TU Clausthal

Clausthal University of Technology

Stepwise Enabling of AUGUSTUS for MediGRID

Dietmar Sommerfeld[1], Thomas Lingner[2] and Harald Richter[3]

IfI Technical Report Series **IfI-07-12**

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Wojciech Jamroga
Contact: wjamroga@in.tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Stepwise Enabling of AUGUSTUS for MediGRID

Dietmar Sommerfeld[1], Thomas Lingner[2] and
Harald Richter[3]

[1] Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen
Am Fassberg, 37077 Göttingen, Germany
dsommer@gwdg.de
[2] Department of Bioinformatics, University of Göttingen
Goldschmidtstr. 1, 37077 Göttingen, Germany
thomas@gobics.de
[3] Department of Informatics, Clausthal University of Technology
Arnold-Sommerfeld-Str. 1, 38678 Clausthal-Zellerfeld, Germany
harald.richter@tu-clausthal.de

Abstract

In the past years researchers from many domains have discovered Grid technology as it opens up new possibilities to solve more complex problems than with traditional cluster computing. In this report we examine the process to enable the life science application AUGUSTUS for execution in MediGRID. The gridification process starts with providing the security requirements and running the application manually using Grid middleware. Afterwards, the application is described as workflow of subsequent program executions, which is automatically distributed on appropriate Grid resources by a workflow engine. Finally, we show how a graphical user interface for end users is created by means of a portal framework.

1 Introduction

Grid computing is the modern key technology to solve large-scale computational problems by using distributed heterogeneous resources. It is a special kind of distributed computing with a focus on the use of computing and data resources across existing administrative domains. Grids provide a new quality of service in resource sharing and problem solving. They allow the utilization of resources at a large number of distributed sites belonging to different organizations. Within a Grid, all members form a new administrative domain called Virtual Organization (VO).

If an application needs more resources than a single computing center can provide the application needs to be gridified, i.e. distributed to several

sites. Gridification is the process of enabling an application for execution on a Grid.

Gridification not only allows to execute an application that is too large for one site. It additionally reduces the execution time of smaller jobs that would fit into the site by acquiring resources from outside that were not accessible before. Usually resources in computing centers are not fully utilized. Such unused resources can be made available to the Grid. Additionally, computational loads are never constant, and with Grid computing an overloaded computing center can migrate jobs to sites with less load. The latter two facts increase the efficiency of a computing center. Finally, if a machine in a computing center is unavailable job migration enhances reliability.

The number of resources made available via Grid technology will increase because many public research institutes will not have the means to set up large local clusters anymore. Instead, computational projects will be funded that buy computing power from a Grid.

The work described in this report was carried out within the German MediGRID project. MediGRID is part of the German e-Science initiative D-Grid. The aim of the project is to provide a community Grid for researchers in the fields of medicine, biomedical informatics, and life sciences. Four types of pilot applications were selected for the first phase of MediGRID: bioinformatics, image processing, biomedical ontology, and clinical research applications. Researchers in the MediGRID community usually do not have a computer science background, therefore the usability of the Grid-solution is very important.

Distributed resources used for scientific computing are very heterogeneous. This refers to their hardware architecture as well as their software basis including operating system, resource management system and database system. To be able to handle the heterogeneity, a Grid middleware is employed which provides a common software basis on which the development of distributed applications can be settled. Middleware is a software layer between the operating system and the applications, and is present on all machines taking part in the Grid. The Grid middleware provides the basic services to use the resources and enables a uniform access to the underlying software architecture. Furthermore, it takes care of establishing a secure network environment and supports the construction of user-friendly interfaces like Grid portals.

The MediGRID project uses the Globus Toolkit 4 (GT4) [Foster, 2005] as its Grid middleware. Globus Toolkit is one of the most popular Grid middlewares and the basis of many Grid projects all around the world. In version 4, the architecture of the toolkit was completely changed and it is now implemented with stateful web services as specified by the Web Services Resource Framework (WSRF) [Foster et al., 2005]. The services implemented by GT4 are compliant to the Open Grid Services Architecture [Foster et al., 2002] standardization effort. OGSA services are in particular responsible for security re-

quirements, execution and data management, monitoring and resource discovery.

On top of the basic GT4 middleware, MediGRID employs further specialized middleware services. The most important one is an advanced workflow system for orchestrating the distributed execution of applications on Grid resources. An application workflow consists of several subsequent program executions and intermediate data transfers. Workflows allow the modeling of sequences of programs and the dependencies between them. We will show how application workflows are created and how they are executed on the Grid by the workflow system. Additionally, in MediGRID a portal framework is employed to provide an appropriate graphical user interface.

This report describes the gridification of a life science application. The gridification process involves the user interface, the workflow system and GT4 as required components. It does not require changes to the application source code. We examine the process with AUGUSTUS [Stanke et al., 2006a] as an example from the field of bioinformatics.

There are many life science applications from different research fields which have the same software architecture properties as AUGUSTUS. For example, analysis of time series like electroencephalograms, processing of image data as in gene-expression micro-array experiments and simulation of complex biological or chemical systems can be considered as similar gridification candidates.

The report is organized as follows: in section 2, we describe an AUGUSTUS application run and how the computation is distributed on clusters without using Grid middleware. Section 3 explains the execution of applications on a Grid using the services of GT4. Section 4 introduces the workflow system and shows what descriptions are necessary to enable automatic application runs on Grid resources. Section 5 describes how graphical user interfaces for Grid applications can be created using portlets, and in Section 6, the report concludes with a summary.

2 AUGUSTUS

AUGUSTUS [Stanke et al., 2006a] is a DNA sequence-based gene-prediction program for eukaryotes, i.e. organisms with a cell nucleus. It is typical for a whole class of bioinformatic applications. Gene prediction is the first and most important step in analysis of newly sequenced organisms.

The program is data- but not function parallel, thus it can easily be distributed and executed in parallel. AUGUSTUS uses sophisticated statistical modeling and prediction algorithms which are computationally demanding. The program does not require user interaction and can be run unattended. Input and output data is accessed from a central data storage repository. An application run consists of several phases which are separated by i/o

operations that address files. Via these files, data exchange between phases is done. The computing phases are loosely coupled, i.e. they have little communication in relation to the computation.

AUGUSTUS can be used as an ab initio program, which means that the prediction is based solely on the input DNA sequence. As a second possibility, the software may also incorporate hints on the gene structure coming from extrinsic sources such as BLAST [Altschul et al., 1990] and DIALIGN [Brudno et al., 2003] search results [Stanke et al., 2006b]. AUGUSTUS is used in several genome sequencing projects worldwide [Ghedin et al., 2007][Nene et al., 2007].

In case of ab initio use of AUGUSTUS, the command line program has two mandatory arguments: the query file and the species. The query file contains the input DNA sequences in uncompressed (multiple) FASTA [Pearson, 1990] format. The species parameter denotes where the DNA sequences originate from, e.g. human, fruit fly, baker's yeast, etc. There are several further parameters which influence the way the genes are predicted. The output of AUGUSTUS is written to stdout which is the command line output. The output is compatible to the General Feature Format which can be visually interpreted by means of a genome browser [Stein et al., 2002]. A typical command line looks like:

```
augustus --species=human example.fa > outputfile
```

Running AUGUSTUS on a single local computer is relatively easy. Nevertheless, depending on the number and length of sequences and the parameter values, the gene prediction can be computationally very intensive. For example, the application to the human genome with its 3 billion base pairs can take up to several weeks. A simple way of speeding up the computation without modifying the program source code is to parallelize the computation. This can be done by splitting the input file into smaller files which contain fewer sequences, and to run the program on these input files separately. Long sequences can be split into several smaller sequences if a suitable overlap according to maximum gene size is used. In order to interpret them, the results can be transformed into gene maps in graphical representation via gff2ps [Abril and Guigo, 2000].

Data parallelization provides the possibility to distribute the computation across several machines, for example the nodes of a local cluster. Typically, these nodes have a common file system, so the handling of input and output data is easy. Still a greater effort is necessary for handling the batch system and for checking if all submitted jobs are executed properly.

The organizational effort increases if local resources are not sufficient for the demands of the computation. In this case, further suitable resource providers have to be found to support the computation. If this succeeds, their resources are used to support the processing. For the AUGUSTUS application, this means splitting the input data and distributing them to multiple sites where

the program shall run. It is required to have access to and knowledge about the potentially different resource management systems if Grid computing is not used. Afterwards, the output files have to be transferred back to the user for result analysis.

Doing all these steps manually is tedious and time-consuming. The benefit of the parallelization is the overall decrease in turnaround time, allowing to tackle larger problem sizes which can not be accomplished with limited local resources. Some of the organizational problems, such as the handling of user accounts on multiple sites, can be solved by the middleware layer of the Grid.

3 Executing AUGUSTUS on the Grid

The first step before we can execute AUGUSTUS on the Grid is to provide for authentication and authorization. The Grid security infrastructure (GSI) of GT4 is based on a public key infrastructure. This means, every user and every server in the Grid is authenticated by a pair of public and private key. The public keys are signed by trusted certificate authorities (CA) and then become X.509 [Tremblett, 1999] identity certificates. Thus, the first step if one wants to use Grid resources, is to get a user certificate from a CA that is recognized by the Grid's security policy. Afterwards one has to become a member of the Grid's Virtual Organization (VO) and thus a member of the Grid. There are several solutions for VO membership registration. MediGRID uses the VOMRS [Demchenko et al., 2006] solution. VO membership is necessary to get an authorization for the servers of the Grid. The authorization is achieved with so-called grid-mapfiles that are present on every machine. They contain one line for every Grid user which states the distinguished name of the user's certificate and an UNIX account:[1]

```
"/C=DE/O=GridGermany/OU=Gesellschaft fuer wissenschaftliche Datenverar\
beitung mbH/CN=Dietmar Sommerfeld" dgmd0010
```

The pair of user certificate and private key is called Grid credential. To prevent the private key from leaving its secure environment on the User's computer, it is used to sign the public key of a new pair of short-lived proxy credentials. Afterwards, these proxy credentials are used to authenticate the user in the Grid. When a user accesses a server with his proxy credentials, first, the authenticity of the proxy certificate is checked, and afterwards the server looks for the distinguished name in the grid-mapfile and then maps the user onto the respective local UNIX account. Thereafter, all user actions are performed under that account and with its specific permissions. Besides

[1] Note that we use the backslash character \ to denote a line break because of limitations in text width.

authorization and authentication, the GSI provides secure communication via TLS [Dierks and Allen, 1999] and enables single sign-on, which means that a user only has to log-on once at one machine with his proxy credentials. Afterwards, he can use any service on any server in the Grid during the validity period of his proxy credentials (usually twelve hours).

The next step is the resource discovery to find a suitable machine for the execution of the application. For this purpose, each GT4 instance on any host in the Grid provides a Monitoring and Discovery Service (MDS). This service collects basic resource information about the host such as the services provided, processor speed, available memory and operating system. Each local MDS is typically configured to send an upstream of its information to another server which aggregates the data of several machines. This way, a hierarchical structure is formed with local MDS instances at the bottom and a central MDS at the top. In MediGRID, the central server has the WebMDS front-end installed which can be accessed with a web browser. For this, WebMDS uses XSLT transformations to convert the XML data of MDS into HTML pages. Additionally, MDS can be accessed using the wsrf-query command-line client. For example, the following command queries the MDS on the local machine:

```
wsrf-query -s https://localhost:8443/wsrf/services/DefaultIndexService
```

In general, the specific application is not installed initially on the grid machines and therefore has to be deployed first. There are several approaches for application deployment in the Grid. First of all, the application developers need interactive access to every kind of machine the application shall be executed on. This is necessary for the compilation and for testing in the respective runtime environment. In MediGRID, it is planned to have so-called interactive nodes which have the same software installation as the actual worker nodes. The standard deployment method is to have a common software stack on all machines in the Grid. In this case, the application programs are pre-installed by the system administrators. The other possibility is to do a dynamic deployment at runtime by transferring already compiled executables from a software repository to the execution hosts. Finally, one could even imagine to copy the source code and to compile the application just in time before execution. Dynamic approaches promise a greater flexibility in adaptation and a quicker employment of new resources. The disadvantage is the organizational effort and the time required for setting up the applications at every target system. Due to the heterogeneous nature of the Grid, dynamic deployment is also error prone. That is why we will suppose that the AUGUSTUS application is already installed, thus we can conclude resource discovery by choosing one server from the list of available machines.

Now the input data has to be transferred to the execution host. For data transfers, the Globus Toolkit provides GridFTP which works similar to the normal FTP and uses GSI for data encryption and credential-based access to

the machines. The command line client is called globus-url-copy. An additional functionality is the capability of third party transfers. These are file transfers between two machines initiated by a third host. GridFTP is a pre web-service component and already existed in GT2. Its web service counterpart is called Reliable File Transfer (RFT). RFT requires a database such as Postgres where it stores the status of the transmissions for an error recovery if a transfer fails.

Once all preparations are done, the job can be submitted. The execution management service of GT4 is called WS-GRAM (Web Service Grid Resource Allocation and Management). WS-GRAM offers a variety of options. A standard command line to execute AUGUSTUS is:

```
globusrun-ws -submit -F https://localhost:8443/wsrf/services/\
ManagedJobFactoryService -s -so outputfile -c /augustus_path/augustus \
--species=human example.fa
```

This is equivalent to the example of section 2 and executes the command on the local machine using the GT4 middleware for job submission instead of the UNIX shell. The working directory is the home directory of the Grid user, thus the input and output files are located there. The default job manager (called factory type) is “Fork”, which means a new UNIX process is created for the job. Other job managers of WS-GRAM enable job submission to PBS, LSF or LoadLeveler resource management systems [Czajkowski et al., 1998].

For more complex jobs, GT offers a XML-based job description language called Resource Specification Language (RSL). Its most important capability is to include file transfers into the job description. The transfers before and after job execution are called stage-in and stage-out. WS-GRAM relies on RFT to do these transfers. Further RSL possibilities include definition of the working directory, redirection of the console output and subsequent deletion of files. The job description in Fig. 1 executes AUGUSTUS with the same arguments as before, but also includes the necessary file transfers. It should be noted, that all file locations are specified from the perspective of the execution host, not of the submission host. The job is submitted with the following command line:

```
globusrun-ws -submit -F https://executionhost:8443/wsrf/services/\
ManagedJobFactoryService -f rsl-job-description.xml
```

As we can see, the Grid solves many of the problems we encountered with cluster computing. A Grid user only needs one certificate to access all resources of the VO in the same way. WS-GRAM is a single execution management service for the diverse batch systems on all sites of the VO. It makes data transfers less laborious by integrating them into the job description.

On the other hand, resource discovery and selection still have to be done manually with the Globus services. These are difficult tasks if there are many

```
<job>
  <executable>/opt/medigrid/augustus/augustus</executable>
  <directory>${GLOBUS_USER_HOME}</directory>
  <argument>--species=human</argument>
  <argument>example.fa</argument>
  <stdout>${GLOBUS_USER_HOME}/outputfile</stdout>
  <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
  <fileStageIn>
    <transfer>
      <sourceUrl>gsiftp://submissionhost/tmp/example.fa</sourceUrl>
      <destinationUrl>file:///${GLOBUS_USER_HOME}/example.fa
      </destinationUrl>
    </transfer>
  </fileStageIn>
  <fileStageOut>
    <transfer>
      <sourceUrl>file:///${GLOBUS_USER_HOME}/outputfile</sourceUrl>
      <destinationUrl>gsiftp://submissionhost/tmp/outputfile
      </destinationUrl>
    </transfer>
  </fileStageOut>
  <fileCleanUp>
    <deletion>
      <file>file:///${GLOBUS_USER_HOME}/example.fa</file>
      <file>file:///${GLOBUS_USER_HOME}/outputfile</file>
    </deletion>
  </fileCleanUp>
</job>
```

Figure 1: Example RSL job description document (see text).

resource providers, and if a load balancing between the sites is desired. Furthermore, the above example only covers a single execution of AUGUSTUS. In case of splitting data for parallel processing, multiple instances of the application have to be run. Implicit data transfers are not performed automatically, and job monitoring remains a tedious task, as there is no user-friendly interface to check the status of submitted jobs. All these issues have a greater significance, the more complex the job scenario is. If our computation consists e.g. of a workflow of parallel branches with subsequent program executions and intermediate data transfers, GT services are not sufficient anymore and we need a workflow management for high-level application steering.

4 Running AUGUSTUS with the workflow system

In MediGRID, the execution of complex compound applications is fully automated by the Grid Workflow Execution Service (GWES) [Hoheisel, 2005].

This is a flexible workflow orchestration infrastructure which is also used in several other European Grid projects, e.g. K-Wf Grid, Core Grid, and InstantGrid.

The control and data flow between the application components is modeled as a graph structure based on the Petri net formalism [Hoheisel and Alt, 2006]. Petri nets consist of places (symbolized by circles) and transitions (symbolized by squares). Places and transitions are connected by directed edges and always alternate with each other. In the workflow, transitions stand for program executions while places represent input and output data. Places are marked with so-called tokens if the data is available and transitions can only be activated (also called fired) when all input places have a token. When the transition is started, one token from every input place is taken, and when the transition is completed one token is given to every output place.

The first step to run AUGUSTUS with the GWES is to set up graphically a workflow for the application by creating its Petri net graph. It is recommended to draw the graph as the first step. The graph of a workflow with two parallel AUGUSTUS invocations is displayed in Fig. 2.

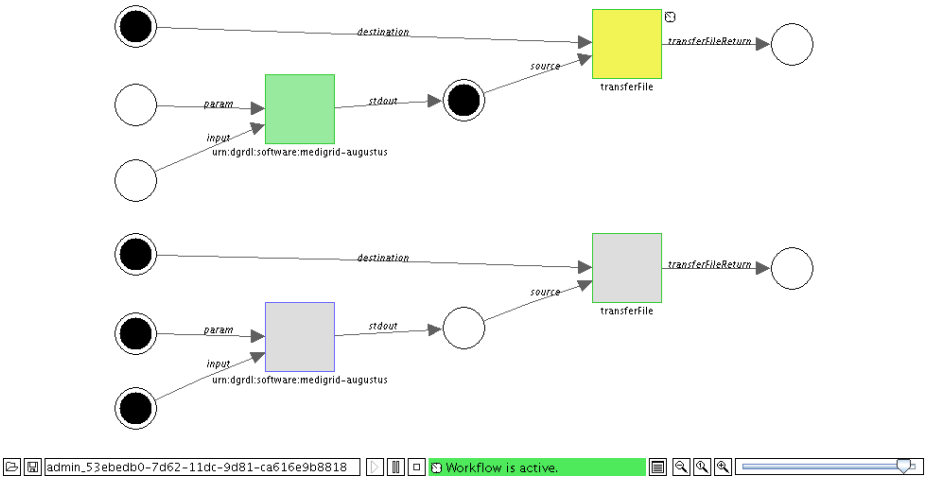

Figure 2: AUGUSTUS Petri net graph displayed in GWUI applet.

The second step is to convert the Petri net graph manually into the XML-based Grid Workflow Description Language (GWDL) [Alt et al., 2006]. The GWDL description is equivalent to the graphical representation. It is necessary for interpretation by the workflow engine. The description consists of a set of XML tags which specify all the data places and transitions. The edges between places and transitions are specified within the transition tags. A shortened description of the AUGUSTUS workflow is given in Fig. 3. The

complete description can be found in appendix A.1.

The inputPlace and outputPlace tags connect the transition with the specified places. Every edge is named with a string called edgeExpression. The operation tag within the transition clause denotes which program will be executed by the transition. In order to set up the workflow, the user has to specify an abstract program class only. When the GWES is invoked, it initializes the workflow and does a resource matching. This means, it searches for all available instances of that program class. For this, MediGRID employs a resource XML-database that contains information about the existing program instances and the software installed on each machine. Both types of descriptions are specified by means of the XML-based D-Grid Resource Description Language (D-GRDL) [Wolf, 2007].

The software resource description is necessary to find out which software class a program instance belongs to. It also contains the path to the executable file. During resource matching, GWES first queries the resource database and retrieves a list of all program instances matching the abstract program class. Thus, the next step in gridifying is to provide for the software resource descriptions of all different program instances. For servers with an identical software version installed at the same path the same D-GRDL description applies. An example software resource description for AUGUSTUS is given in Fig. 4.

The second step in resource matching is to get a list of all machines where one of the instances is installed. This requires the hardware description which lists all program instances provided by the machine. Furthermore, it also contains static machine information and utilization data. The machine descriptions are created and updated automatically by a resource monitoring daemon. An example hardware resource description is given in A.2.

Based on the current utilization of the machines, GWES automatically selects Grid nodes for the execution of the jobs in the workflow. After the resource selection is completed by GWES, the programClassExecution tag contains a sub-entry such as:

```
<pe:programExecution hardware="hardware:medigrid-srv.gwdg.de_PBS"
software="software:medigrid-augustus-2-0" quality="0.75"
selected="true"/>
```

Now the Petri net description can be interpreted by GWES. GWES provides automatic file staging, i.e. it organizes all necessary data transfers so that data is available at each hardware resource on which a job is scheduled and ready to execute. For the file transfers, GWES uses RFT from the underlying GT4 middleware.

To make sure that GWES can execute all kinds of command-line programs, a common parameter syntax is necessary. Therefore, all executables are encapsulated with wrapper scripts to pass arguments such as input and output

```
<?xml version="1.0" encoding="UTF-8"?>
<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://www.gridworkflow.org/kwfgrid/src/xsd/gworkflowdl_1_0.xsd">
  <description>AUGUSTUS MediGRID Workflow</description>
  <property name="resource.repository.collection">/db/dgrdl</property>
  <place ID="fastainput0">
    <token><data>
      <file xsi:type="xsd:string">gsiftp://139.18.18.60//tmp/augustus/\
        HS04636</file>
    </data></token>
  </place>
  <place ID="parameter0">
    <token><data>
      <param xsi:type="xsd:string">--species=human</param>
    </data></token>
  </place>
  <place ID="outputdestination0">
    <token><data>
      <param xsi:type="xsd:string">gsiftp://139.18.18.60//tmp/\
        HS04636.gff</param>
    </data></token>
  </place>
  <place ID="augustusoutput0"/>
  <place ID="result0"/>
  <transition ID="augustus0">
    <description>Augustus worker 0</description>
    <inputPlace placeID="parameter0" edgeExpression="param"/>
    <inputPlace placeID="fastainput0" edgeExpression="input"/>
    <outputPlace placeID="augustusoutput0" edgeExpression="stdout"/>
    <operation>
      <pe:programClassExecution
        xmlns:pe="http://www.gridworkflow.org/gworkflowdl/\
        programclassexecution"
        softwareClass="urn:dgrdl:software:medigrid-augustus">
      </pe:programClassExecution>
    </operation>
  </transition>
  <transition ID="transfer0">
    <description>file transfer 0</description>
    <inputPlace placeID="augustusoutput0" edgeExpression="source"/>
    <inputPlace placeID="outputdestination0" edgeExpression="destination"/>
    <outputPlace placeID="result0" edgeExpression="transferFileReturn"/>
    <operation>
      <ws:WSClassOperation xmlns:ws="http://www.gridworkflow.org/\
        gworkflowdl/wsclassoperation">
        <ws:WSOperation
          wsdl="http://portal.medigrid.izbi.uni-leipzig.de:9081//gwes/\
          services/FileTransfer?wsdl"
          operationName="transferFile" selected="true"/>
      </ws:WSClassOperation>
    </operation>
  </transition>
</workflow>
```

Figure 3: D-GRDL workflow description for AUGUSTUS with a single input sequence.

```
<?xml version="1.0" encoding="UTF-8"?>
<resource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://portal2.medigrid.izbi.uni-\
  leipzig.de:9081/gwes/xsd/instantgrid-d-grdl.xsd"
  uri="software:medigrid-augustus-2-0">
    <ofClass uri="urn:dgrdl:software:medigrid-augustus"/>
    <name>Augustus</name>
    <description>AUGUSTUS gene prediction program</description>
    <simpleProperty ident="executable" type="string" unit="">
      /opt/medigrid/augustus/augustus.sh
    </simpleProperty>
    <simpleProperty ident="score">101.0</simpleProperty>
</resource>
```

Figure 4: Software resource description for AUGUSTUS.

files and program parameters in always the same way. A listing for an example wrapper script is given in Fig. 5. The parameters of the wrapper script comply with the edgeExpressions in the workflow. Command line output can be redirected using stdout and stderr. GWES automatically creates the WS-GRAM job descriptions to submit Globus jobs that in turn execute the wrapper scripts. If a job is not finished successfully, the fault management of the workflow system automatically selects another suitable resource and submits the job again.

The GWES software package also includes a graphical interface called Grid Workflow User Interface (GWUI) which allows the user to monitor and steer the processing of the workflow (see Fig. 2). GWUI is implemented as a Java applet and accessed via a web browser. It displays how the tokens move through the workflow graph as the transitions are executed. The bottom line shows the current workflow state and contains buttons to suspend, resume or stop the workflow. Furthermore, the current state can be stored in GWDL format. By clicking into the graph the properties of tokens, places and transitions can be inspected and manipulated. For transitions e.g. it is possible to select one of the available resources manually.

Most of the problems that exist with the basic Globus middleware are solved with the use of the workflow system. Once an application is gridified for GWES, the complete execution process is automated. This includes resource discovery and selection, implicit data transfers between job executions as well as the job submission. GWES provides a much higher quality of service than GT4 and delivers the scalability and performance for complex application workflows comprising of parallel computational branches. Additionally, GWUI offers an improved usability and better overview of the status during workflow execution.

However, some effort remains as workflow descriptions have to be created for every application run. Furthermore, users are required to have Grid

```
#!/bin/bash
#
# AUGUSTUS wrapper script

# program location
MEDIGRID_HOME = /opt/medigrid
AUGUSTUS=$MEDIGRID_HOME/augustus
AUGUSTUS_BIN=$AUGUSTUS/augustus

function usage {
  echo "### USAGE: $0 -param "parameters" -input inputfile"
}

PARAM=${2}
INPUT=$4

if [ ! -r "$AUGUSTUS_BIN" ]; then
  echo "### Error: executable 'augustus' not found in path"
  exit 1
fi

export AUGUSTUS_CONFIG_PATH=$AUGUSTUS/config
exec $AUGUSTUS_BIN $PARAM $INPUT
echo "### AUGUSTUS done ###"
```

Figure 5: AUGUSTUS wrapper script for execution on grid machines.

knowledge and direct access to a Grid machine. It is advisable to provide the functionality of the application within a graphical user interface (GUI) that can be accessed with a web browser. Such a GUI can be realized with a Grid-enabled web portal.

5 Creating a graphical user interface

MediGRID uses GridSphere [Novotny et al., 2004] which is a framework to establish Grid portals. GridSphere provides central access to Grid resources and services by a single world-wide accessible user interface. Inside a Grid portal, small applications can run which are called portlets [Diaz and Rodriguez, 2004]. In MediGRID, portlets provide access to services from Globus and MediGRID. Each application will have its own portlet as an easy-to-use graphical user interface (GUI).

Development of a GridSphere portlet starts with the creation of a GridSphere project which sets up a project directory and some basic portlet configuration files in XML format. After editing the configuration files, the portlet user interface and logic can be developed by means of Java Server Pages (JSP) and a portlet Java class, respectively. The AUGUSTUS portlet is based on

GridSphere's action provider model. The action provider model and the associated action portlet interface provide the possibility to separate the graphical layout from the portlet logic. The GUI consists of several JSPs which contain the layout description in HTML, and action components from GridSphere's user interface tag-library. These components, e.g. links, buttons and form fields, can be accessed within the Java class. An application usage scenario is implemented as a series of requests and responses. The basic JSP code of the job configuration page of the AUGUSTUS portlet is given in appendix A.4.

The AUGUSTUS portlet consists of three main stages: job configuration, workflow execution, and result presentation. Within the job configuration stage (see Fig. 6), the user can upload a (compressed) multiple FASTA sequence file and configure some basic AUGUSTUS prediction parameters. Users usually do not access the portlet from a Grid machine, thus the input file has to be transferred from the user's computer to a Grid machine. At present, the upload is done via HTTPS and the portal server is used for temporary storage. Another option would be to upload the input data to a Grid storage element with a suitable upload client. After submission by the user, the AUGUSTUS parameters are retrieved from the user interface components and stored in a parameter string.

The splitting of the input file according to the number of available single sequences is done on the portal server. On one hand, splitting is very fast in comparison to gene prediction. On the other hand, splitting on an arbitrary Grid machine would require splitting software on this machine and additional data transfers within the Grid. Furthermore, the input file would have to be parsed twice because the workflow has to be specified fully within the portlet before invocation. The single sequences are stored as files in a temporary directory on the portal server. Their file names can be used directly as data tokens for the input places of the Grid workflow.

In the workflow execution stage, the GWDL-compliant workflow document as shown in A.1 is automatically created using the StringTemplate Java library from StringTemplate.org. For the creation of the workflow document, a template for a single AUGUSTUS invocation is duplicated according to the number of sequences, and the variables within the template are replaced by real filenames and instances. An example template document for one AUGUSTUS invocation is given in A.3. The variables which have to be replaced are surrounded by dollar signs. In the end, the resulting workflow document is concatenated with the corresponding header. The JAVA code for workflow template extension according to the number of input sequences is given in A.5.

After the Grid workflow has been created, it can be initiated using the GWES Java API. The JAVA code for workflow initiation within the portlet is shown in A.6. Then, the GWES executes it according to the specified steps and returns either an execution success or an error code. The progress of

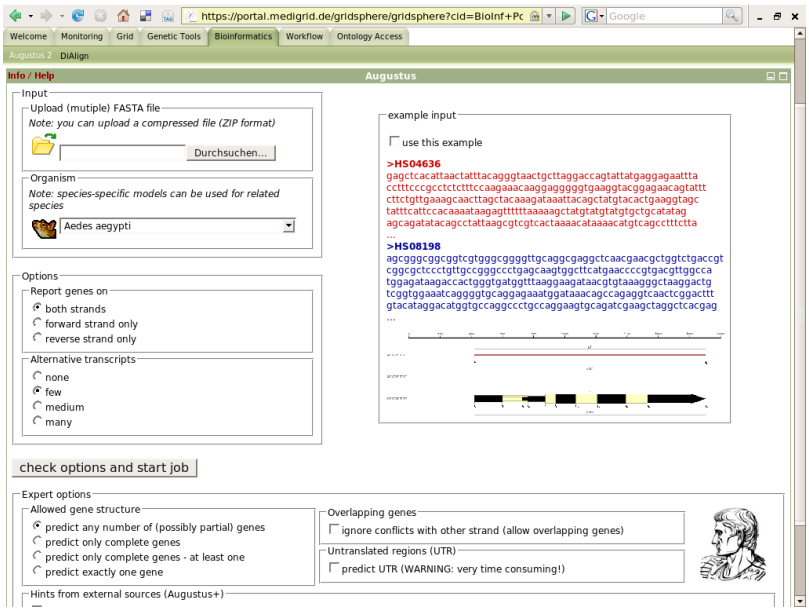

Figure 6: Screenshot of the job configuration stage in the AUGUSTUS portlet in the MediGRID portal at http://portal.medigrid.de.

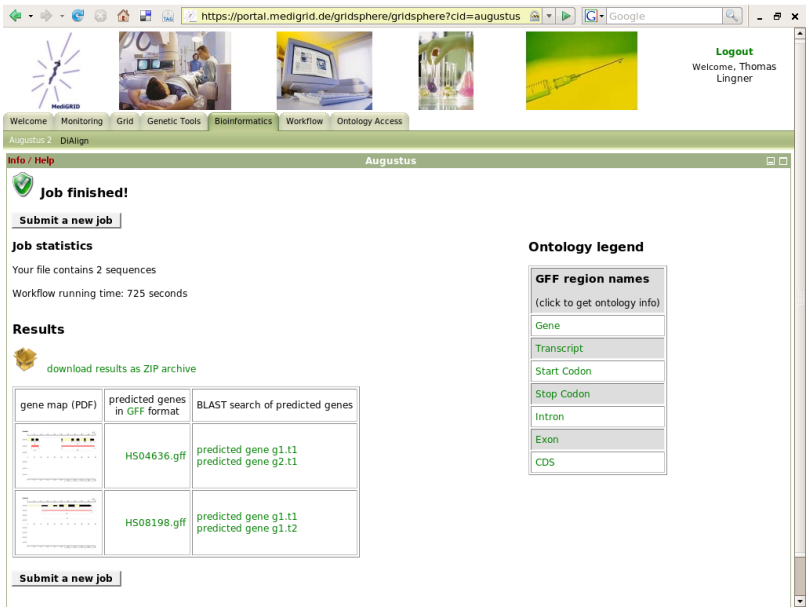

Figure 7: Screenshot of the result stage in the AUGUSTUS portlet.

the workflow execution can be monitored by means of the GWUI Java applet. With this applet, the graphical interface of the workflow engine can conveniently be integrated into the portlet to provide the user with status information and steering possibilities.

The last transition in the AUGUSTUS workflow transfers the results back to the portal server where they are automatically postprocessed after successful workflow completion. The postprocessing consists of a transformation of the prediction result of every sequence into a graphical representation (gene map) using gff2ps and ImageMagick. If the required software is installed on the Grid nodes, this process can also be integrated into the workflow. The prediction results are displayed on a final result page and can be downloaded in a compressed format via a link tag (see Fig. 7).

Portals such as in MediGRID provide the possibility to connect related applications so that the results can be further analyzed. Links to the related ontology access portlet of the MediGRID portal are integrated into the AUGUSTUS portlet via so-called actionlinks from GridSphere's user interface taglibrary. This allows the user to access information about specific sequence regions. Furthermore, predicted genes can automatically be searched for related database entries via links to the NCBI BLAST server using the BLAST URL API [NCBI, 2001].

6 Conclusion

In this report, we have shown the development process of enabling an application for the MediGRID project. The process is subdivided into steps and based on the employed middlewares such as Globus Toolkit and GWES workflow system, as well as the GridSphere portal framework. The described gridification process can be applied to cluster applications without changes to the source code. Still it should ideally be done by the application developers since it requires insight into the application's mode of operation. Additionally, support from the resource providers, i.e. computing centers is necessary. As with traditional parallelization and vectorization, computing centers have to assist in porting the application on their specific hardware architectures, and they must install the application and middleware software components. As a conclusion, computing centers are required to help with training on the new middleware layers imposed by the Grid since gridification is a much more complex task than parallelization.

The benefit of the efforts is that in the end the applications can be run by end users that only have to authenticate and authorize in their VO. Besides, they need no Grid knowledge and can profit from a wide range of computing resources that allow tackling larger problem sizes. For computing centers, Grid technology offers a third mainstay, next to traditional parallel and

vector computing. With Grids they can offer computing services to new customer groups.

The software components used in MediGRID are specific choices from a range of existing middleware solutions. They are accepted among the Grid community and have proven appropriate. Globus Toolkit consists of several components, out of which some are still in development or need to be extended for custom application scenarios. The Globus services employed in MediGRID belong to the core components and are reliable for production use. GWES and the GridSphere portal are operational, but still under development to add more features and improve the functionality. The future work in MediGRID will be focused on these systems and the further development of the application portlets.

7 Acknowledgment

We thank Mario Stanke for helpful comments on AUGUSTUS. This work is funded by the German Federal Ministry of Education and Research (BMBF) within the MediGRID project, grant numbers 01AK803A-H. The workflow system presented in section 4 is being developed by the Fraunhofer-Institute for Computer Architecture and Software Technology (FIRST) based on results of the EU IST project K-Wf Grid as well as the BMBF-funded projects InstantGrid and D-Grid Integration project (DGI).

A Supplementary listings

A.1 Complete AUGUSTUS workflow description

```
<?xml version="1.0" encoding="UTF-8"?>
<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://www.gridworkflow.org/kwfgrid/src/xsd/gworkflowdl_1_0.xsd"
  ID="tlingner_edc80c10-2318-11dc-9824-97d58f9419d1">
  <description>AUGUSTUS MediGRID Workflow</description>
  <property name="resource.repository.collection">/db/dgrdl</property>
  <place ID="fastainput0">
    <token><data>
      <file xsi:type="xsd:string">gsiftp://139.18.18.60//tmp/augustus/\
        HS04636</file>
    </data></token>
  </place>
  <place ID="parameter0">
    <token><data>
      <param xsi:type="xsd:string">--species=human</param>
    </data></token>
  </place>
  <place ID="outputdestination0">
    <token><data>
      <param xsi:type="xsd:string">gsiftp://139.18.18.60//tmp/\
        HS04636.gff</param>
    </data></token>
  </place>
  <place ID="augustusoutput0"/>
  <place ID="result0"/>
  <place ID="fastainput1">
    <token><data>
      <file xsi:type="xsd:string">gsiftp://139.18.18.60//tmp/augustus/\
        HS08198</file>
    </data></token>
  </place>
  <place ID="parameter1">
    <token><data>
      <param xsi:type="xsd:string">--species=human</param>
    </data></token>
  </place>
  <place ID="outputdestination1">
    <token><data>
      <param xsi:type="xsd:string">gsiftp://139.18.18.60//tmp/\
        HS08198.gff</param>
    </data></token>
  </place>
  <place ID="augustusoutput1"/>
  <place ID="result1"/>
  <transition ID="augustus0">
    <description>Augustus worker 0</description>
    <inputPlace placeID="parameter0" edgeExpression="param"/>
```

```
    <inputPlace placeID="fastainput0" edgeExpression="input"/>
    <outputPlace placeID="augustusoutput0" edgeExpression="stdout"/>
    <operation>
      <pe:programClassExecution
        xmlns:pe="http://www.gridworkflow.org/gworkflowdl/\
        programclassexecution"
        softwareClass="urn:dgrdl:software:medigrid-augustus">
      </pe:programClassExecution>
    </operation>
  </transition>
  <transition ID="transfer0">
    <description>file transfer 0</description>
    <inputPlace placeID="augustusoutput0" edgeExpression="source"/>
    <inputPlace placeID="outputdestination0" edgeExpression="destination"/>
    <outputPlace placeID="result0" edgeExpression="transferFileReturn"/>
    <operation>
      <ws:WSClassOperation xmlns:ws="http://www.gridworkflow.org/\
        gworkflowdl/wsclassoperation">
        <ws:WSOperation
          wsdl="http://portal.medigrid.izbi.uni-leipzig.de:9081//gwes/\
          services/FileTransfer?wsdl"
          operationName="transferFile" selected="true"/>
      </ws:WSClassOperation>
    </operation>
  </transition>
  <transition ID="augustus1">
    <description>Augustus worker 1</description>
    <inputPlace placeID="parameter1" edgeExpression="param"/>
    <inputPlace placeID="fastainput1" edgeExpression="input"/>
    <outputPlace placeID="augustusoutput1" edgeExpression="stdout"/>
    <operation>
      <pe:programClassExecution
        xmlns:pe="http://www.gridworkflow.org/gworkflowdl/\
        programclassexecution"
        softwareClass="urn:dgrdl:software:medigrid-augustus">
      </pe:programClassExecution>
    </operation>
  </transition>
  <transition ID="transfer1">
    <description>file transfer 1</description>
    <inputPlace placeID="augustusoutput1" edgeExpression="source"/>
    <inputPlace placeID="outputdestination1" edgeExpression="destination"/>
    <outputPlace placeID="result1" edgeExpression="transferFileReturn"/>
    <operation>
      <ws:WSClassOperation xmlns:ws="http://www.gridworkflow.org/\
        gworkflowdl/wsclassoperation">
        <ws:WSOperation
          wsdl="http://portal.medigrid.izbi.uni-leipzig.de:9081//gwes/\
          services/FileTransfer?wsdl"
          operationName="transferFile" selected="true"/>
      </ws:WSClassOperation>
    </operation>
  </transition>
</workflow>
```

A.2 Hardware resource description for the server medigrid-srv

```
<?xml version="1.0" encoding="UTF-8"?>
<resource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.gridworkflow.org/kwfgrid/\
  src/xsd/resource-d-grdl.xsd" uri="hardware:medigrid-srv.gwdg.de/PBS">
    <ofClass uri="urn:dgrdl:hardware"/>
    <name>medigrid-srv.gwdg.de/PBS</name>
    <description>Hardware resource medigrid-srv.gwdg.de with local\
      resource manager PBS and Globus Toolkit
    </description>
    <provides>
      <!-- tests and benchmarks -->
      <resourceRef uri="software:cat-medigrid"/>
      <!-- utils -->
      <resourceRef uri="software:medigrid-utils-chmod-all-read-0-1"/>
      <resourceRef uri="software:medigrid-utils-show-environment-0-1"/>
      <!-- augustus -->
      <resourceRef uri="software:medigrid-augustus-2-0"/>
    </provides>
    <simpleProperty ident="Info.LRMSType" type="string" unit=""
      validUntil="2007-10-25T15:01:43Z">PBS
    </simpleProperty>
    <simpleProperty ident="Info.GRAMVersion" type="string" unit=""
      validUntil="2007-10-25T15:01:43Z">4.0.3
    </simpleProperty>
    <simpleProperty ident="Info.ContactString" type="uri" unit=""
      validUntil="2007-10-25T15:01:43Z">https://medigrid-srv.gwdg.de:\
      8443/wsrf/services/ManagedJobFactoryService
    </simpleProperty>
    <simpleProperty ident="State.RunningJobs" type="int" unit=""
      validUntil="2007-10-24T15:59:40Z">5
    </simpleProperty>
    <simpleProperty ident="State.WaitingJobs" type="int" unit=""
      validUntil="2007-10-24T15:59:40Z">6</simpleProperty>
    <simpleProperty ident="State.TotalJobs" type="int" unit=""
      validUntil="2007-10-24T15:59:40Z">11</simpleProperty>
    <simpleProperty ident="gwes.gram.home.directory" type="string"
      unit="">/opt/medigrid/tmp
    </simpleProperty>
    <simpleProperty ident="score">1911</simpleProperty>
</resource>
```

A.3 StringTemplate workflow document for a single AUGUSTUS transition

```
<place ID="fastainput$processnumber$">
  <token><data>
    <file xsi:type="xsd:string">$inputfile$</file>
  </data></token>
</place>
<place ID="parameter$processnumber$">
  <token><data>
    <param xsi:type="xsd:string">$parameter$</param>
  </data></token>
</place>
<place ID="outputdestination$processnumber$">
  <token><data>
    <param xsi:type="xsd:string">$outputfile$</param>
  </data></token>
</place>
<place ID="augustusoutput$processnumber$"/>
<place ID="result$processnumber$"/>

<transition ID="augustus$processnumber$">
  <description>Augustus worker $processnumber$</description>
  <inputPlace
    placeID="parameter$processnumber$" edgeExpression="param"/>
  <inputPlace
    placeID="fastainput$processnumber$" edgeExpression="input"/>
  <outputPlace
    placeID="augustusoutput$processnumber$" edgeExpression="stdout"/>
  <operation>
    <pe:programClassExecution
       xmlns:pe="http://www.gridworkflow.org/gworkflowdl/\
       programclassexecution"
       softwareClass="urn:dgrdl:software:medigrid-augustus">
    </pe:programClassExecution>
  </operation>
</transition>

<transition ID="transfer$processnumber$">
  <description>file transfer $processnumber$</description>
  <inputPlace
    placeID="augustusoutput$processnumber$" edgeExpression="source"/>
  <inputPlace
    placeID="outputdestination$processnumber$" edgeExpression="destination"/>
  <outputPlace
    placeID="result$processnumber$" edgeExpression="transferFileReturn"/>
  <operation>
    <ws:WSClassOperation
      xmlns:ws="http://www.gridworkflow.org/gworkflowdl/wsclassoperation">
      <ws:WSOperation wsdl="$GWES_URL$/services/FileTransfer?wsdl"
        operationName="transferFile" selected="true"/>
    </ws:WSClassOperation>
  </operation>
</transition>
```

A.4 JSP job configuration page of the AUGUSTUS portlet

```
<%@ taglib uri="/portletUI" prefix="ui" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>

<%@page import="java.util.Set"%>
<portlet:defineObjects/>

<jsp:useBean id="species" type="Set" scope="request"/>
<% species = (Set) renderRequest.getAttribute("species"); %>

<form action="../gridsphere/gridsphere?cid=augustus&gs_action=startJob"
  enctype="multipart/form-data" method="POST">

<ui:group label="Input">

<ui:group label="Upload (mutiple) FASTA file">
  <i>Note: you can upload a compressed file (ZIP format)</i>
  <ui:table>
  <ui:tablerow>
    <ui:tablecell>
    <img src="../bioinfportlets/images/open-32x32.png" alt="">
    <input type="file" name="fileupload">
    </ui:tablecell>
    <ui:tablecell> </ui:tablecell>
  </ui:tablerow>
</ui:table>
</ui:group>

<ui:group label="Organism">
  <i>Note: species-specific models can be used for related species</i>
  <ui:table>
  <ui:tablerow>
    <ui:tablecell>
    <img src="../bioinfportlets/images/Mouse-icon.gif" alt="">
    </ui:tablecell>
    <ui:tablecell>
    <ui:listbox beanId="param_species">
    <% for (java.util.Iterator iter = species.iterator(); \
      iter.hasNext();) { %>
      <ui:listboxitem value="<%= (String) iter.next() %>">
      </ui:listboxitem>
    <% } %>
    </ui:listbox>
    </ui:tablecell>
  </ui:tablerow>
  </ui:table>
</ui:group>

</ui:group>

<p><h2><input type="submit" value="check options and start job"></h2><p>

</form>
```

A.5 JAVA code for workflow template extension

```
private String createWorkflow(String[] inputFiles, String[] outputFiles,
    String parameter) {

  StringTemplate WorkflowTemplate=augustusWorkflowTemplate;
  StringBuffer workflow = new StringBuffer();

  final String NL = System.getProperty("line.separator");
  final String HEADER =
    "<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:// server:8080/gwes/xsd/gworkflowdl_1_0.xsd\">";
  final String DESC =
    "<description>AUGUSTUS MediGRID Workflow</description>";
  final String PROP =
    "<property name=\"resource.repository.collection\">" +
    "/db/dgrdl</property>";

  workflow.append(HEADER + NL + DESC + NL + PROP);

  for (int i = 0; i < outputFiles.length; i++) {
    WorkflowTemplate.reset();
    WorkflowTemplate.setAttribute("processnumber", i);
    WorkflowTemplate.setAttribute("inputfile", "gsiftp://" + IPADDRESS
        + FS + inputFiles[i]);
    WorkflowTemplate.setAttribute("outputfile", "gsiftp://" + IPADDRESS
        + FS + outputFiles[i]);
    WorkflowTemplate.setAttribute("parameter", parameter);
    WorkflowTemplate.setAttribute("GWES_URL", GWES_URL);
    workflow.append(WorkflowTemplate.toString());
  }

  workflow.append(NL + "</workflow>");

  return workflow.toString();
}
```

A.6 JAVA code for workflow initiation and execution within the portlet

```
private void executeWorkflow(String gworkflowdl) throws IOException {

    AxisProperties.setProperty("axis.ClientConfigFile",
            PORTLET_PATH + FS +"gwes-client-config.wsdd");

    GWESClient client = new GWESClient(new URL(GWES_URL
            + "/services/GWES"), null);
    gwes = client.gwes;
    workflowID = gwes.initiate(gworkflowdl, user);
    gwes.start(workflowID);
}
```

References

[Abril and Guigo, 2000] Abril, J. F. and Guigo, R. (2000). gff2ps: visualizing genomic annotations. *Bioinformatics*, 16(8):743–744.

[Alt et al., 2006] Alt, M., Hoheisel, A., Pohl, H.-W., and Gorlatch, S. (2006). A grid workflow language using high-level petri nets. In et al., R. W., editor, *PPAM2005*, volume 3911 of *LNCS*, pages 715–722. Springer-Verlag Berlin Heidelberg.

[Altschul et al., 1990] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410.

[Brudno et al., 2003] Brudno, M., Chapman, M., Gottgens, B., Batzoglou, S., and Morgenstern, B. (2003). Fast and sensitive multiple alignment of large genomic sequences. *BMC Bioinformatics*, 4:66. Comparative Study.

[Czajkowski et al., 1998] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., and Tuecke, S. (1998). A resource management architecture for metacomputing systems. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, London, UK. Springer-Verlag.

[Demchenko et al., 2006] Demchenko, Y., de Laat, C., and Ciaschini, V. (2006). VO-based dynamic security associations in collaborative grid environment. In *CTS '06: Proceedings of the International Symposium on Collaborative Technologies and Systems*, pages 38–47, Washington, DC, USA. IEEE Computer Society.

[Diaz and Rodriguez, 2004] Diaz, O. and Rodriguez, J. J. (2004). Portlets as web components: an introduction. *Journal of Universal Computer Science*, 10(4):454–472.

[Dierks and Allen, 1999] Dierks, T. and Allen, C. (1999). The TLS protocol. Technical report, Certicom Network Working Group.

[Foster, 2005] Foster, I. (2005). Globus toolkit version 4: Software for service-oriented systems. In Jin, H., Reed, D. A., and Jiang, W., editors, *NPC*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer.

[Foster et al., 2005] Foster, I., Czajkowski, K., Ferguson, D. E., Frey, J., Graham, S., Maguire, T., Snelling, D., and Tuecke, S. (2005). Modeling and managing state in distributed systems: the role of OGSI and WSRF. *Proceedings of the IEEE*, 93(3):604–612.

[Foster et al., 2002] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The physiology of the grid: An open grid services architecture for distributed systems integration.

[Ghedin et al., 2007] Ghedin, E., Wang, S., Spiro, D., Caler, E., Zhao, Q., Crabtree, J., Allen, J. E., Delcher, A. L., Guiliano, D. B., Miranda-Saavedra, D., Angiuoli, S. V., Creasy, T., Amedeo, P., Haas, B., El-Sayed, N. M., Wortman, J. R., Feldblyum, T., Tallon, L., Schatz, M., Shumway, M., Koo, H., Salzberg, S. L., Schobel, S., Pertea, M., Pop, M., White, O., Barton, G. J., Carlow, C. K. S., Crawford, M. J., Daub, J., Dimmic, M. W., Estes, C. F., Foster, J. M., Ganatra, M., Gregory, W. F., Johnson, N. M., Jin, J., Komuniecki, R., Korf, I., Kumar, S., Laney, S., Li, B.-W., Li, W., Lindblom, T. H., Lustigman, S., Ma, D., Maina, C. V., Martin, D. M. A., McCarter, J. P., McReynolds, L., Mitreva, M., Nutman, T. B., Parkinson, J., Peregrin-Alvarez, J. M., Poole, C., Ren, Q., Saunders, L., Sluder, A. E., Smith, K., Stanke, M., Unnasch, T. R., Ware, J., Wei, A. D., Weil, G., Williams, D. J., Zhang, Y., Williams, S. A., Fraser-Liggett, C., Slatko, B., Blaxter, M. L., and Scott, A. L. (2007). Draft genome of the filarial nematode parasite Brugia malayi. *Science*, 317(5845):1756–1760. Comparative Study.

[Hoheisel, 2005] Hoheisel, A. (2005). Grid workflow execution service - user manual. Technical report, Fraunhofer FIRST/K-Wf Grid Project. http://www.gridworkflow.org/kwfgrid/gwes/docs/KWF-WP2-D2-FIRST-GWESUserManual.pdf.

[Hoheisel and Alt, 2006] Hoheisel, A. and Alt, M. (2006). Petri nets. In Taylor, I. J., Gannon, D., Deelman, E., and Shields, M. S., editors, *Workflows for eScience*. Springer-Verlag.

[NCBI, 2001] NCBI (2001). Qblast's URL API. User's guide. http://www.-ncbi.nlm.nih.gov/BLAST/Doc/urlapi.html.

[Nene et al., 2007] Nene, V., Wortman, J. R., Lawson, D., Haas, B., Kodira, C., Tu, Z. J., Loftus, B., Xi, Z., Megy, K., Grabherr, M., Ren, Q., Zdobnov, E. M., Lobo, N. F., Campbell, K. S., Brown, S. E., Bonaldo, M. F., Zhu, J., Sinkins, S. P., Hogenkamp, D. G., Amedeo, P., Arensburger, P., Atkinson, P. W., Bidwell, S., Biedler, J., Birney, E., Bruggner, R. V., Costas, J., Coy, M. R., Crabtree, J., Crawford, M., Debruyn, B., Decaprio, D., Eiglmeier, K., Eisenstadt, E., El-Dorry, H., Gelbart, W. M., Gomes, S. L., Hammond, M., Hannick, L. I., Hogan, J. R., Holmes, M. H., Jaffe, D., Johnston, J. S., Kennedy, R. C., Koo, H., Kravitz, S., Kriventseva, E. V., Kulp, D., Labutti, K., Lee, E., Li, S., Lovin, D. D., Mao, C., Mauceli, E., Menck, C. F. M., Miller, J. R., Montgomery, P., Mori, A., Nascimento, A. L., Naveira, H. F., Nusbaum, C., O'leary, S., Orvis, J., Pertea, M., Quesneville, H., Reidenbach, K. R., Rogers, Y.-H., Roth, C. W., Schneider, J. R., Schatz, M., Shumway, M., Stanke, M., Stinson, E. O., Tubio, J. M. C., Vanzee, J. P., Verjovski-Almeida,

S., Werner, D., White, O., Wyder, S., Zeng, Q., Zhao, Q., Zhao, Y., Hill, C. A., Raikhel, A. S., Soares, M. B., Knudson, D. L., Lee, N. H., Galagan, J., Salzberg, S. L., Paulsen, I. T., Dimopoulos, G., Collins, F. H., Birren, B., Fraser-Liggett, C. M., and Severson, D. W. (2007). Genome sequence of Aedes aegypti, a major arbovirus vector. *Science*, 316(5832):1718–1723.

[Novotny et al., 2004] Novotny, J., Russell, M., and Wehrens, O. (2004). Gridsphere: a portal framework for building collaborations. *Concurrency And Computation-Practice & Experience*, 16(5):503 – 513.

[Pearson, 1990] Pearson, W. R. (1990). Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods Enzymol*, 183:63–98.

[Stanke et al., 2006a] Stanke, M., Schoffmann, O., Morgenstern, B., and Waack, S. (2006a). Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC Bioinformatics*, 7:62.

[Stanke et al., 2006b] Stanke, M., Tzvetkova, A., and Morgenstern, B. (2006b). AUGUSTUS at EGASP: using EST, protein and genomic alignments for improved gene prediction in the human genome. *Genome Biol*, 7 Suppl 1:1–8. Evaluation Studies.

[Stein et al., 2002] Stein, L. D., Mungall, C., Shu, S., Caudy, M., Mangone, M., Day, A., Nickerson, E., Stajich, J. E., Harris, T. W., Arva, A., and Lewis, S. (2002). The generic genome browser: a building block for a model organism system database. *Genome Res*, 12(10):1599–1610.

[Tremblett, 1999] Tremblett, P. (1999). X.509 certificates. *Dr. Dobb's Journal*, 24(7):42–51.

[Wolf, 2007] Wolf, A. (DGI FG 2-4 Technical Report, Fraunhofer FIRST, 2007). Spezifikation der D-Grid-Ressourcenbeschreibungssprache D-GRDL.