Towards OGSA Compatibility in Alternative Metacomputing Frameworks*

Gunther Stuer[1], Vaidy Sunderam[2], and Jan Broeckhove[1]

[1] Dept. of Math and Computer Science,
University of Antwerp, 2020 Antwerp, Belgium.
{gunther.stuer, jan.broeckhove}@ua.ac.be
[2] Dept. of Math and Computer Science,
Emory University, Atlanta, GA 30322, USA
vss@mathcs.emory.edu

Abstract. Lately, grid research has focused its attention on interoperability and standards, such as Grid Services, in order to facilitate resource virtualization, and to accommodate the intrinsic heterogeneity of resources in distributed environments. To ensure interoperability with other grid solutions, it is important that new and emerging metacomputing frameworks conform to these standards. In particular, the H2O system offers several benefits, including lightweight operation, user-configurability, and selectable security levels; with OGSA compliance, its applicability would be enhanced even further In this contribution, a framework is presented which will augment the H2O-system with the functionality to produce and publish WSDL and GSDL documents for arbitrary third party pluglets, and thereby enhance OGSA compatibility.

1 Introduction

The benefits of distributed computational systems are well established, and numerous software architectures and toolkits have evolved in recent years to support this mode of computing. Most of these systems however are rather limited in scope. Some, such as UD Patriot Grid [1] and SETI@Home [2] are targeted at specific research projects, while others, such as MPICH [3] and PVM [4] are usually confined to single administrative domains. More generalized metacomputing systems, or *grids*, have gained tremendous popularity in recent times because they enable secure, coordinated, resource sharing across multiple administrative domains, networks, and institutions. This model [5] has been realized in several software toolkits, such as Globus [6] and Legion [7].

However, many applications of smaller magnitude do not require explicit coordination and centralized services for authentication, registration, and resource brokering as is the case in traditional grid-systems. For these applications, a lightweight and stateless model, in which individuals and organizations share

* Research supported in part by U.S. DoE grant DE-FG02-02ER25537 and NSF grant ACI-0220183.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3036, pp. 51–58, 2004.

their superfluous resources on a peer-to-peer basis, is more suitable. Two examples of such lightweight peer-to-peer distributed computational systems are H2O [8,9,10] and JGrid [11].

H2O, the framework used in our research, is a novel component-based, service-oriented framework for distributed metacomputing. Adopting a provider-centric view of resource sharing, it emphasizes lightweight software infrastructures that maintain minimal state. Resource owners host a software backplane, known as the *kernel*, onto which owners, clients, or third-party resellers may load components or component-suites, known as *pluglets*, that deliver value added services without compromising owner security or control.

In the current phase of evolution, grid research has focused on interoperability and standards in order to facilitate resource virtualization and to accommodate the intrinsic heterogeneity of resources in distributed environments. To this end, OGSA [12] aims to define a new common and standard architecture for grid-based applications based on the concept of Grid Services, an extension of Web Services [13]. The formal and technical specification of these concepts can be found in the OGSI [14] specifications and a reference implementation is provided by the GTK3 [15].

Such standard frameworks, based on XML, are used to describe service specifications in a universally understood manner, thereby permitting clients to discover and utilize services across platforms and context domains. The functional description of a Web Service is written in the *Web Services Description Language* (WSDL) [16]. It can be published using various registration and discovery schemas, such as UDDI [17]. Grid Services are described using an extension of WSDL known as *Grid Services Description Language* (GSDL) [14].

Being compliant to these standards is not only important for heavyweight Grid-systems such as the Globus Toolkit, but also for the lightweight solutions referred to above. In H2O this is already partially the case. Through the use of RMIX [18] as the communications layer, pluglets can be exported using various remote bindings such as stub-less JRMP, IIOP and SOAP. When using the latter, the exported pluglets can be seen as Web Service instances that provide standardized SOAP-endpoints.

However, in H2O there is currently no provision for the creation and publication of WSDL and GSDL documents for pluglets that have already been deployed. At present, the WSDL/GSDL description file needs to be created manually and be published using third party tools. In this contribution, a framework is presented that will augment the H2O-system with the functionality to produce and publish WSDL and GSDL documents for arbitrary third party pluglets, thus enhancing OGSA compatibility and thereby grid interoperability.

2 Architecture

We present a framework that consists of three important components: two pluglets and one command-line tool.

The *SoapExportPluglet*, or *SEPluglet* for short, is the main component. It is responsible for the generation of WSDL / GSDL documents and SOAP / GridService endpoints for a third party pluglet. In this contribution, we will denote such a pluglet as *pluglet-X*. The second pluglet, *PublisherPluglet* is responsible for publishing generated WSDL/GSDL documents to some registry. *ExportTool*, a command-line tool, serves two purposes. It is both a demonstration of how to use the *SEPluglet*'s API and a utility that allows H2O-users to export new or already uploaded pluglets as Web or Grid Services.

Figure 1 shows the architecture of the framework. A kernel can contain zero or more *SEPluglet* instances, which will all have the same name, but different unique identifiers. Each *SEPluglet* stores a *SoapExportInfo* object for every pluglet it exports. Each *SoapExportInfo* object contains the necessary information and business logic to create the endpoints and WSDL/GSDL documents for the pluglet it represents. Each *SEPluglet* can be monitored by zero or more *PublisherPluglet*s and each *PublisherPluglet* can monitor zero or more *SEPluglet*s. This design allows for substantial flexibility in the choice of registry that publishes the WSDL/GSDL description of the exported pluglets.

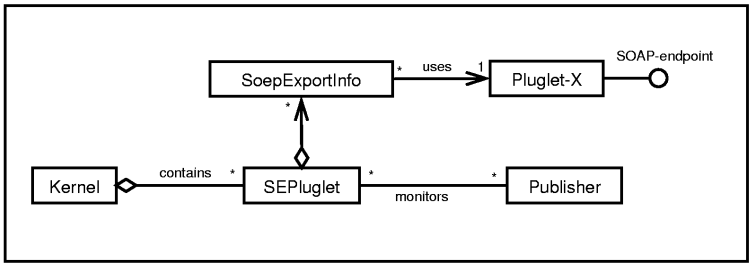

Fig. 1. Architecture of the framework.

3 The Components

3.1 The SoapExportPluglet

The *SEPluglet* is the most important component of our framework. It is responsible for the lookup, creation and destruction of WSDL/GSDL documents and SOAP/GridService endpoints. *SEPluglet* has the capability to deploy and export a pluglet in a single step, but it can also export pluglets that have already been deployed. To allow lookups, it contains a *SoapExportInfo* object for every pluglet it is currently exporting. This feature doubles as a cache: when multiple requests are made to export the same pluglet, the corresponding *SoapExportInfo* object is created only once. *SoapExportInfo* objects remain stored until either the corresponding pluglet is destroyed, or an actor explicitly requests that the export of a pluglet be revoked.

The steps required for uploading and exporting a pluglet are illustrated in the first part of figure 2. Of course, before any service can be exported, the *SEPluglet* and the target pluglet have to be deployed. After this, the *SEPluglet* can be instructed to export a given pluglet by specifying its unique identifier. If this pluglet is not already exported, its PlugletContext will be retrieved from the kernel and stored in a newly created *SoapExportInfo* object which will be returned.

Subsequently, the *SoapExportInfo* object can be instructed to create the endpoints and the WSDL / GSDL documents. Note that artifacts are created only once. Subsequent requests return the cached values. The creation of artifacts is a cascading activity. In order to create a GSDL document, one needs a WSDL document and a Grid Service-endpoint. In order to create a WSDL document, one needs a SOAP-endpoint and the remote Java interface of the pluglet to export. So, as a consequence of constructing the GSDL document, all other artifacts are built as well. This behavior is illustrated in the second part of figure 2.

There are two toggles which modify the behavior of the *SEPluglet*: the *export type* and the *export mode*. The fist determines when the artifacts of an exported pluglet will be created. If set to *lazy*, they will be constructed on first use. If set to *eager*, all artifacts are created during the export process. To preserve resources, by default, the *export type* is *lazy*. The second toggle determines whether all deployed pluglets should be exported automatically or manually. If set to *manual*, an external actor has to explicitly instruct the *SEPluglet* to export the pluglet with the given unique identifier. If set to *auto*, the *SEPluglet* will scan the kernel for all available pluglets and export them all. Furthermore, all subsequently deployed pluglets will be automatically exported as well.

3.2 The PublisherPluglet

Generating the WSDL/GSDL documents is only part of the process. It is also very important to publish them to all appropriate registries, such as UDDI and LDAP, from which they can be discovered by third parties. This allows clients, which are either compliant to the Web Services standard or the Grid Services standard, to discover and use the exported pluglets. Furthermore, when a pluglet is no longer available, the documents should be removed from the registry. These tasks are performed by the *PublisherPluglet*s.

This component is designed in two layers, an abstract base class which takes care of all the bookkeeping and a concrete subclass which implements the business logic to actually publish or remove the WSDL/GSDL documents from a registry. Whether it is the WSDL, the GSDL or both that are published or removed depends on the concrete implementation. This design makes it very easy to add new types of publishers. One only needs to derive from the abstract base class and implement two abstract methods: `publish (SoapExportInfo)` and `unpublish (SoapExportInfo)`. For now, one derived class has been implemented which maintains a cache in memory of all published documents.

The *PublisherPluglet*s can operate in three modes: *manual*, *semi automatic* and *automatic*. They can switch between any of these modes at runtime. In *man-*

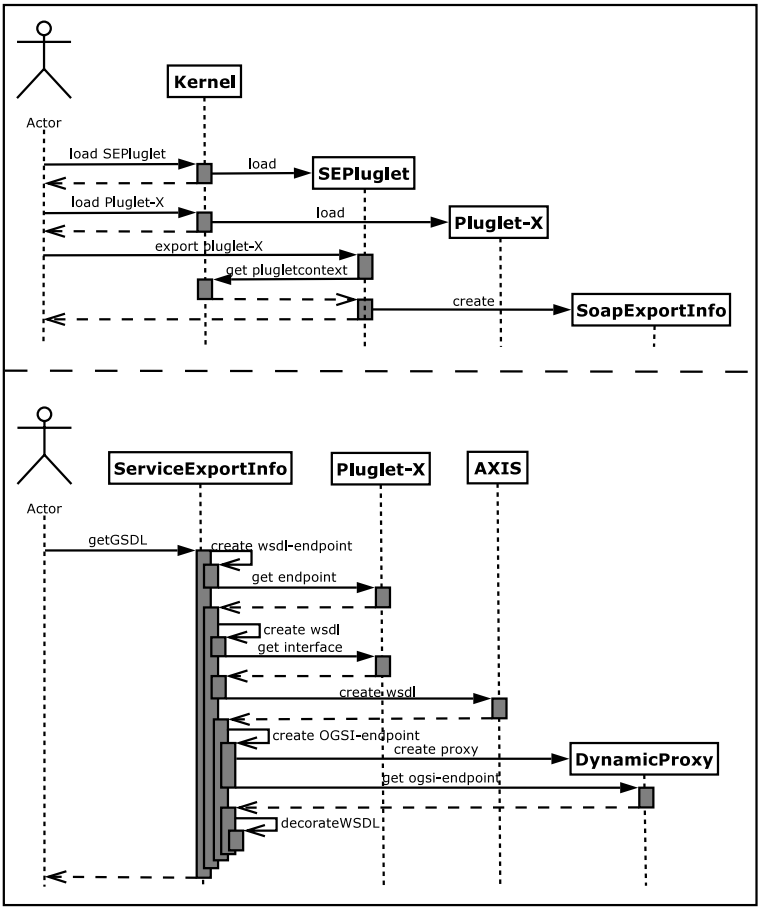

Fig. 2. Exporting and creating a GSDL

ual mode, an actor has to specify the unique identifier of the pluglet that needs its documents published or removed. The *PublisherPluglet* locates the exporting *SEPluglet* and retrieves the corresponding *SoapExportInfo* object. From this, the necessary information can be retrieved to publish or remove the WSDL/GSDL documents. In *semi automatic* mode, an actor can specify which *SEPluglet*s to monitor for export events. If any of these *SEPluglet*s (un)exports a pluglet, its documents automatically get (un)published as well. In *automatic* mode, the kernel is scanned for deployed *SEPluglet*s and all are monitored for export events. Furthermore, all subsequently deployed *SEPluglet*s will be automatically monitored as well.

3.3 The ExportTool

The (*ExportTool*) serves two purposes. It is both a demonstration of how to use the *SEPluglet*'s API and a command-line utility which allows H2O-users to export new or already deployed pluglets as Web or Grid Services. This tool can export an existing pluglet by specifying its unique identifier or name. In the second scenario, all pluglets with that particular name will be exported. Two similar operations are available to unexport one or more pluglets. If the pluglet is not yet deployed, it is possible to deploy and export it in one step. For this, the pluglet name, service class and classpath have to be specified. There are two version of this operation: one taking the list of arguments mentioned above, and one taking the name of a Java properties-file containing the required information.

4 The Artifacts

4.1 The SOAP-Endpoint

The RMIX subsystem is capable of exporting a pluglet using various communication protocols. When RMIX is instructed to export a given pluglet using the SOAP protocol, a new SOAP-endpoint is constructed. This endpoint is represented as a URL, pointing to the address and port where RMIX is listening for incoming SOAP-requests for that particular pluglet.

However, this endpoint only remains valid as long as the session in which it was created exists. In H2O, one has two types of sessions: client-sessions and pluglet-sessions. A client-session is created when a client logs into a kernel and the session is destroyed during logout, or when the connection to the kernel is broken. A pluglet-session is associated with a particular pluglet and it is created when the pluglet is loaded. It is destroyed when the pluglet is removed from the kernel. In the current H2O release (0.8.2), endpoints are associated with the originating session. When a pluglet is manually exported using the *ExportTool*, the originating session is a user-session and the endpoints will remain valid as long as this session is active, i.e., as long as the *ExportTool* is connected to the kernel.

To address this problem, an extension to the H2O-kernel is needed. It has to be possible to specify that an endpoint is associated with a session other than the originating one. This way, it can be associated to the *SEPluglet*'s session and the endpoint will remain valid as long as the *SEPluglet* is active. With this extension, the *ExportTool* could connect to a kernel, export a pluglet, create the endpoints and disconnect again without invalidating the endpoints. This feature will be incorporated into a future H2O release.

4.2 The WSDL-Document

The WSDL-document is created using the Apache AXIS tool[19]. Among other things, this tool has the functionality to convert Java interfaces in WSDL-documents. To do so, AXIS needs access to the class files of the remote interface

defining the pluglet's available operations. In H2O this is not as straightforward because, for security reasons, each pluglet has its own classpath. This prevents the *SEPluglet*, and by extension, the AXIS-engine from reading the class files of the pluglet to be exported. This problem was solved by modifying the kernel's security policy file to allow the *SEPluglet* to construct new classloaders. For each pluglet that is exported, a new classloader is created which is the union of the *SEPluglet*'s classloader and the one from the pluglet to be exported. This new classloader is then passed to the AXIS-engine.

4.3 The Grid Service-Endpoint

An important step in making H2O OGSI-compliant is to ensure that all pluglets with a SOAP-compliant interface can be exported as a Grid Service. According to the OGSI-specifications [14], this means that they must be addressable using SOAP and must implement a number of pre-defined operations. However, since exporting a pluglet as a Grid Service is only one of several possible bindings, it is not desirable that pluglets implement these methods themselves, nor that they have to be derived from a superclass. For this reason, a mechanism has to be in place which will dynamically extend existing pluglets with the pre-defined OGSI-operations whenever they are exported as Grid Services.

A solution can be constructed using Java's dynamic proxies. They allow a new class to be constructed at runtime which will implement all the pluglet's original operations plus those defined by OGSI. This proxy object can subsequently be exported as a SOAP-endpoint and will serve as a method router, dispatching incoming method invocations to either the proxied pluglet or to some object implementing the methods defined by OGSI. There is one remaining problem though. The current version of RMIX-SOAP does not allow an object to export multiple remote interfaces. This is a necessary feature because the exported proxies implement two remote interfaces: one defining the operations on the proxied pluglet, and one defining the OGSI-operations. This feature will be incorporated in a future H2O release.

4.4 The GSDL-Document

GSDL is basically an extension of WSDL. Therefore, building a GSDL-document essentially involves building the WSDL-document and subsequently extending it with the GSDL-specific parts. Three additions have to be made to the WSDL-document. The namespace *girdservicesoapbinding* has to be defined, the file *ogsi_bindings.wsdl* has to be imported and an extra port, *GridServiceSOAP-BindingsPort*, has to be added. The algorithm that we use is based upon the *DecorateWSDL*-algorithm of GTK3. Since it uses files for input and output, which is not desirable in H2O-kernels, the GTK3-implementation could not be used as is. A slightly modified version has been implemented which uses strings for input and output.

5 Summary

In this contribution we have presented a framework to export the pluglets of the H2O computational system as Web/Grid Services, and thereby enhance OGSA compatibility. The core of the framework is the *SEPluglet*, which is responsible for generating the WSDL/GSDL artifacts. The second pluglet, *PublisherPluglet*, is responsible for the publication of the WSDL and GSDL documents of exported pluglets.

A kernel can contain multiple *SEPluglet* instances and each instance can be monitored by multiple *PublisherPluglet*s. This design allows for great freedom as to the location at which the WSDL/GSDL descriptions of exported pluglets will be published.

References

1. UD. Project: The PatriotGrid. `http://www.grid.org/projects/patriot.htm`.
2. SETI@Home. `http://setiathome.ssl.berkeley.edu`.
3. W. Gropp, E. Lusk, N. Doss, and A. Skjellum: *A high-performance, portable implementation of the MPI message passing interface standard*. Parallel Computing, 22(6):789-828, Sept. 1996.
4. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam: *PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambrdige, MA, USA, 1994.
5. I. Foster, C. Kesselman, and S. Tuecke: *The anatomy of the grid: Enabling scalable virtual organizations*. Int J. of Supercomputer Applications, 153), 2001.
6. I. Foster, and C. Kesselman: *A metacomputing infrastructure toolkit*. The Int. J. of Supercomputer Applications and High Performance Computing, 11(2):115-128, 1997.
7. A. Natrajan, M. A. Humphrey, and A. S. Grimshaw: *Grids: Harnessing geographically-seperated resources in a multi-organisationel context*. In 15th Annual Int. Symp. on High Performance Computing Systems and Applications, 2001.
8. V. Sunderam, D. Kurzyniec: *Lightweight Self-Organizing Frameworks for Metacomputing*. In 11th Int. Symp. on High Performance Distributed Computing, 2002.
9. The H2O project home page: `http://www.mathcs.emory.edu/dcl//h2o/`.
10. The H2O tutorial: `http://www.mathcs.emory.edu/dcl//h2o/h2o-tutorial.pdf`.
11. Z. Juhasz, A. Andics, S. Pota: *JM: A Jini Framework for Global Computing*. IEEE Int. Symp. on Cluster Computing and the Grid, 2002.
12. I. Foster, C. Kesselman, J. Nick, and S. Tuecke: *The physiology of the Grid: An Open Grid Services Architecture for distributed systems integration*. `http://www.globus.org/research/papers/ogsa.pdf`.
13. Web Services specifications: `http://www.w3.org/2002/ws/`.
14. Open Grid Service Infrastructure (OGSI): `http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf`.
15. Globus Toolkit 3: `http://www-unix.globus.org/toolkit/download.html`.
16. Web Services Description Language specification: `http://www.w3.org/TR/wsdl`.
17. Universal Description, Discovery and Integration: `http://www.uddi.org/`.
18. D. Kurzyniec, T. Wrzosek, and V. Sunderam: *Heterogeneous Access to Service-based Distributed Computing: the RMIX Approach*. Int. Parallel and Distributed Processing Symp., 2003.
19. The Apache AXIS-tool: `http://ws.apache.org/axis/`.