Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 111–125. http://www.scpe.org
ISSN 1895-1767
© 2008 SCPE

# SERVICE-ORIENTED SYMBOLIC COMPUTING WITH SYMGRID

DANA PETCU, ALEXANDRU CÂRSTEA, GEORGIANA MACARIU, AND MARC FRÎNCU*

**Abstract.** Recent software engineering concepts, like software as a service, allow the extension of the legacy code lifetime and the reduction of software maintenance costs. In particular, exposing computer algebra systems as services allows not only the integration in complex service-oriented architectures but also their further development. While existing standards may be used for service deployment, discovery and interaction, the particularities of services to be built require specialized solutions. A recent technical approach aimed at integrating legacy computer algebra systems into modern service-oriented architectures is presented and discussed in detail in this paper. A special emphasis is put on the ability to compose symbolic services in complex computing scenarios. A short description of how such systems were extended to allow the access of external services is provided as well. The proposed approach was implemented into a specific framework, namely, SymGrid-Services. Simple examples are provided to demonstrate usefulness of the framework.

**Key words:** service-oriented architecture, symbolic computing, wrappers for legacy software, service composition, computer algebra systems.

## 1. Introduction.

Symbolic computation is one of the most high demanding fields in terms of computer power as well as memory requirements. The current tools for symbolic computations are the Computer Algebra Systems (CAS). Standard CASs are designed to be used in the isolated context of the local machine on which they are installed. Their ability to interact with the external world is mostly restricted to a command line user interface, to the file I/O based operations and only occasionally the ability to interact using the TCP/IP sockets. The issues related to the data exchange between CASs were partially solved in the last decade by the introduction of OpenMath [24] and MathML [36] standards (XML-based data description format, designed specifically to represent computational mathematical objects). However, even in this case the inter-operability between the CASs is still an open problem.

Recent solutions to the CAS inter-operability problem through service-oriented approaches where proposed by an on-going international collaborative project, SCIEnce. This paper presents an overview of the achievements of the SCIEnce project relative to a particular middleware component that was developed in the last one and a half year, the SymGrid-Services. While this paper is an extended version of the [5] and focuses mostly on the wrapper services for CASs, it includes also a short descriptions of the other components that allow the composition of the wrapper services, as well as the seamless access of services within CASs, (these components are detailed in [3, 4, 6, 18]).

Overall, the paper is organized as follows. Section 2 discusses shortly the related work. Section 3 presents the SymGrid and its main components. Section 4 points to the solution proposed for service access from inside a CAS. Section 5 goes deep inside the solution adopted to present CASs as services. Section 6 gives some hints about the solution adopted for service compositions. Finally, Section 7 draws the conclusions and highlights the future steps.

## 2. Related work on mathematical Web and Grid services.

The problem of integrating legacy software into modern distributed systems has two obvious solutions: reengineering of the legacy software (an invasive procedure) and creation of wrapper components (an non-invasive procedure).

Due to the complexity of the computer algebra systems, a non-invasive procedure is more appropriate, as stated in [8, 34]. Several efforts have revealed the paths to be followed in wrapping CAS. We present in what follows the most relevant ones. A more detailed overview of the bellow described initiatives can be found in [26].

### 2.1. General wrapping solutions.

Exposing functionality of legacy components can be done using general wrapper tools. In this category we can include Soaplab [30] and JACAW [13].

More specifically geared toward mathematical problems is JavaMath [33], a Java API for connecting to mathematical software systems (a request is made to a broker for a particular software system and the broker establishes a session to such system). OpenMath encoding of objects can be used. An specific abstract interface for service access is given, but there is little abstraction from service implementation.

*Institute e-Austria Timişoara, România, B-dul Vasile Pârvan 4, 300223 Timişoara, Romania (science@ieat.ro).

**2.2. Web services.** Result of an academic activity, the MathWeb-SB [41] is a software bus that allows to combine mathematical services like computer algebra systems: a broker provides access object for service by name, ensuring the abstraction from service locations and from object encodings, but there is no way to interact with unknown services.

Internet-Accessible Mathematical Computation (IAMC) site [14] maintains a list of projects and systems related to mathematics that are accessible on the Internet; in the frame of the IAMC project a HTTP-like protocol was developed for server-client communication—a non-standard informal description of service and an abstract protocol for service access (machine-readable) were provided, but it requires insight to be used (not machine-understandable).

A general framework for the description and provision of Web-based mathematical services was designed within the MONET [22], aiming at demonstrating the applicability of the semantic Web to the world of mathematical software. It allows dynamic discovery of services based on published descriptions which express both their mathematical and non-mathematical attributes. A symbolic solver wrapper was designed to provide an environment that encapsulates CASs and exposes their functionalities through symbolic services. Maple was chosen as computational engine in the initial implementation and it is loaded from the XML configuration file [31]. Simple examples of mathematical Web services were provided: integration and differentiation services, limits and series services, root-finding and solving systems of polynomials. Axiom was used to demonstrate the ability to incorporate different computational engines without changes.

Mathematical Services Description Language (MSDL [2]) was introduced to describe mathematical Web services so that these services can be discovered by clients. It implements a service description model that uses a decomposition of descriptors into multiple inter-linked entities: problem, algorithm, implementation, and realization. More recently, a MathBroker [28] implementation was based on a Web registry to publish and discover mathematical Web services. A usage example of the MathBroker was provided in [2].

MapleNET and WebMathematica are commercial counterparts to these initiatives. In the MapleNET [20] a server manages concurrent Maple instances launched to serve client requests for mathematical computations and display services, and facilitates additional services such as user authentication, logging information, and database access. Similarly, WebMathematica [38] offers access to Mathematica applications through a web browser.

**2.3. Grid services.** GridSolve [42], a component of one of the earliest Grid systems developed, the NetSolve, is a middleware between desktop systems equipped with simple APIs and the existing services supported by the Grid architecture—this API is available for the Mathematica.

The GENSS project [10] followed the ideas formulated in the MONET project. Its aim was to combine Grid computing and mathematical Web services using a common open agent-based framework. The research was focused on matchmaking techniques for advertisement and discovery of mathematical services, and design and implementation of an ontology for symbolic problems.

Another academic initiative, MathGridLink [35] proposed both the development and deployment of Mathematica computational services on the Grid and the usage of existing Grid services from within Mathematica; this initiative is continued by the SCORUM project [23].

Separately, Maple2g (Maple-to-Grid) described in [25] allows the connection between Maple and computational Grids based on the Globus Toolkit. The prototype consists of two parts: a Maple-dependent one, a library of new functions allowing to interact with the Grid, and a Globus-dependent part, a package of Java CoG classes. Maple2g allows the access to Grid services, the exposure of Maple facilities as Grid services, and the cooperation between Maple kernels over the Grid. SymGrid-Services generalizes the Maple2g development experiences to the level of CAS.

GridMathematica [37] was constructed as a commercial solution for dedicated clusters facilitating parallel computing within Mathematica. Another example of exposing CAS to the Grid is HPC-Grid for Maple [12]. It is a distributed computing package using Maple that allows users to distribute computations across the nodes of a network of workstations; it offers a message passing API as well as a set of high-level parallelization commands. Based on MapleNet and HPC-Grid, the recent Grid Computing Toolbox for Maple [21] allows to distribute computations across nodes of a network of workstations, a supercomputer or across the CPUs of a multiprocessor machine (in the same administrative domain), and offers an MPI-like message passing API as well as a set of high-level parallelization commands.

The recent GEMLCA [7] is a solution to deploy a legacy code application (including a computer algebra system) as a Grid service without modifying the code. The front-end, described in the WSDL, offers Grid services to deploy, query, submit, check the status of, and get the results back from computational jobs. In order to access a legacy code program, the user executes the Grid service client that creates a code instance with the help of a code factory, and the system submits the job.

**2.4. Overview.** Overall, it can be observed that using the above mentioned technical solutions, several CASs can be remotely accessible, but, almost all do not use a standard data model for interactions with CASs. Moreover, none of the above shortly described systems conforms to all three of the following basic requirements (as the SymGrid-services does):

(a) deploy symbolic services;
(b) access available services from within the symbolic computing system;
(c) couple different symbolic services into a coherent whole.

Furthermore, the pre-WRSF versions of Web and Grid middleware were used in the previous described projects, making the service discovery a difficult task.

**3. SymGrid.** The aim of the SCIEnce project (Symbolic Computation Infrastructure for Europe, `http://www.symbolic-computation.org`), funded in the frame of the European Commission Programme FP6, is to improve integration between CAS developers and application experts. The project includes developers from four major CASs: GAP [9], Maple [19], MuPAD [29] and KANT [15]. Its main objectives are to:

– develop versions of the CASs that can inter-communicate via a common standard service interface, based on domain-specific results produced by the OpenMath [24] and the MONET [22] projects as well as generic standards for Web and Grid services, such as the WSRF;
– develop common standards and middleware to allow production of Web or Grid-enabled symbolic computation systems;
– promote and ensure uptake of recent developments in programming languages, including automatic memory management, into a symbolic computation systems.

The research is primarily concerned with parallel, distributed, Web and Grid-based symbolic computations. The five year workplan includes the followings stages:

1. produce a portable framework that will allow symbolic computations to access Grid services, and allow symbolic components to be exploited as part of larger Grid service applications on a computational Grid (finalized stage);
2. develop resource brokers that will support the irregular workload and computation structures that are frequently found in symbolic computations (on-going stage);
3. implement a series of applications that will demonstrate the capabilities and limitations of Grid computing for symbolic computations (future stage).

In what follows we describe the portable framework, namely the SymGrid. It was designed and presented first in [11]. SymGrid allows multiple invocations of symbolic computing applications to interact via the Web or Grid and it is designed to support the specific needs of symbolic computations.

SymGrid comprises of two components: the SymGrid-Par to support the construction of high-performance applications on computational Grids, and the SymGrid-Services to manage Web and Grid services.

The SymGrid-Par middleware is used to orchestrate computational algebra components into a parallel application and allows symbolic computations to be executed as high-performance parallel computations on a computational Grid. SymGrid-Par components communicate using the Symbolic Computation Software Composability Protocol (developed in the frame of SCIEnce project), SCSCP [17], which in turn builds on OpenMath. SymGrid-Par provides an API for parallel heterogeneous symbolic components, which extends the Grid-GUM [39], and comprises in two generic interfaces:

*CAG interface:* Computational Algebra system to Grid middleware, that links CASs to the Grid-GUM;
*GCA interface:* Grid middleware to Computational Algebra system, that links the Grid-GUM to these systems.

The purpose of the CAG/GCA interfaces is to enable computational algebra systems to execute on computational Grids, e.g. on a loosely-coupled collection of Grid-enabled clusters. Details about SymGrid-Par are provided in [40]. Here, a GAP library has been build as demonstrator of usage of the SymGrid-Par.

The SymGrid-Services middleware is used to access, from computational algebra systems, Grid and Web services, and to access and compose the CASs deployed as Grid and Web services. It is based on the WSRF standard that ensures uniform access to Grid and Web services. A GAP library is available also for this SymGrid component. As in the case of the SymGrid-Par, the SymGrid-Services has two interfaces that will be detailed in the next sections:

*CAGS interface:* Computational Algebra system to the Grid and Web services, links CASs to external services;

*GCAS interface:* Web and Grid services for the Computational Algebra system, integrating these systems into a service-oriented architecture and allowing service composition.

While there are several parallel computer algebra systems suitable for either shared-memory or distributed memory parallel systems, work on Grid-based symbolic systems is still nascent. A review of current Grid-based systems for symbolic computation can be found in [26] and, in a short version, in the next section.

The main novelty of the SymGrid-Services consists of the fact that it is the only current middleware package that allows generic access to both Web and Grid symbolic and non-symbolic computing services, as well as their composition. The specific mathematical Web services like the ones defined by the MONET project [22], or standard Web services, are easily accessible, and the Grid services wrapping Kant, MuPAD, GAP and other computational algebra systems provided by the SymGrid service container can be called from inside a CAS.

A number of major new obstacles need to be overcome by the SymGrid in the near future. Amongst the most important future developments are mechanisms for adapting to dynamic changes in either computations or systems. This is especially important for symbolic computations, which may be highly irregular in terms of data structures and general computational demands, and which therefore present an interesting challenge to current and projected technologies for computational Grids in terms of their requirements for autonomic control.

SymGrid intends to go beyond current systems by developing a generic framework supporting heterogeneous Grid components derived from a critical set of complementary symbolic computing systems, by incorporating new and vital tools such as dynamic resource brokers and schedulers that can work both at a task and system level, and by creating a number of large new demonstrator applications.

## 4. CAGS interface of SymGrid-Services.

CAGS allows Computer Algebra Systems to leverage the computing capabilities offered by external Grid or Web services. Details about its implementation are provided in [3]. In this paper we present shortly its main functionality.

### 4.1. Description of the interface's implementation.

A CAS user must be able to: discover services, connect to remote services, call remote operations, run jobs, and transfer files in a seamless fashion. CAGS provides this functionality. The SymGrid-Services's CAGS interface consists of three Java classes (Figure 4.1):

**SGServices** provides three kinds of operations—retrieval of a list of services registered in a certain Web or Grid services registry; retrieval of signatures for the exposed operations of a service; and calling remote operations.

**SGProxyCert** handles issues arising from the need to support 'single sign-on' for users of the Grid and delegation of credentials: namely the creation and destruction of proxy certificates, retrieval of information about the owner of a certificate and about the lifetime of a proxy certificate.

**SGUtils** provides additional functionality for explicit file transfer, file deletion and remote job execution.

To access the functionality provided by these three classes it is necessary to create new class instances. Generally, however, CASs do not offer such functionality by default and therefore it is necessary to run the supplied Java classes in a child process created by the CAS. This process then communicates with the CAS using standard input/output streams to pass parameters and return values (Figure 4.2). The CAS will create a child processes by launching a script that starts a Java class called RunManager. The main method of this class should be called with an array of string type arguments:

$$\texttt{java RunManager}\ arg0\ arg1\ arg2\ \ldots\ argN$$

where the *arg0* represents the Java class to load, *arg1* is the name of the method to invoke, and remaining arguments are passed to the method. RunManager is a generic command that exploits Java reflection capabilities to allow the execution of any class.

### 4.2. Usage scenario.

The primary functionality of CAGS lies in obtaining a list of Grid or Web services registered at a certain URL; obtaining the signatures of those operations that are exposed by a certain Grid or Web service; calling an operation and retrieving the result of an operation call. Secondary functionality includes file transfer and job submission, and managing utilities for proxy certificates.

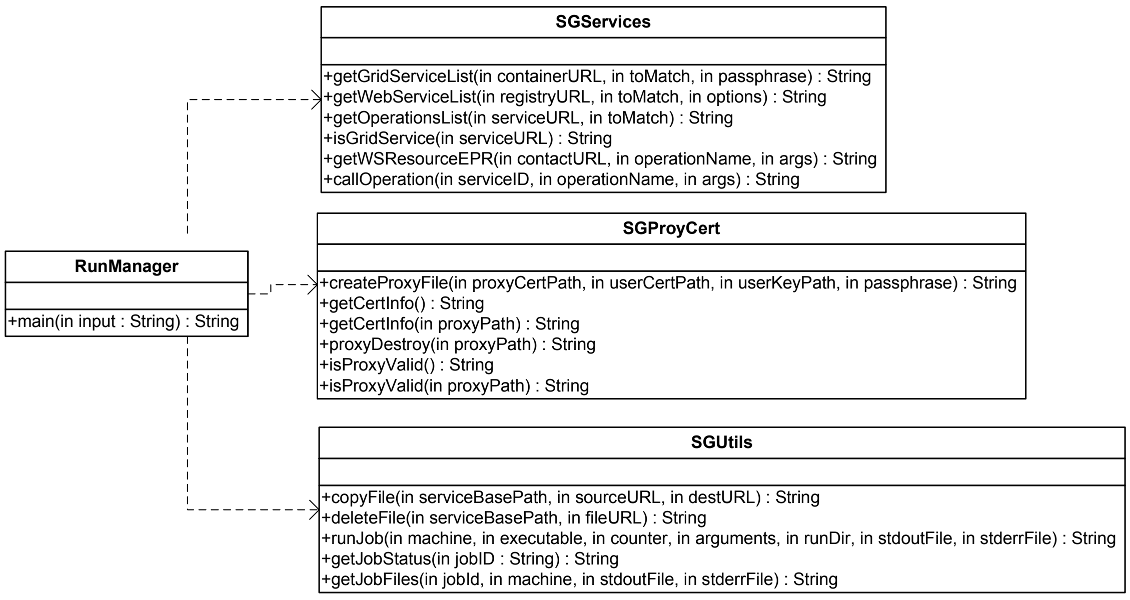
SGServices
+getGridServiceList(in containerURL, in toMatch, in passphrase) : String
+getWebServiceList(in registryURL, in toMatch, in options) : String
+getOperationsList(in serviceURL, in toMatch) : String
+isGridService(in serviceURL) : String
+getWSResourceEPR(in contactURL, in operationName, in args) : String
+callOperation(in serviceID, in operationName, in args) : String
SGProyCert
+createProxyFile(in proxyCertPath, in userCertPath, in userKeyPath, in passphrase) : String
+getCertInfo() : String
+getCertInfo(in proxyPath) : String
+proxyDestroy(in proxyPath) : String
+isProxyValid() : String
+isProxyValid(in proxyPath) : String
RunManager
+main(in input : String) : String
SGUtils
+copyFile(in serviceBasePath, in sourceURL, in destURL) : String
+deleteFile(in serviceBasePath, in fileURL) : String
+runJob(in machine, in executable, in counter, in arguments, in runDir, in stdoutFile, in stderrFile) : String
+getJobStatus(in jobID : String) : String
+getJobFiles(in jobId, in machine, in stdoutFile, in stderrFile) : String

Fig. 4.1. *CAGS structure*

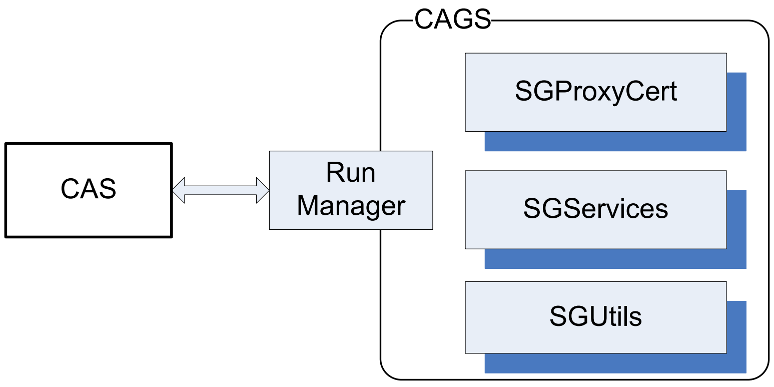

Fig. 4.2. *Interactions CAS—CAGS*

A typical scenario begins with the discovery of a service by consulting a service registry URL:

```
start scenario(registry_URL)
 if(is_Web_service_registry(registry_URL))
  service_list:=get_Web_service_list(registry_URL,toMatch,options)
 else
  service_list:=get_Grid_service_list(registry_URL,toMatch)
 endif
 service:=select_service(service_list)
 operation_list:=get_operation_list(service,toMatch)
 operation:=select_operation(operation_list)
 [create_proxy_certificate();]
 result:=call_operation(service,operation,parameters)
end scenario
```

Here, the *registry_URL* parameter is a valid URL of a UDDI registry or a Globus container. The *toMatch* parameter is a selection string that must be a substring of the service name in the *get_Web_service_list*/*get_Grid_service_list* combined with a substring of the operation name in the *get_operation_list*. The selection functions *select_service*/*select_operation* are user-defined functions that can be used to select the desired service/operation.

Note that this scenario assumes that the user only knows the *registry_URL*. If the user already knows, for instance, the service URL and the signature of the operation, the unnecessary steps can be omitted.

One of the initial SymGrid-Services targets is the GAP [9] computational algebra system. A demonstrator library of GAP functions was built to allow a GAP user to access the CAGS functionality without needing to know the details of the implementation. Here we ahev developed a single function in the GAP library for each method in the CAGS interface. The general pattern used to wrap the CAGS functionality is:

```
directoryName := DirectoryCurrent();
fileName:=Filename(directoryName,"script.sh");
ListMethods := function(arg2\dots argN)
  local jm, response;
  jm:= InputOutputLocalProcess(DirectoryCurrent(),fileName,[]);
  #send handler method: class, method, no. args, args, end signal
  WriteLine(jm, "java_class_to_call"); WriteLine(jm, "method_name");
  WriteLine(jm, "nr_of_parameters");
  WriteLine(jm, arg2);  \dots   WriteLine(jm, argN);
  WriteLine(jm, "quit");
  #retrieve response from the process
  repeat
    response := ReadLine(jm);
    if (response <> fail) Print(response); fi;
  until (response = fail);
end;
```

Since Java cannot be invoked directly from a GAP program, the solution is to invoke instead a shell script that starts a Java program using the *InputOutputLocalProcess* function of the GAP. The needed parameters are passed to the script *script.sh*. The variable *jm* is the handle to the running process. It can be used as an argument to the *WriteLine* function to pass the arguments. Arguments provided to the shell script will be mapped to the parameters required to call the RunManager.

**4.3. Usage examples.** We illustrate below how the CAGS can be used from the command line by invoking several methods in the public interface of the tool with the help of the RunManager utility.

Three categories of services have been used for the initial tests:

1. general Web services (see the paper [3]);
2. domain-specific symbolic Web services such as these provided by MONET [22] and GENSS [10];
3. simple test Grid services, that we have deployed on a single cluster and wraped publicly available CASs: CoCoA, Fermat, GAP, Kant, Macaulay, MuPAD, PARI, Singular and Yacas. These test services were deployed in a Globus container (available, for example at http://matrix.grid.info.uvt.ro:8082/).

A registry can be interrogated to obtain the list of services registered to that registry. For example, the command:

```
java science.run.RunManager science.clients.wrappers.SGServices
getWebServiceList "http://matrix.grid.info.uvt.ro:8082/wsrf/services/"
"Service" "caseSensitiveMatch"
```

produces the result:

```
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/CoCoAService
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/FermatService
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/GAPService
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/KANTService
...
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/YACASService
```

This is a list of service URLs representing all services in the registry whose names include the substring "Service".

Once the address of a service is known, CAGS can supply the signatures of the operations exposed by the service. Based on the list of the methods exposed, the user can then discover all details that are needed to call a remote operation. In the case of the service wrapping Fermat that is deployed on the SymGrid testbed, as the result of the command:

```
java science.run.RunManager science.clients.wrappers.SGServices
getOperationsList "http://matrix.grid.info.uvt.ro:8082/wsrf/services/
science/FermatService" ""
```

the following list of operations can be obtained:

```
    string Bin (string)
    string Prime (string)
    ...
```

Remote operation invocation is one of the main reasons for using CAGS. To call the Fermat service operation that calculates the greatest common divisor of 96 and 80, one should use the following:

```
java science.run.RunManager
science.clients.wrappers.SGServices callOperation
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/
FermatService GCD "96,80"
```

obtaining as result *string:16*. Note that the GCAS version of the services described in the next section accepts only two OpenMath objects instead of the two integer values currently allowed by the service used in this example.

To show how GAP can use CAGS to interact with external services, we have built an example in which GAP calls an external service. The external service is a YACAS instance, that easily interacts with OpenMath objects. The first step in the example is to list all the services from a Globus container that can be matched using the string "YACAS". From the list of services that are obtained, we choose the wrapping service, and ask for the list of operations supported by that service that relate to OpenMath (OM). The final step of the example launches a call from GAP to the YACAS service. The result of the call is displayed by GAP on the console:

```
gap> SG_CreateProxy("path_proxy","","","pswd");
gap> gridServList := SG_GridServiceList(
"http://matrix.grid.info.uvt.ro:8082/wsrf/services/","YACAS");
 http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/
 YACASService
gap> operationList := SG_OperationList(gridServiceList[1], "OM");
 string YACAS_OMDef (string)
 string YACAS_OMForm (string)
 string YACAS_OMRead (string)
gap> SG_CallOperation(gridServiceList[1],operationList[2], "25");
 string:<OMOBJ> <OMI>25</OMI> </OMOBJ>
```

## 5. GCAS interface of SymGrid-Services.

This section introduces the CAS Server architecture as the main component of the GCAS interface of the SymGrid-Services. The CAS Server was presented first in the paper [5], and the description is extended by this paper.

The main functionality of the CAS Server is to enable virtually any CAS to have its own functions remotely accessible. Several CASs can be exposed through the same CAS Server at the same time. Additionally to the services that where exposed in the testing phase of the CAGS, the CAS Server allows the limitation of the number of functions exposed for a CAS and the interrogation of the CAS Server about the functions exposed. The following subsection is an overview of the CAS Server architecture.

Let us stress that integration of legacy software in service oriented architectures must consider three major issues: data encoding, exposing technology, and wrapping the legacy system. These issues are treated in different subsections that follow. Implementation issues are provided at the end of the section.

### 5.1. Architecture of CAS Server.

GCAS aims to expose CASs functionality as services. The interaction between these services can lead to computing efficiency gains when solving complex symbolic problems. Unfortunately, the different data encoding and algorithms used within distinct CASs to solve the same problem hinders the integration process.

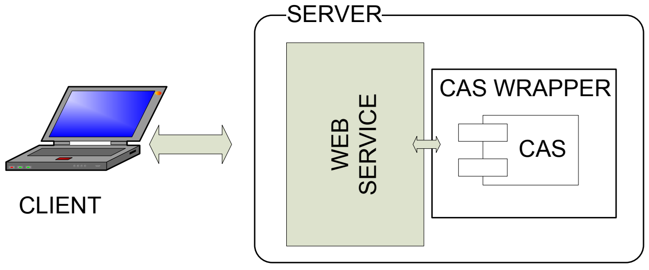

Fig. 5.1. *CAS-wrapper service architecture.*

We assume that it is the client's responsibility to choose the right functions needed for a computation, just as she would have had the CAS installed on the local machine. Alternatives like predefined workflows or patterns for service composition are subject of later developments of the SymGrid-Services' GCAS component. Patterns description were described first in [11] and were recently implemented in the composer component of the GCAS described in [6] and shortly in Section 6.

The first and most important issue to deal with in the component design is to find means to expose CAS functions in order to make them available through service invocation. Other issues that must be taken care of are related to user assistance tools and security.

A significant number of existing CASs is not conceived to be remotely accessible. Most of them allow interaction with the external world through line commands. Only few of them have special facilities like socket connections or multi-threading. In general, the redesign of these systems to allow modern communication with outside world cannot be achieved easily. Due to these reasons, we have considered the wrapper approach for integrating CASs.

Wrapping legacy software and exposing their functionality using service technologies involves the creation of a three level architecture at the server side, as shown in Figure 5.1. The server has the role of receiving the calls from the client, resolving them using underlying legacy software and returning the result(s) of the computation to the client.

The proposed architecture intends to expose functions implemented by several CASs in a secure manner. To achieve this goal the simple wrapper architecture is augmented with several features. Setting up the architecture on a server machine requires that all necessary software tools and components are available on the machine, namely the service tools and the CASs. A simple tool should allow the administrator of the machine to register into a CAS registry the CAS functions that he wants to expose. Every function exposed by a CAS will have an entry in the registry. Thus, a method that does not appear in this registry is considered inaccessible to remote invocations (for example the system function available in different CASs).

The remote clients of the system may interrogate the system in order to find out the available functions and their signatures. The signatures of the functions should not differ in any way from the original signature of the function as provided by a CAS. The remote invocation of a function should be reduced to the invocation of a remote

*execute(CAS_ID,call_object)*

where the *CAS_ID* is the CAS unique identifier and the *call_object* represents an OpenMath object, as described in the following subsection.

Additionally, several other related operations should be available: find the list of the CASs that are installed on the CAS Server machine or find the list of the available functions that were exposed.

**5.2. Parameter encoding level.** One important aspect of interactions between the client and the legacy system is the model of data exchange. The data model used by the client must be mapped to the internal data model of the legacy software. Currently, the available CASs use a variety of encoding standards, from plain text to XML structured documents. Due to the benefits involved in representation and manipulation of the data in a structured manner, the XML standard was adopted as the natural choice for machine to machine interaction. Representation of mathematical objects was achieved by MathML and OpenMath standards. However, while the former is well suited for representing mathematical objects in Web browsers, the latter is more appropriate for describing mathematical objects with semantic context.

Efforts to enable parsing of OpenMath objects are under way for several CASs. A standard encoding for CAS invocations is described in [17].

Since most currently existing CASs do not support OpenMath, for the immediate implementation purposes, we considered an intermediate approach: a function *procedure(Arg1,Arg2)* is translated into a corresponding OpenMath object as the one shown in the following:

```
<OMOBJ>
    <OMA>
        <OMS cd="casall1" name="procedure_call"/>
        <OMSTR>procedure</OMSTR>
        <OMOBJ>Arg1</OMOBJ>
        <OMOBJ>Arg2</OMOBJ>
    </OMA>
<OMOBJ>
```

The parser will use the information encapsulated in the OpenMath object to create the appropriate CAS command.

The internal OMOBJ objects must be OpenMath encoding of the corresponding mathematical objects, either atoms or compound. For the case of a CAS that does not parse OpenMath objects it is possible to encapsulate the generic representation of *Arg1, Arg2* using OMSTR atoms; the CAS is then responsible to convert the encapsulated string to the CAS object.

**5.3. Exposing technology.** In what follows we argue that exposing the full functionality of the CASs is difficult due to the high number of functions that CASs implement. Another issue is security since certain functions exposed could represent a security gap.

The first approach to expose functions of a CAS that one might consider is a one-to-one approach. This means that for every function of a CAS a corresponding operation of a service should be implemented. The experience gained by constructing the CAGS tool (see [3] for more details) leads us to the conclusion that this approach is not feasible for a large number of operations. A service with many operations exposed makes impossible dynamic creation and invocation of the service. Additionally, the WSDL 2.0 standard explicitly forbids that operations with the same name exist within the same service definition, while in a CAS functions from different libraries can have the same name.

The second approach (considered also in the GENSS platform) is to implement a generic operation: *execute(function_name, parameters)*. In this call the *function_name* represents the name of the CAS function and the *parameters* represent encoding for the parameters of the function. In this case, a Java method with the name *function_name* can be dynamically invoked using the reflection mechanisms. Afterwards, this method has to invoke the function exposed by the CAS. Hoever, also this solution, has some drawbacks. Deploying such services into a service container is not efficient and obtaining the list of the exposed functions and assuring access to them on a per user basis is not trivial.

The solution that we have considered for the GCAS uses the second approach as a starting point. We have created a registry mechanism that allows the administrator of the server to register CAS functions into a database. The general execution schema associated with this approach is composed from several steps:

1. The client invokes an *execute(CAS_ID,call_object)* operation on the service.
2. The server verifies that the CAS identified by *CAS_ID* is available on the server and that the function encoded in the *call_object* is available.
3. The server returns a unique job identifier that identifies the job and starts the execution.
4. At a later moment the client will use the job identifier to retrieve information about the status of the job and the results of the computation.

As mentioned above, the interaction between the client and the server is carried out in an asynchronous manner. Additional functionality is available using this approach, such as: the list of the CASs exposed, the list of functions of a certain CAS that are exposed, the signature of functions, and so on.

**5.4. Wrapper interaction with the legacy system.** CASs were meant to run on a single machine, or sometimes on clusters, in order to solve very specific computational tasks. Usually the interaction with these systems involves a command line interface. According to [32] software can be encapsulated at several levels: job level, transaction level, program level, module level and procedure level. The way that the wrapper interacts with the legacy software component depends on the native technology of the legacy component. The wrapper may use TCP/IP sockets, redirecting of the I/O streams for a process, input and output files or it may have to use JNI encapsulation.

The communication model we had to use depends on the CAS that we want to communicate with. As a regular basis, we communicate with the CASs by redirecting I/O streams in a program level encapsulation style. For the GAP and the KANT we have used the RunManager component shortly described in the previous section and detailed in [3]. The interaction with Maple was implemented using the Java API that it provides.

The WS-GRAM service offered by Globus Toolkit enables to run command line batch jobs. As an alternative to the RunManager component we implemented the interaction to the CAS by making appropriate calls to the WS-GRAM service. An efficiency comparison of the two approaches is presented in [5] proving that RunManager approach is much faster.

**5.5. Implementation details.** The CAS server that we implemented exposes two main categories of service operations. The first category includes operations that offer information about the CASs exposed on that server and the functions that the client is able to call. The second category refers to the generic operation that allows remote invocation of CAS functions.

The first operation category includes:

*getCASList(),*
*getFunctionList(CAS_ID),*
*getNrFunction(CAS_ID),*
*getFunctionsByIndex(CAS_ID, startIndex, endIndex),*
*getFunctionsMatch(CAS_ID, stringToMatch).*

These functions offer functionality for the client to discover the CASs installed on the server machine, and, for the exposed CASs, the functions being exposed. The parameter *CAS_ID* uniquely identifies a CAS system. The identifier of CASs can be obtained by calling the *getCASList()* function. Since the number of functions exposed on the CAS Server can be large, we created convenient operations to filter the list of function signatures when issuing a *getFunctionList(CAS_ID)* call. For example, we can display functions with a certain name or we can display a limited number of functions that match a certain criteria.

The actual execution of a command can be achieved by calling a generic operation *execute()*. The operation *String execute(CAS_ID, call_object)* returns a unique identifier for the submitted job. This identifier will be used to retrieve information about the status and the result of the job.

The registry of the exposed functions can be populated by the system administrator using a registry editing tool that we implemented. CAS related information stored into registry includes, but is not restricted to, the identifier of the CAS, the path where the CAS is installed and a brief description of the CAS. Function related information includes the function name, the signature and a brief description. If the CAS allows socket connections, it can reside on another machine and the specific path can include the shell command that allows the connection to that machine that runs the CAS.

GCAS's main component, CAS Server, is implemented using Java 5 SDK with supporting tools and software from Globus Toolkit 4, Apache Axis, Apache Tomcat 5 and RIACA's OpenMath Java API. The generic services implemented in the CAS Server architecture invoke CAS functions that are applied to OpenMath objects.

**5.6. Grid services benefits.** Several benefits already stated bellow and several others mentioned in [27] motivate the migration to the WSRF standards and exposing CAS functionality using Grid service technology.

SymGrid-Services complies with the WSRF standard for Grid services. While Grid services have different goals from pure web services (sharing computing power and resources like disk storage databases and software applications, versus sharing information), a Grid service is basically a Web service with some additions: stateful services, service instantiation, named service instances, two-level naming scheme, a base set of service capabilities, and lifetime management. These supplementary capabilities allow improved user interaction with remote symbolic computing services: previous computations performed by a service can be stored as service state, personalized services are possible due to the instantiation facilities, services can be easily modified due to the naming schemes, standard search facilities can be implemented due to the standard service data elements, and resource management can be easily done due to the transient character of the services. Complying the WSRF standard imposes not only that the interface is offered to the user, but that it preserves the original specified behavior.

With non standard Web or Grid services, the architect of the service is the one that designs the access interface to data. With the WSRF services, most of those access interfaces are already presumed and the

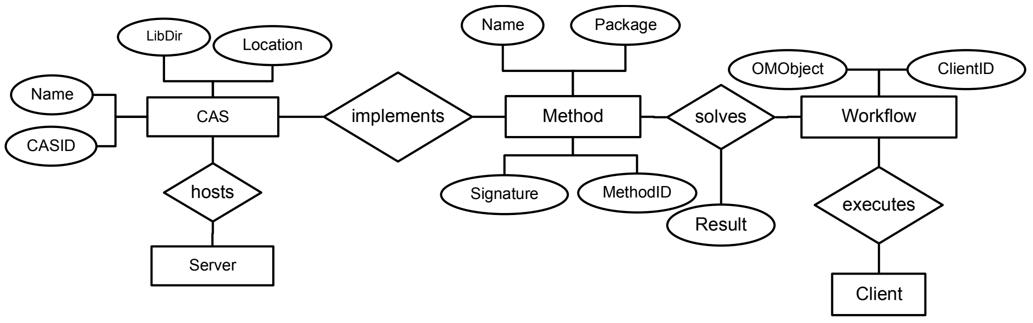

FIG. 5.2. *Data model for creating CAS services.*

developer must only provide the functionality behind the scenes. Migrating from non standard approach to a standard WSRF solution is not covered so far, to the best of our knowledge, by any methodology.

Grid services are the solution for sharing resources in a uniform way. Interfaces are described using WSDL language and XSD technologies and the resources are described using XSD data types. The structure of a Grid Resource is described, in XML terms, as a root document element that maps to the resource and several top level complex elements representing the Resource Properties (RP). Implementations of the WSRF standards, e.g. by Globus Toolkit 4, map XML documents that describe resources to in-memory representations as JavaBeans classes. The access to attributes stored in resources, as specified by the WSRF, is achieved mainly using standardized Web service operations. The Grid resource mechanism is not intended to state how data is stored at server level; the most common mechanism to store data remains backend databases.

Databases normalization usually has as a basis the conceptual Entity-Relation (ER) diagram. A methodology that transforms the ER into a blueprint of a Grid service that supports the access operations to data is an important step in the widely adoption of Grid services. Several papers [1, 16] cover the process of transforming ER diagrams to XML normalized documents. While the algorithms presented in these papers seem to offer a satisfactory solution for converting ER to XML documents, these solutions cannot be used as they are because they use attributed elements and reference attributes that are not allowed by the WSRF standard and they cannot be used in the context of Grid Services to create JavaBeans.

Several changes may be applied on existing implementation of non WSRF Web services to create corresponding Grid services. However, this approach would not represent the WSRF implementation in the semantic sense because standard intended operations functionality is replaced by non standard ones. To leverage the full power of Grid Services a simple translation would not be enough. Thus, having a set of guidelines in the Grid Service process would improve the quality of the design. The experience gained in designing and implementing Grid symbolic services led to identifying several design rules that were highlighted in [18].

Starting with the ER that was the conceptual basis for the Web service model we have presented in [5], we expose in Figure 5.2 the process that led to the current Grid services model. The structure of the Grid services we created uses the basic principles for designing Grid Resources defined in [18] and a variation of the algorithm presented in [16].

For the simplicity reasons we assume that the user only wants to gather information about the CASs exposed by the system and the functions he/she is allowed to use, to submit a request for a service, and to get the result of a computation. Several attributes of several entities must be available for the user. Moreover, entities such as the *CAS* and the *Methods* hold information that changes more often and the user should not have the right to modify them. For this reason, the Factory service that we have implemented exposes these attributes as RPs of a single Resource, not as multiple Resources, as the ER states. The Factory service is responsible for creating an additional Resource that holds the details regarding the result of the computation.

As a result of this analysis we came to the conclusion that the GCAS interface must be composed by two Grid services, i. e. a Factory stateful service and a Resource. The Factory stateful service has two RPs, the *CAS* and the *Method* that hold the *(casID, name)*, and the *(methodID, casID, name, signature, package)* respectively. The Resource has as the RP the result of the computation.

The CAS Servers are implemented as Grid services deployed in GT4 containers. A CAS Server provides access to CASs (e.g. GAP, Kant) installed on the same machine. The Resource associated with the service

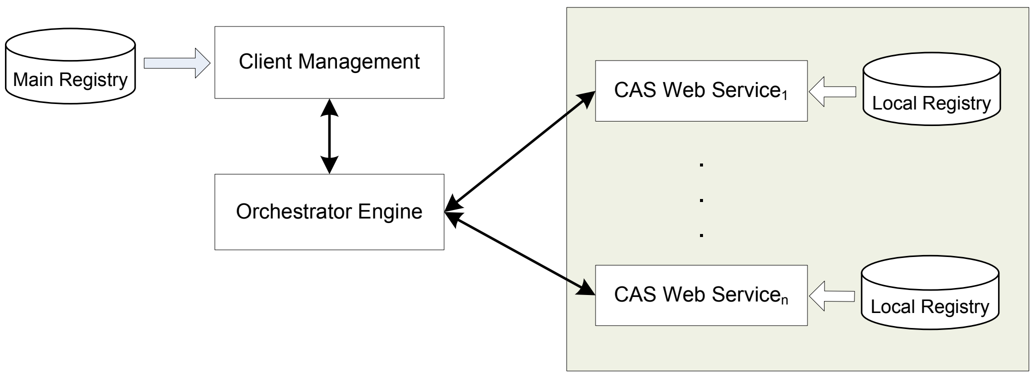

Fig. 6.1. *Server side architecture*

keeps track of the exposed CASs and functions. The Grid service's Resource is stored persistently in a backbone database rising problems of mapping these resources to database tables. For each received computation request the server creates a new WS-Resources to handle the request. The results of the computations can also be retrieved using operations specified by the WSRF standard. This is rather important advantage of using Grid services for implementing CAS servers as they offer a standard interface for accessing information about the exposed CASs.

**6. Composing the SymGrid's services.** A service oriented architecture relays on loosely coupled software components exposed as services. The service composition is usually done based on workflows descriptions and engines.

The static composition in workflows is achieved at the design time by specifying all the details of the composition elements, i. e. services and binding details for these services. A special type of static composition, namely the workflows with dynamic bindings, offers the ability to specify the actual addresses of the services at runtime.

The SymGrid' set of publicly available Web and Grid services intends to be highly dynamic: some of the services are occasionally available, others are vanishing, new and potentially better services become available, while for some of them the interface changes. In this situation the static composition approach fails to offer a robust solution. The alternative is to use dynamic workflows, generated at the runtime using latest information available on the services to be composed. In this case special mechanisms and techniques to identify and invoke the right Web services are required.

An in-between approach, namely semi-dynamic composition, that can be used when several characteristics of the services involved in the composition are known, was considered for SymGrid-Services and reported in [4]. The system does not have to have any a'priori knowledge about addresses of Web services needed by the composition workflow thus the binding is dynamic. Known is the structure of the operations exposed by the services.

In what follows we outline the proposed solutions. Details can be found in the following recent papers [4, 6, 18].

**6.1. GCAS' composer architecture.** The composer obtains the execution of mathematical based workflows with the aid of several software components. At the server side level, the main components needed to carry out the workflow execution and to manage related issues include a client manager component, an engine needed to execute the workflow, and several CAS servers that expose CAS functionality as services (Figure 6.1).

A typical scenario implies that the users specify within a CAS the interdependent tasks that compose the workflow. To execute it, the workflow is submitted to a service located at a previously known URL address. An assumption is that the user is indicating the CAS system for every task.

Several steps are needed to transform the workflow described within the CAS to a format that complies with a standard orchestration language. The workflow that results at the client side is not complete because the client is not, and should not be aware of the URL addresses of the services that will compute the subtasks. As a result, the client component sends an incomplete workflow to the client manager component.

One of the most important responsibilities of the client manager component is to obtain addresses of the CAS servers that will execute the task by consulting the main registry and to supply this information to the execution engine. For every CAS server there is a corresponding local registry that contains information about the CAS systems exposed and the functionalities supported by them. The main registry duplicates information contained into local registries that are spread on different sites of the distributed infrastructure and it is used to find appropriate CAS servers to solve the atomic tasks of the workflow.

Another responsibility of the client manager component is to send back to the client a workflow identifier that will be used later by the client to retrieve the result of the whole computation.

The management of the workflow execution must be carried out by a workflow engine located at the server side. The client management component is the one that invokes the operations provided by server machines that expose CAS functionality as Web services. It must be emphasized that all client-server interactions as well as servers to server component interactions are encoded in XML format. The description of mathematical expressions uses the standard OpenMath.

Details about the composer implementation are provided in [4].

**6.2. Using standard workflow patterns.** The suitability of the BPEL workflow description language for the dynamic composition of SymGrid services was investigated recently and results are reported in [6]. General workflow patterns are helping the CAS user to describe the relationships and sequence of service calls; the resulted description is deployed and executed by SymGrid-Services components implemented using Java 5 SDK relaying on the ActiveBPEL workflow engine and the PostgreSQL database servers.

Application specialists need not be aware of all the details required for the complete specification of the whole workflow using a specialized language. Instead, they only need to be able to combine several workflow patterns in order to describe a high level solution to their problem. The user-specified workflow can be automatically translated into a specialized workflow language, deployed and executed by a workflow management server. The blueprint of the client component that we have implemented can be used to enable this functionality within every CAS with a minimal effort. GAP system is used, again as representative CAS in the demonstration, to combine workflow patterns and execute workflows that use the functionality of several other CASs installed on remote machines.

The description of the problem specified at the client level is submitted to a server that will manage the rest of the process. At the client side, the workflow specified within the CAS is encoded using the XML language similar to BPEL that was described in [18]. The main reason for using an XML intermediate language instead of a complete BPEL description is the significantly larger size of the completely specified BPEL workflow. The drawback of this approach is the additional server load needed to convert the XML format to the BPEL format. The client manager component is responsible not only for receiving new workflows and for providing clients access to the result of their computation, but also for translating the XML workflow representation received from the client to the corresponding BPEL workflow format, to deploy the workflow into the ActiveBPEL engine and to launch the execution of the process.

**6.3. A simple example.** SymGrid-Services' client is currently able to specify workflows by composing standard workflow patterns. A very simple example is provided in [4, 6] to demonstrate the system functionality: compute the value of the Gcd(Bernoulli(1000), Bernoulli(1200)) using remote machines and two different CASs, GAP and KANT. The Gcd() is computed using the KANT system by combining the Bernoulli values obtained from two separate instances of GAP. The system allows the execution of workflows that are not bound to a two level invocation scheme. The corresponding GAP code that would allow obtaining the same result as the previous system is:

```
startWorkflow();
 startSequence();
   startParallel();
     v1:=invoke("KANT",Bernoulli(1000));
     v2:=invoke("KANT",Bernoulli(2000));
   endParallel();
   invoke("GAP",gcd(v1,v2));
 endSequence();
endWorkflow();
readOutput(processHandler);
```

The above code is translated at the client level into the simplified BPEL like format and it is submitted to a server. The simplified BPEL format is the following:

```
<workflow xmlns = "http://ieat.ro" >
    <sequence>
        <parallel>
            <invoke invokeID = "invoke_0">
                <casid>KANT</casid>
                <call>Bernoulli(1000)</call>
            </invoke>
            <invoke invokeID = "invoke_1">
                <casid>KANT</casid>
                <call>Bernoulli(2000)</call>
            </invoke>
        </parallel>
        <invoke invokeID = "invoke_2">
            <casid>GAP</casid>
            <call>gcd($invoke_0,$invoke_1)</call>
        </invoke>
    </sequence>
</workflow>
```

The server will translate this code into a regular BPEL workflow format (over 300 lines for the above described example) and will manage the execution.

At a later time, the user may access the computed result based on the identifier that it is received when submitting the workflow. More complex examples are provided in [6, 18]. The GAP library is presented in [18].

**7. Conclusions.** A service-oriented framework, SymGrid-Services, was introduced to manage mathematical services. One of its main components consists of several CAS servers representing symbolic computing services. The services are wrapping legacy codes. The novelty of the wrapping approach, compared with similar ones that were identified, is the fact that is based on current standards for Web and Grid services and the only non-standard technique is given by the usage of a software and function register that is proved to be more efficient than using the standard approach of a special service call. The novelty of the service access component consists in the fact that it allows the seamless access from inside of computer algebra systems to both Web and Grid services. Moreover, the symbolic computing services can be combined into complex application using standard workflow patterns. SymGrid-Services will be further developed in the near future to include specific dynamic resource brokers and schedulers. Performance tests are only at an infancy stage and a number of large new demonstrator applications need to be provided soon.

**Acknowledgments.** This research is partially supported by European Framework 6 grant RII3-CT-2005-026133 SCIEnce: Symbolic Computation Infrastructure for Europe.

## REFERENCES

[1] M. Arenas and L. Libkin, *A normal form for XML documents*, ACM Transactions on Database Systems, Vol. 29 (1), ACM New York (2004), pp. 195–232.

[2] R. Baraka, O. Caprotti and W. Schreiner, *A Web registry for publishing and discovering mathematical services*, in Procs. EEE-05 (2005), 190-193.

[3] A. Cârstea, M. Frîncu, G. Macariu, D. Petcu and K. Hammond, *Generic access to Web and Grid-based symbolic computing services: the SymGrid-services framework*, in Procs. ISPDC 2007, IEEE CS Press (2007), pp. 143–150.

[4] A. Cârstea, G. Macariu, M. Frîncu and D. Petcu, *Composing Web-based mathematical services*, in Procs. SYNASC 2007, IEEE Computer Press (2007), pp. 327–334.

[5] A. Cârstea, M. Frîncu, A. Konovalov, G. Macariu and D. Petcu, *On service-oriented symbolic computing*, in Procs. PPAM 2007, LNCS 4967, Springer (2007), in print.

[6] A. Cârstea, G. Macariu, D. Petcu and A. Konovalov, *Pattern based composition of Web services for symbolic computations*, in Procs. ICCS 2008, LNCS, Springer (2008), in print.

[7] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S.Winter and P. Kacsuk, *GEMLCA: running legacy code applications as Grid services*, Journal of Grid Computing Vol. 3. Springer Science (2005), pp. 75–90.

[8] J. Denemark, A. Kulshrestha and G. Allen, *Deploying legacy applications on Grids*, in Procs. 13th Annual Mardi Gras Conference, Frontiers of Grid Applications and Technologies (2005), pp. 29–34.

[9] GAP Group, *Groups, Algorithms& Programming*, `http://www.gap-system.org`

[10] GENSS Consortium, *Grid-enabled numerical and symbolic services*, (2005), `http://genss.cs.bath.ac.uk/`

[11] K. Hammond, A. Al Zain, G. Cooperman, D. Petcu and P. Trinder, *SymGrid: a framework for symbolic computation on the Grid*, LNCS 4641 (2007), pp. 447–456.

[12] MapleConnect, *HPC-Grid for Maple*, 2006, `http://www.hpcgrid.com`

[13] Y. Huang, I. Taylor, D.W. Walker and R. Davies, *Wrapping legacy codes for Grid-based applications*, in Procs. IPDPS'03, IEEE Computer Society (2003), pp. 139–147.

[14] Institute for Computational Mathematics, *IAMC: Internet accessible mathematical computation*, `http://icm.mcs.kent.edu/research/iamc.html`

[15] KANT Group, *KANT/KASH—Computational algebraic number theory*, `http://www.math.tu-berlin.de/~kant/kash.html`

[16] C. Kleiner and U.W. Lipeck, *Automatic generation of XML DTDs from conceptual database schemas*, in Tagungsband der GI/OCG-Jahrestagung, Kurt Bauknecht et. al. (Eds.) (2001), pp. 396–405.

[17] A. Konovalov and S. Linton, *Symbolic Computation Software Composability Protocol Specification*, CIRCA preprint 2007/5, University of St Andrews, `http://www-circa.mcs.st-and.ac.uk/preprints.html`

[18] G. Macariu, A. Cârstea, M. Frîncu and D. Petcu, *Towards a Grid oriented architecture for symbolic computing*, submitted to ISPDC'08 (2008).

[19] MapleSoft, *Maple*, `http://www.maplesoft.com`

[20] MapleSoft, *MapleNet*, `http://www.maplesoft.com/maplenet/`

[21] MapleSoft, *Grid Computing Toolbox* (2008), `http://www.maplesoft.com/products/toolboxes/GridComputing/`

[22] Monet Consortium, *MONET: Mathematics on the net* (2004), `http://monet.nag.co.uk`

[23] M. Naifer, A. Kasem and T. Ida, *A system of Web services for symbolic computation*, in Procs. Asian Workshop on Foundation of Software, Xiamen, (2006), `http://www2.score.cs.tsukuba.ac.jp/publications/2006-1/publications/`

[24] OpenMath Society, *OpenMath*, `http://www.openmath.org/`

[25] D. Petcu, D. Dubu and M. Paprzycki, *Extending Maple to the Grid: design and implementation*, in Procs. ISPDC'04, J.Morrison et al (eds.), IEEE Computer Press (2004), pp. 209–216.

[26] D. Petcu, D. Tepeneu, M. Paprzycki and T. Ida, *Symbolic computations on Grids*, in Engineering the Grid, B. Di Martino, J. Dongarra, A. Hoisie, L. T. Yang, H. Zima (eds.) American Scientific Publishers (2006), pp. 91–107.

[27] D. Petcu, *Between Web and Grid-based mathematical services*, in Procs. ICCGI'06, P. Dini, C. Popoviciu, C. Dini, Gunter Van de Velde, E. Borcoci (eds.), IEEE Computer Society Press (2006), pp. 41–47.

[28] W. Schreiner, *Brokering mathematical services in the global network*, in Procs. IIWAS 2003, G. Kotsis, S. Bressan and I. Ibrahim (eds.), Austrian Computer Society, vol. 170 (2003).

[29] SciFace, *MuPAD*, `http://www.mupad.de`

[30] M. Senger, P. Rice and T. Oinn, *Soaplab—a unified sesame door to analysis tools*, in Procs. UK e-Science, All Hands Meeting 2003, Cox, S. J. (ed.) (2003), pp. 509–513.

[31] E. Smirnova, C.M. So and S.M. Watt, *Providing mathematical Web services using Maple in the MONET architecture*, in Procs. MONET Workshop (2004), `http://monet.nag.co.uk/cocoon/monet/proceedings/`

[32] H. M. Sneed, *Encapsulation of legacy software: a technique for reusing legacy software components*, Annals of Software Engineering, Springer (2000), pp. 293–313.

[33] A. Solomon and C. A. Struble, *JavaMath—an API for Internet accessible mathematical services*, in Procs.Asian Symposium on Computer Mathematics (2001).

[34] A. Solomon, *Distributed computing for conglomerate mathematical systems*, in Integration of Algebra& Geometry Software Systems, Joswig M. et al (eds.) (2002).

[35] D. Tepeneu and T. Ida, *MathGridLink—A bridge between Mathematica and the Grid*, in Proc. JSSST'03 (2003), pp. 74–77.

[36] W3C, *MathML 2.0* (2001), `http://www.w3.org/Math/`

[37] Wolfram Research, *gridMathematica*, `http://www.wolfram.com/products/gridmathematica/`

[38] Wolfram Research, *webMathematica*, `http://www.wolfram.com/products/webmathematica/`

[39] A. Al Zain, P. W. Trinder, H. W. Loidl, G. J. Michaelson, *Supporting high-level Grid parallel programming: the design and implementation of Grid-GUM2.* UK e-Science Programme All Hands Meeting (AHM) (2007).

[40] A. Al Zain, K. Hammond, P. Trinder, S. Linton, H. W. Loidl, and M. Costantini, *SymGrid-Par: design a framework for executing computational algebra systems on computational Grids*, in Procs. ICCS'07, Intl. Conference on Computer Science Beijing (2007), LNCS, Springer, in print.

[41] J. Zimmer, A. Franke and M. Kohlhase, *MATHWEB-SB: A Software Bus for MathWeb* (2006), `http://www.mathweb.org/mathweb/`

[42] A. YarKhan, J. Dongarra and K. Seymour, *GridSolve: The Evolution of Network Enabled Solver*, in Procs. WoCo9, A. Prescott (ed), Series IFIP Intern. Federation for Information Processing vol. 239, Springer (2007), pp. 215–224.

*Edited by:* Marcin Paprzycki
*Received:* May 4, 2008
*Accepted:* May 20, 2008