transition (black circle).

```
<workflow xmlns="http://www.gridworkflow.org/gworkflowdl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.gridworkflow.org/gworkflowdl http://portal.medigrid.izbi.uni-
leipzig.de:9081/gwes/xsd/gworkflowdl_1_0.xsd" ID="No_ID" >
  <description>This workflow automatically deploys and executes the application
"date"</description>

  <place ID="executable">
    <token>
      <data xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        <file xsi:type="xsd:string">
          gsiftp://othello.zih.tu-dresden.de//opt/medigrid/generic-executor/date.tgz
```

```
        </file>
      </data>
    </token>
  </place>

  <place ID="stdout"/>

  <place ID="stderr"/>

  <transition ID="generic-execution">
    <description>generic execution</description>
    <inputPlace placeID="executable" edgeExpression="executable" />
    <outputPlace placeID="stdout" edgeExpression="stdout" />
    <outputPlace placeID="stderr" edgeExpression="stderr" />
    <operation>
      <pe:programClassExecution
xmlns:pe="http://www.gridworkflow.org/gworkflowdl/programclassexecution"
softwareClass="urn:dgrdl:software:generic-executor_x86_64" />
    </operation>
  </transition>

</workflow>
```

Figure 9. The GWorkflowDL corresponding to the workflow graph represented in Figure 8. The highlighted data element (green) represents the software package "date.tgz", which should be installed and executed automatically by the "generic-executor" application (highlighted red).

3.5 Deployment of Application-Specific User Interfaces

The top layer between the user and the Grid application is the application-specific user interface. There are several possibilities to provide end users with a suitable interface to Grid applications. In MediGRID, a common way is to implement application-specific "portlets" following the JSR 168 standard, and to deploy them in the MediGRID portal which is based on the GridSphere portlet technology. All security issues, such as authentication, authorization, and encryption are then handled by the portal and the corresponding browser. The portlets hide the technical complexity of the Grid application and provide a domain-specific user interface to the application. The portlet itself may delegate the execution of pre-defined and user-customized workflows to the Grid Workflow Execution Service (GWES) using the Java GWES Client API or by directly connecting to the GWES using SOAP. Best practice examples that demonstrate how to implement application-specific portlets are available on request from the Portal Administrator. This section focuses on the deployment of user interfaces, which have been implemented regarding the JSR 168 standard using GridSphere technology. Other kinds of user interfaces, such as Servlets or client applications, are not within the focus of these guidelines.

The following steps can be used as a guideline for implementing and deploying application-specific portlets:

1. Download the GridSphere release version 2.2.x from http://www.gridsphere.org/
2. Unpack GridSphere, and follow the documentation at http://www.gridsphere.org/gridsphere/docs-GS-2.2.X/gridportlets/ to implement the

portlet. Some MediGRID best practice examples for implementing MediGRID portlets are available from the Portal Administrator. For additional support from the MediGRID community, please consult the MediGRID developer mailing list developer@medigrid.de.

3. If the portlet is related to the execution of workflows, it is a good idea to use the StringTemplate library (http://www.stringtemplate.org/) to build specific GWorkflowDL documents from workflow templates.

4. Workflows can be executed by using the GWES Client API available from the GWES Administrator. The GWES interface description is available at http://www.gridworkflow.org/kwfgrid/gwes/docs/apidocs/net/kwfgrid/gwes/GWES.html. Another less comfortable alternative is to directly connect to the GWES using SOAP. The corresponding WSDL is available at http://portal1.medigrid.izbi.uni-leipzig.de:9081/gwes/services/GWES?wsdl

5. Test and debug the portlet on your local GridSphere installation.

6. Ask the Portal Administrator for access to the test portal at http://portal2.medigrid.izbi.uni-leipzig.de:8080/gridsphere/gridsphere. Deploy, test, and debug your portlet on the test portal.

7. If the portlet works stable on the test portal, ask the Portal Administrator to deploy the portlet on the production portal (https://portal.medigrid.de/).

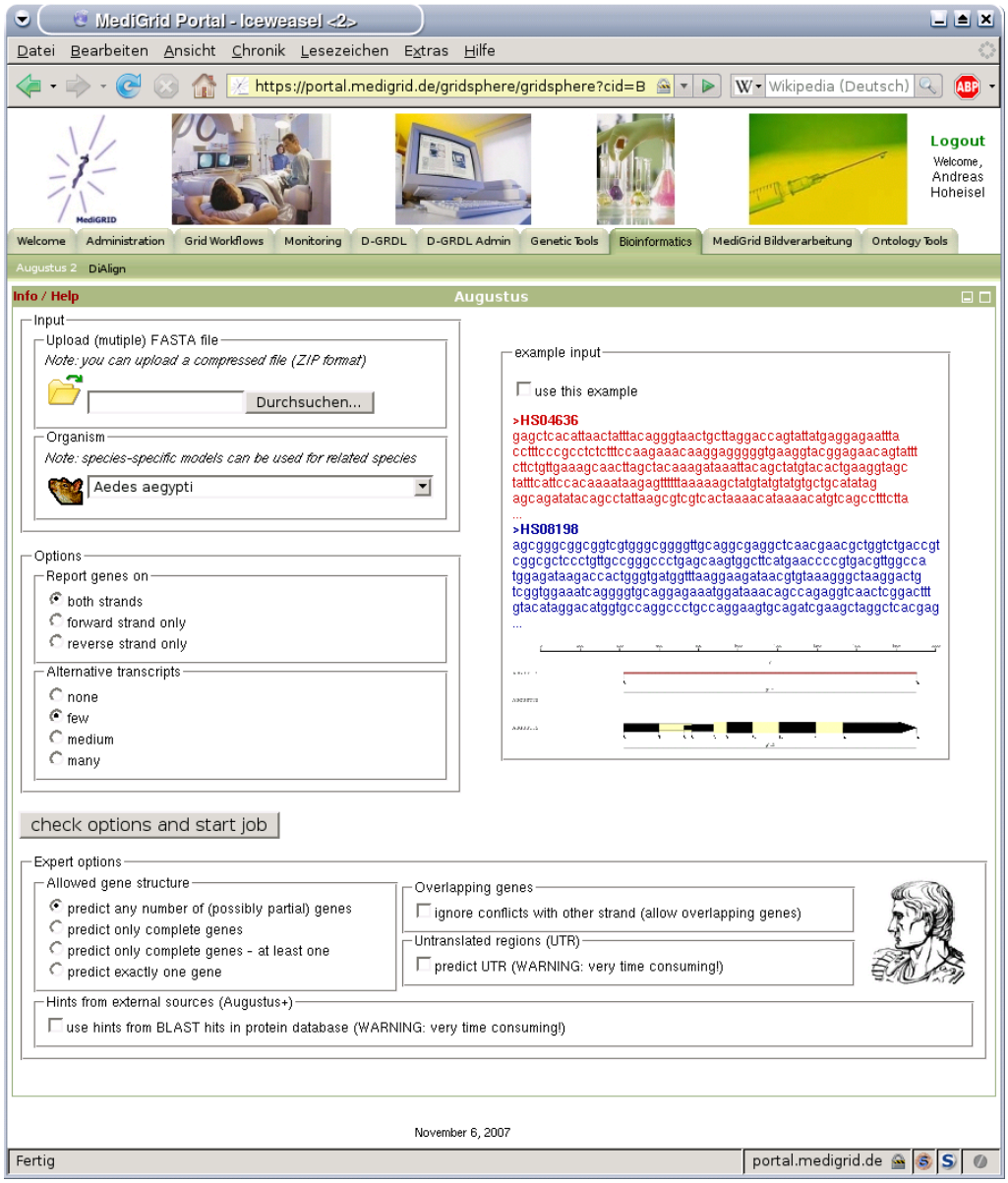


Figure 10. Screenshot of the „augustus“ portlet, integrated in the MediGRID portal. This portlet hides the technical details regarding the Grid technology and focuses on domain-specific knowledge.

Appendix A: Contact Persons

1. Portal Administrator:
   Julian Bart (julian.bart@iao.fraunhofer.de)
2. Resource Administrators: refer to “contacts” in Appendix B
3. Developers: Responsible for application development and deployment:

application-dir	responsible	email	uid[1]
augustus	Thomas Lingner	thomas@gobics.de	dgmd0015
AGRIPPA	Thomas Lingner	thomas@gobics.de	dgmd0015
generic-executor	Andreas Hoheisel	andreas.hoheisel@first.fraunhofer.de	dgmd0007
makeload	Andreas Hoheisel	andreas.hoheisel@first.fraunhofer.de	dgmd0007
medigridbvapp	Dagmar Krefting	dagmar.krefting@charite.de	dgmd0018
resourceupdater	Andreas Hoheisel	andreas.hoheisel@first.fraunhofer.de	dgmd0007
utils	Andreas Hoheisel	andreas.hoheisel@first.fraunhofer.de	dgmd0007

4. eXist XML DB Administrator:
   Julian Bart (julian.bart@iao.fraunhofer.de)
5. GWES Administrator:
   Andreas Hoheisel (andreas.hoheisel@first.fraunhofer.de)

[1] User name on the D-Grid-Resource regarding VORMS, as specified in the gridmapfile. The user name may vary on different Grid nodes.

Appendix B: MediGRID Resources

The following hardware resources are currently available for MediGRID workflow applications:

hostname	gsissh	g+w[2]	arch	CA[3]	contact
iwrgt4.fzk.de	✓	✓	x86_64 GNU/Linux PBS	GridKA-CA	Olaf.Schneider@iwr.fzk.de
medigrid-srv.gwdg.de	✓	×	x86_64 GNU/Linux PBS	DFN-PKI	dgrid-admin@gwdg.de
mardschana.zib.de	✓	✓	x86_64 GNU/Linux PBS	DFN-PKI	steinke@zib.de kathrin.peter@zib.de
romeo.urz.tu-dresden.de	✓	✓	ia64 GNU/Linux PBS	GridKA-CA	samatha.kottha@tu-dresden.de
othello.zih.tu-dresden.de	✓	✓	x86_64 GNU/Linux PBS	GridKA-CA	samatha.kottha@tu-dresden.de
hector.zih.tu-dresden.de	✓	✓	x86_64 GNU/Linux PBS	GridKA-CA	samatha.kottha@tu-dresden.de

For up to date information about hardware resources available for MediGRID, please refer to:

- List of D-Grid resources (only with D-Grid certificate installed in your browser): https://dispatch.fz-juelich.de:8814/D-Grid-Resource-List
- Web-MDS information about D-Grid resources: http://webmds.lrz-muenchen.de:8080/webmds/xslfiles/csm/
- List of MediGRID resources in SnipSnap: http://dgrid.first.fraunhofer.de/snips/medigrid/space/Middleware+%26+Ressourcenfusion/Compute+%26+Storage+Ressourcen/MediGRID+Ressourcen
- List of "Kern-D-Grid" resources in SnipSnap: http://dgrid.first.fraunhofer.de/snips/medigrid/space/Middleware+%26+Ressourcenfusion/Compute+%26+Storage+Ressourcen/Kern-Grid+Ressourcen

[2] Write permissions for group „medigrid" in /opt/medigrid

[3] Certificate Authority of the host certificate evaluated using the command "`grid-cert-info -file /etc/grid-security/hostcert.pem`"