Generic Access to Web and Grid-based Symbolic Computing Services: the SymGrid-Services Framework

Alexandru Cârstea, Marc Frîncu, Georgiana Macariu, Dana Petcu and Kevin Hammond
Institute e-Austria Timișoara, Romania
School of Computer Science
University of St. Andrews, Scotland
Email:{science, dana}@ieat.ro
Email:kh@cs.st-and.ac.uk

Abstract—Modern Grid and Web technologies provide straightforward access to applications and services running on remote resources. The user (which may be a software developer or even an application) should be able to discover such a service and dynamically load an interface to interact with it. While some software systems support either fully or partially automatic creation of client software to access Web services, fewer support Grid services in the same way. This paper introduces a new system that automatically generates client software to allow access to both Grid and Web services. Although designed as a stand-alone tool for accessing any Web or Grid service, we demonstrate its usefulness in the context of the SymGrid framework for Grid-based symbolic computations.

I. INTRODUCTION

Grid and Web services are generally designed not to interact with humans, but rather with software components, namely service clients and other Grid or Web services. Each time a service is created, corresponding client software must be produced. It is, however, impractical to produce a specialised client for each instance of a remote Grid or Web service. Rather, having discovered a service, the user should be able to dynamically load an interface that allows interaction with that service.

Although several recent tools are able to encapsulate and execute complex applications as Grid services, few also provide a way for an application developer to generate an application specific interface that can be provided to the end user. One obvious solution is to exploit the fact that every Web service and all recent Grid services are described by a WSDL document describing the Web service and automatically generates Java client code that calls the Web service; the developer must create the real method calls on the prepared interfaces in the client code. Other solutions to generate the Java classes needed to invoke a Web service programmatically are Novell exteNd Workbench, JAX-RPC Stylus Studio, etc.

Other solutions have been prototyped by various research projects. For instance, the ASSIST [1] framework aids the application developer by providing them with a proxy library whose entries are the stub methods for the remote Web service. These are generated from the service’s WSDL file. The programmer must instantiate the stubs with the code needed to invoke the service’s methods, and place calls to the stub methods within the code provided by the framework modules. A different approach is taken by Xydra-OntoBrew [4]. This provides on-the-fly WSDL to Web-form generation for simple services and portlet clients: the Xydra servlet takes WSDL as input, generates an XHTML form that allows the user to provide an input message, gathers the submitted input values and converts name-value pairs into an XML message that is sent to the Web service. Finally, it displays any result messages. While Xydra is a sophisticated response to the client code creation problem, simpler solutions are needed especially when workflow execution of combined services is desired.

Although Grid services have different goals from pure web services (sharing computing power and resources such as disk storage databases and software applications, rather than sharing information), a Grid service is basically a Web service with a few extensions: statefulness, service instantiation, named service instances, two-level naming scheme, a base set of service capabilities, and lifetime management [9]. It follows that Web services tools must be extended if they are also to be used for Grid services.
The first Grid service standard to appear, OGSI, was not fully compliant with the SOAP standard. Lessons learned from this were applied to create the WSRF standard [20]. Working with WSRF allows Grid services and Web services to be treated uniformly. Remote operations can be invoked by supplying the location of the server, the name of the operation and the list of parameters to be passed.

Application experts are often unwilling to deal with the low-level code development that is mandated by approaches such as those described above, but require straightforward ways to access remote services from within domain-specific applications or environments. The classical solution is to provide application-specific libraries that allow access to specific services. However, if the underlying Web or Grid technologies change in any significant way, then all of these library routines must be rewritten. A more flexible solution is to build a single generic package to access Web and Grid services and to then provide simple application-specific adapters to this package. This is the approach that we are taking in the SymGrid system, a new standard platform for Grid-based symbolic computations (see Section II).

As noted above, some tools already exist that will automatically generate Web services clients. However, these tools do not include Grid service capabilities, and, as far as we are aware, there is no previous automatic generator that supports WSRF-compliant services. In this paper, we describe a generator that treats both Grid services and Web services in a uniform manner. By using this tool, Computer Algebra Systems (CAS) can access external Grid and Web services via an external component: CAGS (the Computer Algebra to Grid Services component), which interfaces the CAS to the Grid. CAGS supports URL-based access using both http and https transport protocols. It also supports the SSL security certificates management required by the Globus Toolkit GSI [8].

The remainder of this paper is structured as follows. Section II briefly describes our objectives in constructing the SymGrid system; Sections III and IV describe the design and implementation of the generic CAGS tool; Sections V and VI describe examples. Finally, Section VII concludes.

II. SymGrid Aims and Objectives

Symbolic computation software systems are vital tools in several areas of modern academic and commercial research. The aim of the EU Framework VI SCIEnce project (Symbolic Computation Infrastructure in Europe, http://www.symbolic-computations.org) is to improve integration between CAS developers and application experts. The project includes developers from four major CASs: GAP, Maple, MuPAD and Kant; plus application experts organised through the international Research Institute for Symbolic Computation, RISC-LINZ. Its main objectives are to:

1) develop versions of the CASs that can intercommunicate via a common standard Web services interface, based on domain-specific results produced by the OpenMath [14] and MONET [12] projects as well as generic standards such as WSRF;
2) develop common standards/middleware to allow the production of Grid-enabled symbolic computation systems;
3) promote and ensure uptake of recent developments in programming languages, including automatic memory management, into symbolic computation systems.

Since our research is primarily concerned with parallel and Grid computations, we are mainly concerned with achieving the second objective: that of providing Grid-enabled symbolic computations using a novel framework. SymGrid. Our goals in developing this framework are to:

1) produce a portable framework that will both 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;
2) develop resource brokers that will support the irregular workload and computation structures that are frequently found in symbolic computations; and
3) implement a series of applications that will demonstrate the capabilities and limitations of Grid computing for symbolic computations.

These objectives cannot be achieved without introducing new higher-level middleware systems than are currently available. By providing a new domain-specific framework for symbolic Grid computations we aim to supply a sophisticated interactive computational steering interface integrating seamlessly into the interactive front-ends provided by each CAS, and providing simple, transparent and high-level access to Grid services. By defining common data and task interfaces, we will allow complex computations to be constructed by orchestrating heterogeneous distributed components into a single symbolic application. By providing generic interfaces where possible, we also anticipate that it will in fact, in time, be possible to exploit our framework for other application domains.

In this paper, we focus on the Grid service provision requirement, that is on the component we term SymGrid-Services. The complementary component that allows symbolic computations to be executed as high-performance parallel computations on a computational Grid is described elsewhere [2]. 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 [15]. None of these systems conforms to all three of our basic requirements, however, which are to:

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

In addition to dealing with these key issues, a number of major new obstacles remain to be overcome if effective symbolic Grid applications are to be developed. Amongst the most important requirements are mechanisms for adapting
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. The intention of SymGrid is 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.

III. THE CAGS ARCHITECTURE

CAGS allows Computer Algebra Systems to leverage the computing capabilities offered by external Grid or Web services. A CAS user must be able to i) discover services; ii) connect to remote services; iii) call remote operations; and iv) run jobs and transfer files in a seamless fashion. In order to meet these requirements we have created a component that provides this functionality to the CAS. The component interface consists of three Java classes: SGServices; SGProxyCert; and SGUIUtils.

1) SGServices: this provides three kinds of operations – retrieval of the list of services registered to a certain UDDI registry or Globus container; retrieval of signatures for the exposed operations of a service; and calling remote operations. The corresponding methods are (Figure 1):

- **getGridServiceList**: returns the list of services deployed in the specified Globus container. The list of services is obtained by querying the Globus ContainerRegistryService using XPath queries [21];
- **getWebServiceList**: returns the list of services registered to a UDDI registry;
- **getOperationsList**: returns the list of operations exposed by a Grid/Web service;
- **isGridService**: indicates whether a service is a WS-Resource (usually for Grid services);
- **getWSResourceEPR**: used to call a factory service and to retrieve the EPR of the newly created WS-Resource (usually for Grid services); and
- **callOperation**: calls an operation exposed by a service.

2) SGProxyCert: this 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. The class can also be used to retrieve information about the owner of a X509 certificate and about the lifetime of a proxy certificate. Since a user may have more than one X509 certificate, but with only one being used at a certain moment, when creating a proxy certificate, the location of the
certificate is automatically stored in a session file. When a proxy certificate is needed, this default certificate is loaded and used. The SGProxyCert class provides the following methods:

- `createProxyFile` creates a new proxy – the location of the proxy can either be supplied by the user or automatically generated;
- `proxyDestroy` used to destroy an existing proxy;
- `getCertInfo` returns information about the owner of the certificate; and
- `isProxyValid` indicates whether the proxy is valid.

3) `SGUtils`:

- `copyFile` copies files using the Globus Toolkit RFT service;
- `deleteFile` deletes files using the RFT service;
- `runJob` allows Globus jobs to be submitted in a synchronous manner;
- `submitJob` offers the possibility to allow Globus jobs to be submitted in an asynchronous manner and returns a unique identifier for further reference;
- `getJobStatus` checks the status of an asynchronous job;
- `getJobFiles` enables the user to retrieve the files resulted by executing the job.

**A. RunManager**

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 we therefore run the supplied Java classes in a child process created by the CAS and which communicates with the CAS uses standard input/output streams to pass parameter and return values (Figure 2). The process is invoked by a script that executes the command `java RunManager arg0 arg1 arg2 ... argN`, where `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 completely generic command that exploits Java reflection capabilities to allow the execution of any class.

**IV. THE CAGS IMPLEMENTATION**

**A. WSRF Specifications and Implementation**

A Grid service is a Web service that conforms to a set of conventions (interfaces and behaviors) that permits the creation of stateful Web services [20]. Statefulness is achieved by separating the service from the state information and by providing mechanisms to identify and manage this state information. A stateful WS-Resource is uniquely identified by an Endpoint Reference (EPR). The WSRF standard for Grid services includes specifications for: WS-Resource, WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup, and WS-BaseFaults. Related standards include WS-Addressing and WS-Notification. The de-facto standard middleware for creating computational grids that implementing these specifications is Globus Toolkit 4 [8]. Using an EPR is more complex than simply using the list of methods exposed by a registered service, and a special method was therefore included in CAGS to deal with this case. The service that is pointed to by its URL is not the service itself, but a registered factory that generates a service instance for the particular service requestor.

**B. Tools and Libraries**

Globus-based Grid services are currently developed using one of three WSRF implementations: Java WS Core, C WS Core or the Python WS Core. The most comprehensive implementation is the Java WS Core component, which uses Java 1.5. Amongst other features, it implements a set of predefined port types that simplify access to and manipulation of WS-Resources. Since Globus Toolkit 4 exposes any built-in capability as a Grid service, the Globus container also exposes by default several services used to support resource management, file transfer and job submit and control. These services are grouped as data management services, information services, and execution management services. From these existing services, the CAGS tool uses the Reliable File Transfer (RFT) service for data management and transfer, DefaultIndexService and ContainerRegistryService to obtain the list of services deployed in a virtual organization, and WS-GRAM to run jobs. We have therefore used the Java 1.5 SDK as the basis the CAGS implementation, extending the Apache Axis implementation of the standard SOAP protocol that is used by the Globus Toolkit to also provide https support. We use the UDDI4J package to query UDDI-compliant Web services registries.

On the client side, before generating the service client, the stubs and Java Beans that are needed by the call must first be generated. CAGS uses the Apache Axis subcomponents...
offered by the WSDL2Java tool to achieve this. Since these tools were originally intended only to be used in conjunction with the http protocol, we have needed to extend them to also cover the https protocol.

C. Security Considerations

Grid environments allow remote execution of operations that could potentially expose security holes, through access to file systems, access to licensed software tools etc. Sensitive information must be protected and the Grid environment must prevent unauthorized access. The Globus Toolkit GSI offers the required security features: secure communication; single sign-on; delegation of credentials; and the ability to manage security issues in a decentralized manner. GSI exploits X509 security certificates to provide this security. These base certificates allow the user to generate a proxy certificate with a limited lifetime. This ensures a good level of security at reasonable effort. The CAGS tool permits access to both secure and non-secure services. In particular SymGrid provides secure access based on Grid certificates to commercial mathematical software and also provides public access to open source mathematical software.

D. Service Discovery Issues

The main purpose of CAGS is to permit remote Grid/Web services calls. If a Web service is intended to be public, the Web service can be advertised by registering the service to a public UDDI registry. Such a registry is designed to be interrogated using SOAP requests and provides access to WSDL documents describing the Web services that are registered in it. Obtaining a list of the Grid services deployed in a container is similar to normal Web service discovery.

The list of available Grid/Web services provides little or no information about the suitability of those services to the user’s needs. Research into how best to discover the most appropriate service and provide secure access based on Grid certificates to commercial mathematical software and also provides public access to open source mathematical software.

E. The FlattenXML format

Both Grid and Web services allow operations to be exposed that accept complex user-defined types in either input parameters or return values, where the type is described using an XSD definition. To obtain an in-memory representation, the type can be mapped to a Java Bean, which can in turn be generated by the Apache Axis WSDL2Java tool. The challenge in CAGS is in representing such types so that they can be easily used in the CAS. Since the CASs and the Java environment communicate using a string-based representation; it is not possible to share in-memory objects. The flattenXML format that we propose here is one solution to this problem. Rather than providing a full definition of flattenXML, we give a simple illustrative example here. For a complex type represented by the hierarchy:

```xml
<person>
  <name>
    <firstname>
      John
      </firstname>
      <lastname>
        Smith
      </lastname>
  </name>
  <age>23</age>
</person>
```

the corresponding flattenXML string is:

```xml
"person:name:firstname:string:John;
person:name:lastname:string:Smith;
person:age:int:23;"
```

where the path separator and parameter separator used here, for simplicity, are "," and ";", but would normally be less commonly used separators such as ":#" and ":###".

Since SymGrid services will communicate using the OpenMath format (an extension of XML), all SymGrid methods will receive an OpenMath object as their input and should return an OpenMath object. Such objects will be communicated between the CAS and CAGS using flattenXML.

F. Calling Operations

In order to call an operation, the user must supply the service providing the operation, the name of the operation, and the list of required parameters. CAGS first uses the information supplied in the WSDL file that describes the operation to generate the corresponding Java Beans and communication stubs. It then dynamically generates a new client that will invoke the required operation. In order to provide the service stubs, CAGS uses a helper tool that has been implemented as part of the GSBT project [10].

One special category of services is the standard set provided by the Globus Toolkit. The RFT service and WS-GRAM-related services are treated specially since it is possible to create static clients for these services and require a valid proxy certificate to be supplied.

V. Usage scenarios

A. Typical scenarios

The primary functionality of CAGS lies in obtaining a list of Grid/Web services registered at a certain URL; obtaining the signatures of those operations that are exposed by a certain Grid/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.

A typical scenario begins with the discovery of a service by consulting a service registry URL (either a UDDI registry or a Globus Container):

```java
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
```

Here, 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.

B. Accessing CAGS from the GAP System

One of the initial SymGrid targets is the GAP [5] computational algebra system. We have therefore constructed a library of GAP functions that allows a GAP user to access the CAGS functionality without needing to know the details of the implementation. There is one function in the GAP library for each method in the CAGS interface. The general pattern used to wrap the CAGS functionality is:

```plaintext
service := select_service(service_list)
operation_list := get_operation_list(service, toMatch)
operation := select_operation(operation_list)
[result := call_operation(service, operation, parameters)]
end scenario
```

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, then the unnecessary steps can be omitted.

A. Listing Services

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

```plaintext
java science.run.RunManager
    science.clients.wrappers.SGServices
    getWebServiceList
    "http://uddi.sap.com/uddi/api/inquiry"
    "A" "caseSensitiveMatch "
```

produces the result:

```
http://192.168.6.98/HelloWorld/Service1.asmx
http://www50.brinkster.com/vbfacileinpt/np.asmx
http://www.webservicex.com/AustralianPostCode.asmx
```

This is a list of Web service URLs representing all services in the UDDI registry whose names include the substring A.

B. Listing Operation Signatures

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 Fermat service that is deployed on the SymGrid testbed, as the result of the command:

```plaintext
java science.run.RunManager
    science.clients.wrappers.SGServices
    getOperationsList
    "http://194.102.62.44:8082/wsrf/services/
    science/FermatService"
```

the following list of operations can be obtained:

```plaintext
string Bin (string)
string Prime (string)
string Rand (string)
string Sigma (string)
string Prod (string)
string Modulus (string)
```

Note that since SymGrid services take and return OpenMath objects, the arguments and results to each operation are strings rather than the integers that might be expected.

C. Calling an Operation

Remote operation invocation is one of the main reasons for using CAGS. To call the FermatService operation that

VI. EXAMPLES

In this section, we show how CAGS can be used directly by exploiting the RunManager utility program. Three categories of services have been used for our initial tests:

1) general Web services such as those deployed at [http://uddi.sap.com/uddi/api/inquiry](http://uddi.sap.com/uddi/api/inquiry);

2) domain-specific symbolic Web services such as those provided by MONET [13] and GENSS [7];

3) simple test Grid services, that we have deployed on a single cluster. These symbolic computation services wrap publicly available CASs: CoCoA, Fermat, GAP, Kant, Macaulay, MuPAD, PARI, Singular and Yacas. The test services are deployed in a Globus container at [http://194.102.62.44:8082](http://194.102.62.44:8082) (this is due to be replaced with an official SymGrid address in Summer 2007).
calculates the greatest common divisor of 96 and 80, we might use the following call:

```gap
java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
science.run.RunManager
science.clients.wrappers.SGServices
callOperation
"/bin/ls" "-l:" "-l::/tmp" "" ""
```

obtaining as result the string `string:16`. Note that the final version of `FermatService` (to be produced by September 2007) will accept only two OpenMath objects instead of the two integer values currently allowed by this service.

### D. Creating a Proxy

The creation of a proxy certificate assuming that the user already has a X509 certificate can be done with the command:

```java
java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
science.run.RunManager
science.clients.wrappers.SGProxyCert
createProxyFile
"/tmp/testproxy1" "" "" "" "" "" "" ""
```

This returns the local location of the newly created proxy certificate, here `/tmp/testproxy1`.

### E. Running a Job

Standard Globus services offer, amongst other things, the ability to run a Globus job on a remote machine. To achieve this, the user must supply the shell command or the program to run, the number of times the job must be run, the parameters and the names of the result files. Empty result files result in default locations being used. The result of the command indicates whether the job was successful on not. For example, to run a job that lists the contents of a directory we can issue the command:

```java
java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
science.run.RunManager
science.clients.wrappers.SGUtils
callOperation
"/bin/ls" "-l:" "-l::/tmp" "" "" "" ""
```

A successful invocation will result in the return value 1, with results stored in a new file created in the `/tmp` directory.

### F. The GAP Interface to CAGS

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 service. The result of the call is displayed by GAP on the console:

```gap
gap> SG_CreateProxy("path_proxy", ",", ",", "pswd");
gap> gridServList := SG_GridServiceList( 
"http://194.102.62.44:8082/wsrf/services/", 
"YACAS");
http://194.102.62.44:8082/wsrf/services/science/
```

### G. Web Portal Example

To demonstrate that CAGS can be integrated into a variety of applications, we have created a Web portal, using Java servlets deployed into a Tomcat Apache server. The general structure of servlet applications imposed some minor changes regarding the location of the libraries used by CAGS, but was otherwise straightforward, and allows the full CAGS functionality to be accessed from HTML forms. Users are able to login into the system, query for a Web/Grid service list, call a remote Grid/Web service operation, or launch a Globus job. If an anonymous connection is made to the portal, a Web user is restricted to non-secure Grid services. Logging to the portal causes a proxy certificate to be created that can be used when invoking secure services. Figure 3 gives some screenshots showing how the portal can be used. The portal is available for test purposes at [http://194.102.62.36:8080/portal](http://194.102.62.36:8080/portal).

### VII. CONCLUSIONS AND FUTURE DEVELOPMENTS

In this paper, we have introduced a generic tool, CAGS, that has been constructed to allow easy access to WSRF-based Web and Grid services from within symbolic computations. While this is only a small component of the planned SymGrid middleware, it represents an important first step. We will now test its utility as part of the SymGrid design by trying a larger number of existing mathematical services, using a larger private test Grid that is being constructed to link Romania, Germany and Scotland, and also by testing special services that are requiring high-performance computing.

In the next few months, we will construct a new component to allow deployment of Web and Grid-based symbolic computing services. This new component must be also generic to easily expose new services. Initially this component will be used for the deployment of basic SymGrid services, wrapping instances of the GAP, MuPAD, Kant and Maple kernels into Grid services.

Another issue that we will address in the coming months will be service composition. Currently, only a few workflow tools such as GridNexus [3] can be used in conjunction with the CAGS since it is necessary to generate the service stubs we require.

Finally we review some of the novel elements of the system that is described in this paper: a solution to uniform access of Web and Grid services; application to a new domain that will strongly benefit from Web and Grid service integration; a generic scenario for interacting with Web and Grid services from within symbolic computing systems; and finally the
Fig. 3. Examples of using the Web portal: top, a Web service, middle a Grid service wrapping YACAS [22], bottom client call simulation and the result construction of a publicly available test environment that links several symbolic computing facilities into a single Web portal.

ACKNOWLEDGMENTS
This research is partially supported by European Union Framework 6 grant RII3-CT-2005-026133 SCIENCE: Symbolic Computing Infrastructure in Europe, and by a one-year SOED support fellowship from the Royal Society of Edinburgh.

REFERENCES
[19] WSDL, http://www.w3.org/TR/wsd1