Extending Maple to the Grid: Design and Implementation

Dana Petcu [*†], Diana Dubu [*†] and Marcin Paprzycki [‡§]
[*] Computer Science Department, Western University of Timişoara, Romania
[†] Institute e-Austria Timişoara, Romania
[‡] Computer Science Department, Oklahoma State University, USA
[§] Computer Science Department, SWPS, Warsaw, Poland
Email: {petcu,ddubu}@info.uvt.ro, marcin@cs.okstate.edu

***Abstract*— One of the important issues facing the development of the grid as the computational framework of the future is availability of grid-enabled software. In this context, we discuss possible approaches to constructing a grid-enabled version of a computer algebra system. Our case study involves Maple: the proposed Maple2g package allows the connection between Maple and the computational grids based on the Globus Toolkit. We present the design of the Maple2g package and follow with a thorough discussion of its implementation.**

I. Introduction

One of the developments that can lead to a wider practical usage of computational grid technologies is grid-enabling of computer algebra systems (CAS). These systems are routinely used by mathematicians and/or engineers to perform complex calculations and are dully seen as a major source of their productivity. However, currently, a number of useful functionalities are implemented in only one particular CAS or as standalone programs; sometimes running best on special hardware such as a parallel computer. In either case, it is desirable to be able to augment the CAS with functionality from another software module. It is the grid technology that should facilitate the necessary infrastructure to support this process [1].

We thus begin this paper by reviewing, in Section II, the state of the art of network and grid-aware CAS. In Section III we follow by summarizing the most important issues in designing grid-enabled CAS and possible approaches to addressing them. We proceed with a practical example of how a CAS can be made grid enabled and for this purpose Maple became our CAS of choice. The main reason is that, despite its robustness and ease of use, we were not able to locate efforts to link Maple with the grid. Furthermore, it is well known that Maple excels other CASs in solving selected classes of problems e.g. systems of nonlinear equations or inequalities [2]. Finally, Maple has already a build-in socket library for communicating over the Internet, and a library for parsing XML. These capabilities match very well with our goal as they suffice to make Maple a client for an external computational service (in this context one should note a recent trend to use the XML syntax as a de-facto standard in the grid community). In Section IV we describe the functionality of Maple2g, the grid-wrapper for Maple. Maple2g consists of two parts a CAS-dependent and a grid-dependent one. Therefore, any change in the CAS or in the grid will be reflected only in one part of the proposed system. Furthermore, the CAS-dependent part is relatively simple and easy to be ported to support another CAS or legacy software. We complete our description of Maple2g, in Section V, with implementation details concerning access to grid services. Our attention is focused on the implementation of the grid service search facility. Some experimental results are provided in Section VI. Further research directions are outlined in the last section.

II. Current network and grid-enabled CAS

Grid enabling and augmenting mathematical software tools with functionality from an external software module(s) is the subject of a number of current research projects. We will thus summarize the most important developments in this area.

A. Accessing external services

Several projects (e.g. NetSolve [3], Nimrod/G [4], Ninf [5]) aim at providing simple ways (APIs, GUIs) to execute software modules available in scientific libraries and/or standalone programs over the Internet/grid. This approach has been commonly labeled “network-enabled server” (NES) [6]. Fully developed NES systems are expected to follow the basic tenets of the grid framework and change the RPC model by incorporating resource discovery, dynamic problem solving capabilities, load balancing, fault tolerance, security, etc.

Currently, the NetSolve system [3] seems to be the most developed NES. It is a grid based server that, among others, supports Matlab and Mathematica as native clients for grid computing. NetSolve provides a web tool that users can query for information concerning all available software modules within the NetSolve system. Furthermore, NetSolve searches for computational resources across the network, chooses the best one available, and using retry for fault-tolerance solves the problem, and returns the answers to the user. A load-balancing policy is used by the NetSolve system to ensure good performance by enabling the system to use available computational resources as efficiently as possible. Recently, a proxy was built for the NetSolve client that knows how to interact with and make use of Globus resources.

MathLink [7] enables Mathematica to interface with external programs via an API interface. Such an external program

sends its arguments to a mathematical computation service and returns result directly into Mathematica.

MathGridLink [8] permits the access to the grid service and deployment of new services entirely from within Mathematica. MathGridLink allows two ways of interaction: one from the view-point of a Mathematica user who wants to use an existing service, and the other from the viewpoint of a grid user, who wants to access Mathematica as a grid service.

Finally, the Geodise toolkit [9] is a suite of tools for grid-services which are presented to the user as Matlab functions. The user of the Geodise toolkit acts as a client to the remote computational resources. Users are authenticated, and then authorized to access resources for which they have rights. The user is able to discover available resources, to decide where to run a job, to monitor its status, and to retrieve its results. The functions implemented in the "language of Matlab" call Java classes which in turn access the Java CoG API [10].

Note that CASs like Matlab and Mathematica are used mainly as interfaces for grid services (e.g. in Geodise and not as tools offering services on grids, while MathGridLink envisages support of both types of activities. Looking to both approaches we try to add a new functionality to the system by developing a wrapper that facilities the development of grid-oriented high performance computing codes based on a standard message-passing interface.

B. Availability of interactive mathematical web content

Let us now consider a particular situation of interactive access to web enabled computational resources. This scenario can be achieved in a limited way using applets in a web browser. Observe however that computing even the most fundamental mathematical operations such as an integral can require a complicated software module and thus, it is usually necessary to incorporate existing mathematical software into a web application to achieve the required functionality. To implement interactive mathematical web content the following steps are required [11]: install/maintain the "external" computational component, write the wrapper for this component to enable it to be called from another program, write an applet to present the interactive element together with a (most likely Java) servlet which will interact with the wrapper, and write the content and embed the applet or form into the text with the appropriate parameters. JavaMath SDK [12] can assist the user in this process enabling the development of conglomerate systems in Java from existing components. It gives a template for writing wrappers and an API for creating and using sessions utilizing these components. For example the code would be part of a servlet on a server, and it would make use of Maple running on a JavaMath server.

To achieve a single generic mechanism which could be used for all computation requests with no extra software that needs to be loaded into the CAS to interface with each new online service, it is necessary to establish a standard for the request-response exchanges. Part of this is a standard for the representation of the mathematical objects to be exchanged. MathML [13] is well advanced on the path to solving this problem.

In the context of our paper, where one of our goals is to explore the possibility of adding Maple modules to the set of grid available services, we note that MapleNet [14] offers a software platform for effective large-scale deployment of comprehensive content involving live math computations. MapleNet client is an applet which encapsulates the mathematical content; MapleNet publisher offers tools to create applet-based exploration tools. MapleNet server coordinates all the essential software infrastructure, including the general web server, math engines, content, and other databases; it manages concurrent Maple instances as required to serve client connections for math computation and display services, and it provides some additional services including user authentication.

C. Parallel/distributed CAS versions

While thus far we have mostly discussed the possibility of making the CAS available as a part of grid services, obviously it can be beneficial if the CAS is capable of utilizing the computational capabilities of the grid itself. In this context observe that coarse grain parallelism can be very efficient in an interpreted computation environment such as the CAS. To be able to facilitate development of coarse-grain parallel grid distributed CAS applications, a CAS interface to a message-passing library is needed.

gridMathematica [15] allows the distribution of Mathematica tasks among different kernels in a distributed environment. It is built on a PVM-like architecture. A typical installation of gridMathematica has one master kernel and several computational kernels: the master kernel handles all inputs, divides computations into independent subtasks, schedules calculations for the computation kernels, and collects the results.

There exist more then 30 parallel projects involving Matlab (for more details and a list of projects see [16]). They use diverse approaches to achieving their goal(s): compile Matlab script into a parallel native code, provide a parallel backend to Matlab using Matlab as a graphical frontend, or coordinate multiple Matlab processes to work in parallel. For example, Matlab*P 2.0 [16] is a parallel Matlab environment using the backend support approach. MatlabMPI [17] implements basic basic MPI routines like send, recv, size and rank entirely in Matlab scripts. PVMTB [18] is a complete Matlab interface to PVM, by means of which Matlab users can prototype applications in the usual high-level programming environment, while retaining the ability to make PVM calls.

Distributed version of Maple have been recently reported in [19] and [20]. For example, Parallel Virtual Maple [19] (PVMaple), was developed to allow several independent Maple kernels on various machines connected by a network to cooperate in solving a problem. This is achieved by wrapping Maple into an external system which takes care of the parallel execution of tasks: a special binary, the command-messenger, is responsible for the message exchanges between the Maple processes, coordinates the interaction between Maple kernels via PVM daemons, and schedules tasks among nodes. Initial experiments show sufficient efficiency in solving large problems to follow this path in Maple2g which has a number of

similar functionalities with PVMaple [21]. In the context of this paper it has to be stressed that while there exist attempts at developing *distributed* Maple, there were no attempts at developing *grid-enabled* Maple (and these are somewhat similar, but different goals), which is the goal of our current research.

In summary, there exist a large number of projects that attempt at grid enabling known computer algebra systems. Their main goals are: (1) to make CAS modules available through the grid, (2) to allow CAS to utilize the grid, (3) to provide direct, web-based access to CAS modules, and (4) to develop parallel and/or distributed CAS by utilizing the networked/grid environment and message-passing parallelism. Out of these goals 1, 2 and 4 are of particular of interest to us in this paper. As indicated above, in the case of Maple, goal 3 has been already mostly achieved and thus will be omitted.

III. Developing a grid-aware CAS extension

Let us now look in a bit more details into main issues involved in developing grid enmeshed CAS systems.

Our analysis of the grid aware CAS systems presented in the previous section indicates, that any such a system must have at least the following facilities (Figure 1):

Ability to accept services from the grid:
: the CAS must be opened to augment its facilities with external modules, in particular it should be able to explore computational grid facilities, to connect to a specific grid service, to use the grid service, and to translate its results for the CAS interface;

Being a source of grid or web services:
: the CAS or some of its facilities must be seen as grid or web services and allowed to be activated by remote users under appropriate security and licensing conditions; furthermore, deployment of the services must be done in an easy way from the inside of the CAS;

Ability to communicate and cooperate over the grid:
: similar or different kernels of CASs must be able cooperate within a grid in solving general problems; in order to have the same CAS on different computational nodes a "grid-version" of the CAS must be available; in the case of different CASs, appropriate interfaces between them must be developed and implemented or a common languages for intercommunication must be adopted.

There exist multiple ways of achieving the above described functionalities. Rewriting a CAS kernel in order to grid-enable it is likely to be a complicated, time-consuming and high-cost solution. Wrapping the existing CAS kernel in a special code acting as the interface between the grid, the user and the CAS can be done relatively easily as an added-functionality to the CAS. Moreover it can be adapted "on-the-fly" when new versions of the grid software or the CAS in question become available. It is therefore the latter solution that we advocate and pursue here. Let us now describe each of the three functionalities in more details.

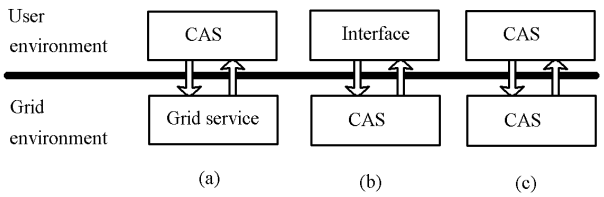

Fig. 1. Operating modes between a CAS and a computational grid: (a) CAS as an interface for the grid services; (b) CAS as grid service; (c) multiple CAS kernels on user and grid sides

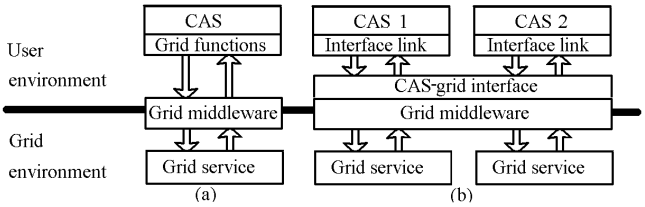

Fig. 2. CAS openness towards grid services: (a) using specific grid-aware function library; (b) using a general CAS-grid interface

A. CAS input from grids – importing grid services in a CAS

Most CASs have the possibility to launch system commands or to call external modules written in non-native languages. Using these facilities special libraries can be constructed in the CAS language, describing in an user-friendly manner, the calls to the grid middleware tools, like those provided by the Globus environment. On the user side, some minimal facilities to access the computational grid and the CAS must be installed. The grid facilities which must be provided to the user are those currently provided by the grid middleware.

The interface between the grid middleware and the CAS can be written entirely in the CAS language or partially in the CAS language and partially in some other language, more appropriate for the grid middleware (for example, in the case of Globus, such a language would be Java CoG). In the first approach the added-code is oriented towards a particular CAS and is not portable (Figure 2). The second approach can be more flexible in integrating a new CAS in the user environment and this approach will be pursued here.

It is worth mentioning, that the Geodise project [9] has already adopted the second approach. The Geodise toolbox includes a Java-grid client and a special library mapping current Globus line commands into the Matlab environment. Java-grid client interacts with the Globus server, sending and receiving information from and to the location service(s), authorization service and metadata archive/query service(s). Acting on user request, via the special Matlab functions, it sends to the Matlab interface data concerning the available grid services, and then makes connections to the specific service(s).

B. CAS output toward grids – deploying CAS services on grid

The access to the CAS facilities must be available to the user of the computational grid.

MapleNet [14], allowing the secure access of a thin-client to a Maple server, gives a good example for grid-enabling the CAS: the entire functionality of the CAS can be exposed to the computational grid (respecting the license conditions). Full

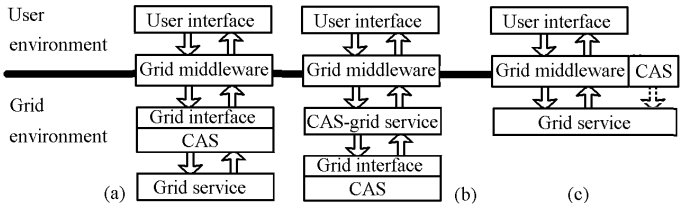

Fig. 3. CAS as grid service: (a) entire functionality exposed as grid-service; (b) partial exposing; (c) using CAS to create stand-alone grid-services

access to the CAS functionality can further give access to other grid services. To achieve this the entire CAS kernel is rebuild to construct a multi-threaded version or a wrapper is build in order to launch multiple CAS kernels for each user request.

The CAS functionality can be also exposed only partially with the possibility to install a visibility hierarchy with different levels of security (Figure 3).

A CAS installed in the user or in the grid environment can be used to deploy services in other languages than the one provided by the CAS, using its facilities to export codes.

C. CAS over the grids – grid-aware distributed CAS version

The computational power given by a CAS can be augmented by using several other CAS kernels (the same or different CASs) when the problem to be solved can be split between these kernels or a distributed-memory parallel method is used in order to solve it. The usage of a standard message-passing interface for inter-kernel communication allows the portability of the parallel version of a CAS in particular the easy deployment on clusters and grids (Figure 4).

The two extreme approaches to design the interaction with the message-passing interface are minimal, respectively full, access to the functions of the message-passing interface. In the first case the set of functions is restricted to those allowing to send commands and receive results from the remote kernels. In the second case it is possible to enhance the CAS with parallel or distributed computing facilities, allowing the access of the CAS to other parallel codes than the ones written in the CAS language (the message-passing interface can be used as interpreter between parallel codes written in different languages, including those of different CASs).

D. A functional approach

In the next section we describe a prototype of a grid-enabling wrapper for Maple. Having in mind the above described approaches, we have considered the following roles as the most appropriate for our prototype:

1) the CAS-grid-interface from Figure 2.b,
2) the CAS-grid-service from Figure 3.b,
3) the master/worker interfaces from Figure 4.

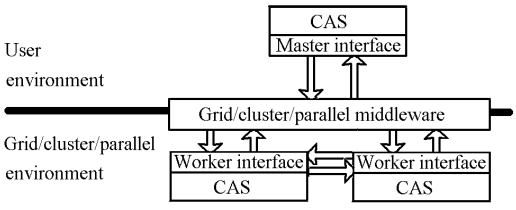

Fig. 4. Grid-version of a distributed CAS

The first selection was made so that any change in the CAS or in the grid will be reflected only in the corresponding part of the wrapper. Moreover the CAS-dependent part must be relatively simple and easy to be ported to support another CAS or legacy software. The same idea is motivating also the second selection. Finally, grid enabling should also lead to the capability for large-scale distributed computing and thus the last choice.

IV. Case study: Maple2g

We proceed with a practical example of how a CAS, in our case Maple, can be made grid enabled. Maple2g package allows the connection between Maple and computational grids based on the Globus Toolkit.

The prototype of a grid-enabling wrapper for Maple, consists of two parts a CAS-dependent and a grid-dependent one:

- *m2g*, the Maple library of functions allowing the Maple user to interact with the grid/cluster middleware;
- *MGProxy*, the middleware, a package of Java classes, acting as interface between m2g and the grid environment.

The m2g functions are implemented in the Maple language, and they call MGProxy which accesses the Java CoG API.

Maple2g has three operating modes:

user mode	for external grid-service access;
server mode	for exposing Maple facilities as grid services;
parallel mode	for cooperative Maple kernels over the grid.

Let us now discuss in more details these operating modes.

A. User mode: grid-service access in Maple2g

In order to make grid services available to the user of the CAS the coupling with the services exposed within the grid has to be performed in a transparent way. This implies that the service methods call should be done only in the CAS native language syntax. In order to achieve this we have incorporated a suite of Maple functions which allow the communication with the services available within the grid in the package m2g.

In the current version of Maple2g we have implemented a minimal set of functions (described in Table I) allowing access to the grid services.

MGProxy is activated from inside the Maple environment by the m2g command m2g_MGProxy_start. The user command(s) from the user's Maple interface are send to the MGProxy via a socket interface, when m2g_getservice and m2g_jobsubmit

TABLE I

M2g functions enabling Maple to use grid services

Function	Description
m2g_connect()	Connection via Java COG to the grid
m2g_getservice(c, l)	Search for a service c and give a link to it, retrieve its location l
m2g_jobsubmit(t, c)	Based on the service location retrieved in the previous step, perform a job submission, in the grid environment, labeled t: the command from the string c is send to the MGProxy which treats it as a grid-service request
m2g_results(t)	Retrieve the results of the submitted job labeled t

```
> with(m2g);
  [m2g_connect, m2g_getservice, m2g_jobstop,
   m2g_jobsubmit, m2g_maple, m2g_MGProxy_end,
   m2g_MGProxy_start, m2g_rank, m2g_recv,
   m2g_results, m2g_send,m2g_size]
> m2g_MGProxy_start(); m2g_connect();
  Grid connection established
> m2g_getservice("newton",'service_location');
   ["&(resourceManagerContact="myri1.info.uvt.ro")
      (count=1) (label="subjob 0")
      (directory=/home/Diana)
      (executable=/home/Diana/newton)",
    "&(resourceManagerContact="myri8.info.uvt.ro")
      (count=1) (label="subjob 0")
      (directory=/home/Dana/g)
      (executable=/home/Dana/g/newton)",]
> m2g_jobsubmit(3,service_location[1]);
    job submitted
> m2g_results(3);
  Solving nonlinear system with Newton method:
  Input in.txt, Output out.txt
> m2g_MGProxy_end();
  Grid connection closed
```

Fig. 5. Accessing in Maple an external numerical nonlinear solver, available as grid service

are invoked. MGProxy contacts the grid services, queries the contacted services, and sends to the Maple interface the results of performed queries. By the m2g_receive, the user gets the results. The Maple commands are passed in the system as strings and the results are presented in the MathML format.

For example, accessing a grid-service can be achieved through the sequence of steps presented in Figure 5. Implementation details are provided in the next section.

B. Server mode: Maple services on grid

Concerning access to Maple as service, Maple2g is similar to MapleNet [14]. The main difference is that instead a new version of Maple, we have used the classical kernel and a wrapper.

In the current version of the Maple2g prototype, the access to the fully functional Maple kernel is available from the grid: we have implemented only an account check procedure in order to verify the user rights to access the licensed version of Maple provided on the grid. Obviously, our system can be modified to restrict user-access to a subset of Maple commands or function libraries, but this type of enhancement is outside of focus of our current interest.

The user interface activates a simple Java applet which allows the user to send Maple commands as strings via a socket connection to a local Maple2g process awaken in the user mode by the Java applet initialization.

The connection with the remote Maple kernel is established at the initialization stage by sending a specific string in the format in which m2g_jobsubmit sends the information, specifying in this case the remote MGProxy as the grid-service. MGProxy activates a Maple process (which enters an infinite cycle of interpreting commands incoming via the socket interface from the MGProxy), acts as a server waiting for external calls, interprets the requests, sends the authentications requests to the Maple twin process, waits for the Maple results returned in the MathML format.

TABLE II

M2G FUNCTIONS FOR REMOTE PROCESS LAUNCH/COMMUNICATIONS

Function/const.	Description
m2g_maple(p)	Starts p processes MGProxy in parallel modes
m2g_send(d, t, c)	Send at the destination kernel labeled d a message labeled t containing the Maple command c; d and t are numbers, c is a string; when 'all' is used in destination field, c is send to all Maple kernels
m2g_recv(s, t)	Receive from the source kernel labeled s a message containing the results from the a previous Maple command which was labeled with t; when 'all' is used in source field, a list is returned with the results from all Maple kernels which have executed the command labeled t
m2g_rank	MGProxy rank in the MPI World, can be used in a command
m2g_size	Number of MGProxy processes, can be used in a command

C. Parallel mode: message passing interface in Maple

Parallel codes using MPICH as the message-passing interface can be easily ported to grid environments due to the existence of a MPICH-G version which runs on top of the Globus Toolkit. On other hand, the latest Globus Toolkit is build in Java, and the Java clients are easier to write. This being the case, we selected mpiJava [22] as the message-passing interface between Maple kernels.

In Maple2g a small number of commands have been implemented and made available to the user, for sending commands to other Maple kernels and for receiving their results (Table II).

MGProxy is activated from user's Maple interface with several other MGProxy copies by m2g_maple command. The copy with the rank 0 enters in user mode and normally runs in the user environment, while the others enter in server mode. Communication between different MGProxy copies is done via mpiJava.

These facilities are similar to those introduced in the PVMaple [19] and in the Distributed Maple [20]. The user's Maple interface is seen as the master process, while the other Maple kernels are working in a slave mode. Command sending is possible not only from the user's Maple interface, but also from one kernel to another (i.e. a user command can contain inside a send/receive command between slaves).

As a side-note, we have tested the feasibility of this approach to development of distributed Maple applications on a small PC cluster. We have observed a reasonable speedup obtained when splitting time-consuming computations. Detailed report as well as a complete description of functionality of this component of Maple2g can be found in [23].

V. ACCESSING GRID SERVICES FROM MAPLE: IMPLEMENTATION DETAILS

The Maple functions made available through the Maple2g package m2g allow the programmatic access to Globus grid enabled resources. The m2g package translates internally functions from the syntax familiar to the Maple user into commands, allowing the initiation and further communication with the MGProxy middleware.

MGProxy acts as an intermediary between Maple and the grid and has been written in Java, due to its portability

TABLE III

MGPROXY ACTIVATION AND RETRIEVAL OF GRID SERVICES' PROCESSING

Action	Description
Activate MGProxy	User commands in Maple syntax, parsed within the m2g package, initiate the communication with the middleware which acts as an intermediary between Maple and the grids. This is performed via the commands for external code invocation (including Java) which are already available in Maple 8
Grid Services' invocation	The user invokes the remote services by issuing commands in RSL syntax
Job Submission	MGProxy activates GridJob, a Java class encapsulating GRAM job that deals with job submission over the grids
Results Retrieval	Results can be requested either during the communication or after closing the grid connection

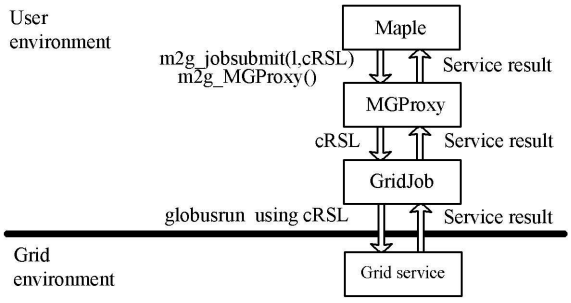

Fig. 6. Communication flow when accessing grid services from Maple

and to the fact that libraries supporting Globus-based grid computing have been already implemented in Java. Java CoG (i.e. Commodity Grid) kit provided with Globus integrates the software for grid computing developed by Globus and the Java commodity framework, thus facilitating the development and deployment of grid services, while also permitting the use of Web services as parts of the grid.

The procedural steps to communicate within the grid starting from the user's Maple interface are described in Table III and depicted in Figure 6.

Table IV enumerates the main classes of the MGProxy package. MGProxy can be viewed as the entry point to the grid. The commands issued by the user from her Maple interface are passed as strings to the MGProxy which forwards these messages further to the MapleListener class responsible for parsing the messages and calling the appropriate tool for their management.

For the invocation of grid services MapleListener activates, according to the request, either MDSService or GridService. MDSService is responsible for the retrieval of information

TABLE IV

JAVA CLASSES IN MGPROXY PACKAGE

Name	Description
MGProxy	Activate the Maple link
MapleListener	Parse the Maple messages
MDSService	Retrieve information regarding the grid resources
GridService	Server for GridJob
GridJob	Client performing the requested job
MapleService	Used for Maple as grid service
MPIMaple	Used for parallel Maple

regarding the grid-available resources, including the software resources. The current version is based on the first of the two approaches for the discovery of services described below. GridService acts as a server for the GridJob which is the client performing the actual task of submitting the job; the client-server communication being established via sockets. GridService receives the commands to be send over the grid to the available services from MapleListener until an 'end of job' is signaled, meaning that the connection with the grid is no longer necessary.

The underlying principles for service retrieval in Globus, referred to as Monitoring and Discovery Service, can be implemented in two alternative ways. Either, the "old" MDS can be used, or the facilities from the new GT3 can be utilized. In Maple2g we have implemented the first approach. In the near future, we intend to implement also the second one and compare their performance.

A. First approach: using the MDS

The Globus toolkit provides a directory service which has the functionality of the white pages directory and yellow pages directory and is named the Metacomputing Directory Service (MDS) [24]. MDS makes available the information regarding the computational resources within the grid and the grid network, i.e. information about the hardware, the software and the system status.

While the White Pages offers the information concerning the hardware performance, Yellow Pages deal with the computers of a particular class or with a particular property. This later organizational principle is what we are interested in.

MDS is based on LDAP (Lightweight Directory Access Protocol) which is a software protocol for enabling the localization of organizations, individuals, and other resources such as files and devices in a network, whether on the public Internet or on a corporate intranet. LDAP, as a version of Directory Access Protocol, is part of the X.500, a standard for directory services in a network. A directory provides information within a network about the localization of components. LDAP allows the search for an individual without knowing where its particular location.

An LDAP directory is organized in a simple tree hierarchy consisting of the following levels: root directory, countries, organizations, organizational units, individuals. An LDAP directory can be distributed among many servers. Each server can have a replicated version of the total directory that is synchronized periodically. A LDAP server is called a Directory System Agent (DSA). An LDAP server that receives a request from a user takes responsibility for the request, passing it to other DSAs as necessary, but ensuring a single coordinated response for the user.

In order to make use of the LDAP principles within Globus, several steps have to be performed, namely:

1) *Initialization.* LDAP schema files must be updated with the description of the attributes associated with grid-specific entry classes. The directory structure described

TABLE V

SEQUENCE OF STEPS FOLLOWED IN ORDER TO POPULATE THE LDAP DATABASE AND SEARCH FOR AVAILABLE SERVICES

Command	Result
slapd -f <conf file>	Activate the LDAP server with the configuration from slapd.conf
ldapadd -h localhost -a -w <passw> -x -D <binddn> -f <.ldif file>	Populate LDAP database with the entries specified in the LDIF file
ldapsearch -x -s <scope> -b <baseDN> filter -p <ldapport>	Search for the objects specified within the filter starting in the directory from the baseDN. The scope restricts the search level
grid-info-mds-core	Retrieve the information for the above queries such as Globus directory, base DNs of servers, slapd process ids.
ldap stop	Stop the ldap server

above has therefore to be adapted such that it would contain the information regarding the grid resources

2) *Population.* The LDAP directory has to be further populated with information according to the hierarchies established at the initialization phase.
3) *Querying.* Is the essential step as it represents the request for information. The information is retrieved from the directory service.

For the Initialization step, there exists a naming schema for the MDS reported in [25]. The MDSService class was implemented starting from the MDSService class proposed in [26].

We have used a combination of LDAP and Globus commands in order to perform the above operations. Alternatively, Java APIs can be used. The suite of commands and their use is depicted in Table V.

B. Second approach: using OGSI services from GT3

As an alternative to the MDS the OGSI services from GT3 (Globus Toolkit 3) can be used. Here, the data format is not LDIF but XML and the reception, storage and delivery of the data is performed within the Service Data containers. The schema used is GLUE (Grid Laboratory Uniform Environment) [27]. The functions provided within Globus GT3 for the management of data and thus services are described on the Globus webpage [28] and the functions of interest are summarized in Table VI.

TABLE VI

OGSI SERVICES FOR SERVICE INFORMATION RETRIEVEL

Command	Action
ogsi-find-service-data	Command-line interface for querying the service data available from any Grid service
ogsi-find-service-data-by-name	Search for Service Data Element values in a service by name
ogsi-set-service-data-by-name	Add Service Data Elem. values to a service
ogsi-delete-service-data-by-name	Delete Service Data Elements in a service
ogsi-add-service	Add a service to a Service Group Registration service supporting remote registration
ogsi-remove-service	Remove a service from a Service Group Registration service supporting remote registration

TABLE VII

GRIJOB METHODS

Method	Description
GridJob(C, p, b)	Contructor responsible for the initialization of the contact string variable C, gatekeeper port p and submission mode b (i.e. whether batch or not)
startGassServer(*credent*)	Starts the Globus GASS Server. Retrieves the output from the GASS server and sends it to the client via GridService and MapleListener as a string
initJobOutListeners()	Initiate/register listeners for non-batch mode jobs
statusChanged(*job*)	Used to notify the implementer when the status of a GramJob has changed. A waiting thread is notified when a job is finished and when this is the case the URL is returned and output
outputChanged(*output*)	When the *output* is modified, performs an update
GlobusRun(RSL)	The default Globus proxy is loaded and user credentials are setup properly. The GASS server is started. The RSL is formatted accordingly to the expected structure. A GramJob instance is created and the object sends a request to the remote host

C. Grid job submission

Once the information regarding the existing services has been obtained - whether it is a script which is activated within the Java code or via the functions provided by a Java API - the subsequent step is to deploy such service in order to retrieve the result. Information is passed in the form of strings, the GridJob class being responsible for the job submission to the service provider which in turn sends back the result of computation, again in the String format. This result is further retrieved by the Maple user, which can use it in subsequent operations. The results are valid even after the connection is closed, thanks to the label which identifies them.

The GridJob class incorporates the methods described in Table VII. It was written starting from the class proposed in [29]. The GridJob class is responsible for the job submission. Requests received from the Maple's user interface in the RSL syntax are send over the grids to remote resources. The underlying framework used here is Java CoG. The connection is established with the remote server, referred to as the 'gatekeeper', which is responsible for the execution of the job (i.e. a binary executable or command to be run remotely). Both the host and the gatekeeper must comply with the authentication requirements. The Grid Security Infrastructure (GSI) is used for enabling secure authentication and communication over an open network. Globus uses GASS for porting and running the applications requiring I/O files to the Grid environment. Therefore, GridJob starts the GASS server and submits all GRAM job requests to this server. The request is formatted accordingly in the RSL format. Output of the processed job is returned through the MGProxy intermediary to Maple into the user's interface.

VI. EXPERIMENTAL RESULTS

In order to test the performance of the grid-wrapper (not the efficiency of problem solving) we have performed several tests on a small Globus-based grid environment: 2 local PCs each with a P4 processor running at 1.5 GHz and 256 Mb of memory, connected in a cluster via a Myrinet switch at

TABLE VIII

TIME RESULTS (MEAN VALUES FOR 5 RUNS)

Package	Dimension	Local	Cluster	Remote	Maple
Newton	5 eqs.	18 s	38 s	74 s	0.3 s
	20 eqs.	435 s	460 s	496 s	934 s
Gauss	5 eqs.	18 s	37 s	73 s	0.01 s
	100 eqs.	535 s	557 s	596 s	822 s

full 2Gb/s, and a remote PC (located in Linz), with a P4 processor running at 2.4 GHz and 512 Mb of memory. We have experimented with two codes:

Newton package the Maple2g code from Fig.5, a solver for system nonlinear equations based on Newton's method, written in C, and the Maple's *fsolve* function;

Gauss package the same Maple2g code, a linear system solver based on Gauss' elimination written in Java, and the Maple's *linsolve* function.

The test problem was: solve $\sum_{j=1}^{n} a_{ij} x_j^{f(i)} = b_i, i = 1, \ldots, n$, where $f(i) = 1$ in the linear case, $f(i) = i$ in the nonlinear case, while A and b are random matrices.

Table VIII presents the most significant results. The user's Maple interface was executed on one of the local PCs. The grid-service was launched on (note that in each case code solving the problem was executed on a single computer):

Local: the same computer as the Maple user interface;
Cluster: on the other PC in the cluster (see above);
Remote: on the remote PC (in Linz);
Maple: only the local Maple was invoked (no grid).

The results indicate that for large problems, Maple2g user can efficiently utilize the external code(s) residing on the grid (the apparent "inefficiency" of Maple is related to the particular approach to solving our problems and should be ignored). Times obtained for small problems estimate the overhead introduced by the Maple external code launcher, the Maple2g, and the Globus middleware and the network. The overhead is almost independent of the problem size (small differences result from the size of exchanged messages).

Separately, we have tested parallel performance of Maple2g and the detailed results can be found in [23].

VII. CONCLUDING REMARKS

Developing grid-enabled computer algebra systems is a necessary part of the emergence of true value of grid computing. Several approaches to construct such systems were discussed in this paper.

Following one such path, we have developed Maple2g, a wrapper for Maple, enabling it to access the grid services and to be accessed as a grid service. Furthermore, Maple2g allows distribution of computational effort to several Maple kernels running on a parallel computer, a cluster, or a grid.

At this stage Maple2g exists as a demonstrator system with all of the functionalities described above implemented. In the near future it will be further developed to include facilities existing in other systems, in order for it to become comparably robust as NetSolve (in issues like load balancing, fault tolerance, security) or Geodise (in issues like monitoring and authentication).

We will also perform experiments on the grid on a large domain of problems. Experimental results will help guide further development of the system. Deployment of grid services from Maple in other languages than Maple using the code generation tools must be take also into consideration. The next MGProxy version will allow the cooperation between different CAS kernels lying on the same or different sites of a computational grid.

REFERENCES

[1] I. Foster, C. Kesselman, *The Grid. Blueprint for a new computing infrastructure*, Morgan-Kaufmann, 1999.

[2] M. Wester, "A critique of the mathematical abilities of CA systems", in *Computer Algebra Systems: A Practical Guide*, ed. M.Wester, John Wiley & Sons, 1999, http://math.unm.edu/~wester/ cas_review.html

[3] H. Casanova and J. Dongarra, "NetSolve: a network server for solving computational science problems",in *Inter.J. Supercomputer Appls.& HPC* 11(3), 212–223 1997, http://icl.cs.utk.edu/netsolve/

[4] D. Abramson, J. Giddy, L. Kolter, "High performance parametric modelling with Nimrod/G: A killer application for the global grid?", in *Proc. IPDPS*, 2000, 520–528, http://www.csse.monash.edu.au/~davida/papers/ipdps.pdf

[5] H. Nakada, M. Sato, S. Sekiguchi, "Design and implementations of Ninf: towards a global computing infrastructure", in *Future Generation Computing Systems, Metacomputing Issue*, 15(5-6), 1999, 649–658.

[6] S. Matsuoka, H. Casanova, "Network-enabled server systems and the computational grid", in *Proc.GF4-WG3*, 2000, http://www.eece.unm.edu/~apm/WhitePapers/GF4-WG3-NES-whitepaper-draft-000705.pdf

[7] Wolfram Research,MathLink,www.wolfram.com/solutions/mathlink/.

[8] D. Tepeneu and T. Ida, "MathGridLink - A bridge between Mathematica and the Grid", in *Proc. JSSST*, 2003, in print.

[9] M.H. Eres, G.E. Pound, Z. Jiao, J.L. Wason, F. Xu, A.J. Keane, J.S. Cox, "Implementation of a grid-enabled problem solving environment in Matlab", in *Proc. WCPSE*, 2003, in print, http://www.geodise.org

[10] Java CoG Kit, http://www-unix.globus.org/cog/java/.

[11] A. Solomon, "Distributed computing for conglomerate mathematical systems", in *Integration of Algebra and Geometry Software Systems*, eds. M. Joswig, N. Takayama, http://www.illywhacker.net/papers/webarch.ps

[12] A. Solomon and C.A. Struble, "JavaMath: an API for internet accessible mathematical services", in *Proc.5th Asian Symp.on Computer Mathematics*, World Scientific, 2001, http://javamath.sourceforge.net/.

[13] MathML, The W3C's Math Homepage, http://www.w3.org/Math/.

[14] MapleNet, http://www.maplesoft.com/maplenet/.

[15] Wolfram Research, gridMathematica, http://www.wolfram.com.

[16] R. Choy, A. Edelman, "Matlab*P 2.0: a unified parallel MATLAB", in *Proc.2nd Singapore-MIT Alliance Symp.*, 2003, in print.

[17] J.Kepner,"Parallel programmimg with MatlabMpi",in *Proc.HPEC*,2001

[18] J.F. Baldomero, "Parallel Virtual Machine Toolbox", in *Proc. MATLAB*, ed. S. Dormido, 1999, 523-532, http://atc.ugr.es/javier-bin/pvmtb_eng

[19] D.Petcu,"PVMaple:A distributed approach to cooperative work of Maple processes",*LNCS* 1908,eds.J.Dongarra et al.,Springer,2000,216–224.

[20] W. Schreiner, C. Mittermaier, K. Bosa, "Distributed Maple – parallel computer algebra in networked environments", in *J. Symbolic Computation* 35(3), Academic Press, 2003, 305–347.

[21] D.Petcu, D.Dubu, M.Paprzycki,"Towards a Grid-aware Computer Algebra System", *LNCS* 3036, eds.M.Bubak et al, Springer, 2004, 490–494.

[22] mpiJava, http://www.npac.syr.edu/projects/pcrc/HPJava/mpiJava.html

[23] D. Petcu, D. Dubu, M. Paprzycki, "A Grid-based Parallel Maple", submitted to EuroPVM/MPI 2004.

[24] G. von Laszewski and I. Foster, "Usage of LDAP in Globus", CSE 225 (High Performance Distributed Computing), http://www.globus.org/mds/globus_in_ldap.html

[25] MDS 2.2 Schemas. Definition of Schema, http://www.globus.org/mds/Schema.html

[26] V. Silva, Querying the Grid with the Globus Toolkit Monitoring and Discovery Service, http://www-106.ibm.com/developerworks/grid/library/gr-mds.html

[27] Grid Laboratory Uniform Environment, www.hicb.org/glue/glue.htm

[28] Globus Toolkit 3.2: Developer's Guide, http://www-unix.globus.org/toolkit/docs/3.2/developer/commandlineclients.html.

[29] V. Silva, Grid Job submission using the Java CoG Kit, http://www-106.ibm.com/developerworks/library/gr-gridcog.html