Computing and Informatics, Vol. 23, 2004, 305–315

PUBLISHING H2O PLUGLETS IN UDDI REGISTRIES

Gunther Stuer, Jan Broeckhove

Department of Math and Computer Science
University of Antwerp, 2020 Antwerp, Belgium
e-mail: {gunther.stuer, jan.broeckhove}@ua.ac.be

Vaidy Sunderam

Department of Math and Computer Science
Emory University, Atlanta, GA 30322, USA
e-mail: vss@mathcs.emory.edu

Manuscript received 24 September 2004
Communicated by Ladislav Hluchý

Abstract. Interoperability and standards, such as Grid Services are a focus of current Grid research. The intent is to facilitate resource virtualization, and to accommodate the intrinsic heterogeneity of resources in distributed environments. It is important that new and emerging metacomputing frameworks conform to these standards, in order to ensure interoperability with other grid solutions.

In particular, the H2O metacomputing system offers several benefits, including lightweight operation, user-configurability, and selectable security levels. Its applicability would be enhanced even further through support for grid services and OGSA compliance.

Code deployed into the H2O execution containers is referred to as pluglets. These pluglets constitute the end points of services in H2O, services that are to be made known through publication in a registry. In this contribution, we discuss a system pluglet, referred to as OGSAPluglet, that scans H2O execution containers for available services and publishes them into one or more UDDI registries. We also discuss in detail the algorithms that manage the publication of the appropriate WSDL and GSDL documents for the registration process.

Keywords: OGSA, UDDI, H2O, distributed computing, grids

1 INTRODUCTION

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 [1].This model has been realized in several software toolkits, such as Globus [2] and Legion [3]. Extensive overviews of grid computing can be found in [4, 5].

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 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 [6] and JGrid [7].

Recently, 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 [8] aims at defining a new common and standard architecture for grid-based applications based on the concept of Grid Services, an extension of Web Services [9]. The formal and technical specification of these concepts can be found in the OGSI [10] specifications and a reference implementation is provided by the GTK3 [11].

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) [9]. It can be published using various registration and discovery schemas, such as Universal Description, Discovery and Integration(UDDI) [9].

Grid Services are described using an extension of WSDL known as Grid Services Description Language (GSDL) [10]. Compliance with these standards is not only important for heavyweight grid systems such as those based on the Globus Toolkit, but also for the lightweight solutions referred to above.

However, currenly, in H2O there is no provision for the automated creation and publication of WSDL and GSDL documents of deployed H2O pluglets. Both the WSDL and GSDL (from now on referred to as xSDL whenever the context of the discussion applies to both WSDL/Web Services and GSDL/Grid Services) description files have to be created manually and need to be published using third party tools.

This paper outlines an architecture that allows pluglets to be addressed as Web/Grid services by providing the appropriate SOAP endpoints and by generating and publishing the corresponding xSDL documents in UDDI registries.

2 THE H2O METACOMPUTING FRAMEWORK

H2O is a framework for cooperative distributed computing [6, 13, 14]. It provides a lightweight, general-purpose platform for resource sharing, and aims to support a wide range of application models. Because resource aggregation is a client abstraction and responsibility, no specific centralized discovery or authentication mechanisms are assumed. This opens up the possibility of diverse deployment scenarios, each with appropriate discovery and aggregation techniques.

In the H2O architecture, resource providers independently share their raw resources such as CPUs, memory, or storage, via software backplanes called kernels. Authorized entities can deploy software components, called pluglets, into those kernels. The kernels provide support for lifetime management, communication, and in particular security. Subsequently, pluglets provide services that can be used by any client.

The H2O framework is Java-based: kernels and pluglets are Java objects. As a consequence, H2O kernels may run on virtually any platform, and every pluglet can be loaded into any H2O kernel. Communication between clients, kernels and pluglets utilizes a substrate that features multiple, dynamically pluggable protocols and underlying byte transport customization [15]. This enables non-Java clients to communicate with H2O pluglets. It allows pluglets to be exported using various remote bindings such as stub-less JRMP, IIOP and SOAP.

With the latter, the exported pluglets appear as Web Service instances providing standard SOAP-endpoints. The next step is to describe the Web/Grid service in an xSDL document and register the service with a UDDI registry. Specifically, we need to develop the following capabilities :

- Generating the Web/Grid Service Description Language (xSDL) document from an arbitrary, third-party, pluglet.
- Publishing the generated xSDL-documents to one or multiple UDDI registries.
- Implementing the OGSI-defined container and service features.

Currently, the first two requirements have been fully met. The architecture, design and implementation will be discussed in the next section. The third requirement is future work.

3 ARCHITECTURE

We present a framework that consists of three distinct components: a pluglet, a deployment tool and an administration tool, as depicted in Figure 1. Instead of implementing the new OGSA/Web Service centric features into the kernel itself, we develop a pluglet that provides the necessary functionality. This is in keeping with the H2O philosophy of making the kernel as lean as possible.

The pluglet, called *OGSAPluglet*, is the main component of the framework. It is responsible for the creation of Web and Grid Service endpoints, and for the generation and publication of WSDL and GSDL documents into multiple UDDI registries. Furthermore, two command-line tools are provided: one to deploy the *OGSAPluglet* into a running H2O-kernel and one to administer it.

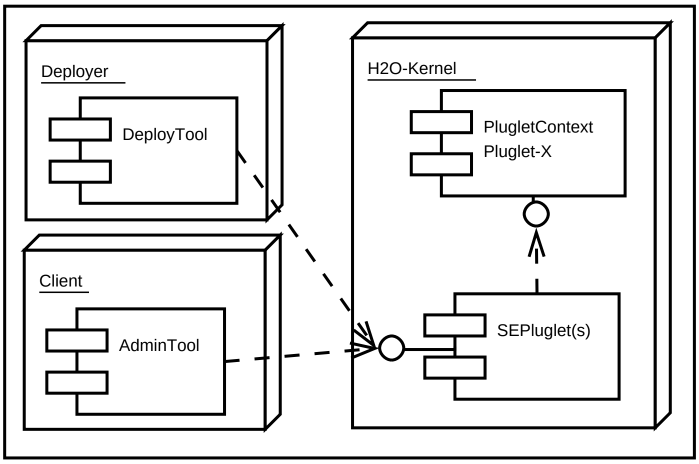

Fig. 1. The main components of the framework

The pluglet can operate in two modes: manual or automatic mode. In manual mode, user interaction is required to start or stop exporting a pluglet as a Web/Grid Service. In automatic mode, the kernel is monitored for deployment events and all pluglets are automatically exported when they first become active. If they disappear from the kernel, all relevant xSDL documents will be removed as well.

Figure 2 shows the architecture of the framework. A kernel can contain zero or more *OGSAPluglet* instances, which will all have the same name, but different unique identifiers. Each *OGSAPluglet* stores an *OGSAExportInfo* object for every pluglet it exports. In this contribution, we will denote such a pluglet as *pluglet-X*. Each *OGSAExportInfo* object contains the necessary information and business logic to create the endpoints and the xSDL documents for the pluglet it represents.

In order to dynamically extend the public interface of third party pluglets with the methods required by OGSI, for each exported pluglet, a dynamic proxy is created which acts as a message router between the proxied pluglet and an object which implements the methods specified by OGSI.

Furthermore, each *OGSAPluglet* holds a list of UDDI registries. Whenever a pluglet is exported using a particular instance of *OGSAPluglet*, this instance's list of registries will be queried by the *UDDIPublisher* component which will perform the publishing of the associated xSDL documents in the respective registries. Since UDDI registries do not contain the xSDL documents themselves, but rather store URLs to them, a minimal HTTP-server, called GSDLServer, is provided.

This design allows for substantial flexibility in the choice of registry that publishes the xSDL description of the exported pluglets.

For a detailed outline of the pluglet logic associated with automatic generation of xSDL documents we refer to [12]. The next section will describe the *UDDIPublisher* component, covering the UDDI publication process.

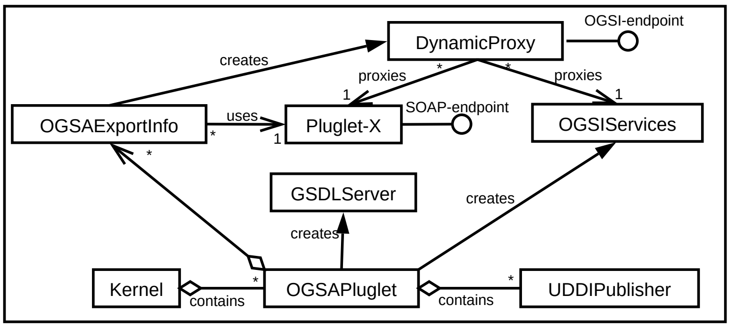

Fig. 2. Architecture of the framework

4 UDDIPUBLISHER

4.1 UDDI Registries

UDDI (Universal Description, Discovery & Integration) is a set of standardized specifications for service description and discovery [9]. A UDDI registry allows a business to publicly list a description of itself and the services it provides. The registry is structured as a hierarchy of business, service, binding, and technical information, using four core information types.

The *BusinessEntity* structure represents a provider of services. Within the UDDI registry, this structure contains information about the company itself, including contact information, industry categories, business identifiers and a list of services provided.

The *BusinessService* structure respresents an individual service provided by the business entity. Its description includes information on how to bind to the web service, what type of web service it is, and what taxonomical categories it belongs to.

A *BindingTemplate* is the technical description of a service represented by a *BusinessService* structure. A single business service may have multiple binding templates. The binding template represents the actual implementation of the business service.

A *TModel* is a way of describing the various business, service, and template structures stored within the UDDI registry. Any abstract concept can be registered within UDDI as a *TModel*. For instance, if you define a new WSDL port type (e.g. GridService), you can define a *TModel* that represents that port type within UDDI. Then, you can specify that a given business service implements that port type by associating the *TModel* with one of the business service's binding templates.

4.2 Selecting a UDDI Registry

Currently there are two freely available implementations of a UDDI registry. The first is from the Apache project and is called jUDDI [16]. The biggest advantage of this implementation is that it is a dedicated UDDI registry, and as such, rather small and easy to configure. Unfortunately, it is a recent project and at the time of this writing, no stable builds are available.

The second implementation is from Sun. It is part of their Java Web Services Developer Pack (JWSDP) [17] and is currently at version 1.3. The size of the JWSDP distribution is considerable but it comes with extensive, well written documentation. We have chosen to use this UDDI registry implementation.

4.3 Using UDDI in Java

There are three different approaches to interacting with a UDDI registry in Java. The first one is to explicitly build SOAP messages to communicate with the registry, the second one is to use some UDDI client API, and the third approach is to use Java API for XML Registries [18]. The important issues for us are ease of use, and the number of Java libraries (jar files) that need to be uploaded to the H2O kernel in order for the OGSApluglet to execute because we want to minimize deployment overhead.

Explicitly programming the SOAP messages to register a web service with a UDDI registry requires considerable effort and is error-prone. There is, however, an important advantage to this approach: it does not require any libraries to be uploaded.

For the second approach, a number of open source client APIs are available for accessing UDDI registries. These APIs allow to interact with UDDI registries without knowing the specifics of SOAP or of the XML messages and data structures used in UDDI. For example, IBM provides the Service Registry Proxy (SRP) that is part of the UDDI4J [19] project. Here the uudi4j.jar library needs to be uploaded.

The Java API for XML Registries (JAXR) specification defines a standardized way for Java programs to access a registry [18]. JAXR allows developers to write code that can access several different registries, including UDDI and ebXML. The trade-off for portability is dealing with the additional layer of abstraction introduced by JAXR. Also, the use of JAXR requires many jar files, seventeen to be precise [20], to be uploaded.

After weighing the advantages and disadvantages of each approach, we have decided to use UDDI4J. UDDI4J supports three SOAP-Transport libraries: Apache AXIS, Apache SOAP and HP-Soap. Unfortunately AXIS is not applicable in our work because it requires access to the local filesystem, which is not allowed in an H2O-kernel (or at least requires specific privileges). HP-SOAP is part of HP's Web Services Platform, which is not freely available. This leaves us with Apache SOAP.

This library uses the Java mail library (mail.jar) and bean activation framework (activation.jar), and both have to be uploaded to the kernel.

4.4 Registering a Service

Registering a pluglet as an Web/Grid-service can be broken down into two steps:

1. The xSDL-documents have to be generated from the interface of the pluglet one wants to publish.
2. The xSDL-documents must be registered to one or more UDDI registries. The service's name and description will be identical to the pluglet's name. The access URL will be the Web/Grid-endpoint constructed by *OGSA Pluglet*.

The UDDI registry stores URL references to the xSDL-documents rather than the documents themselves. This implies that a basic HTTP server is required in order to respond to requests from interested third-parties for the xSDL-document. To this end, the OGSA pluglet contains a minimal HTTP server which is started when the pluglet is deployed.

4.5 Registration Viewpoints

There are two distinct points of view concerning the registration attributes of pluglets in a UDDI registry. In the first point of view, it is the kernel owner that decides under which *BusinessEntity* deployed pluglets are published. In the second point of view it is the pluglet deployer who decides under which *BusinessEntity* his pluglets get registered.

To support both viewpoints, *OGSAPluglet* provides methods to retrieve and set a default *BusinessEntity*. The default will be used in automatic mode, and in manual mode when no *BusinessEntity* is specified. On the other hand, a pluglet deployer can specify a particular *BusinessEntity*, that is to be used in the registration process.

4.6 The xSDL Server

Because UDDI registries do not contain the xSDL documents themselves, but rather URLs to them, a minimal HTTP server is needed. The URL for the xSDL document combines the address of this server, a port number, the unique identifier of the pluglet, and a GSDL or WSDL trailer. The HTTP server will listen on the port referred to in the URL for incoming HTTP GET-requests. If an incorrect identifier is used, an error is thrown. Depending on whether the URL ends with "?WSDL" or "?GSDL" the WSDL or GSDL document is returned, respectively. In all other cases an error is thrown.

4.7 Registration Algorithm

First the UDDI registry is queried for *BusinessEntities* with a given name. If none exists, a new entry will be created. Otherwise, the unique key of the first matching entry will be retrieved. In the second step, the *BusinessEntity*'s services are queried for a service with a given name. If none exists, a new entry will be created. Otherwise, the unique key of the first matching service entry will be retrieved. Thirdly, the access points and URLs of the service's xSDL documents are retrieved. Finally, the new access points and the *TModels* containing the xSDL URLs are added to the end of the list.

This setup allows for different kernels, each publishing the same pluglet to the same UDDI registry. When users query the registry, they will receive a list of all instantiated services. For each registered pluglet a data structure is kept with the business, service, WSDL and GSDL binding keys. This algorithm is depicted in Figure 3.

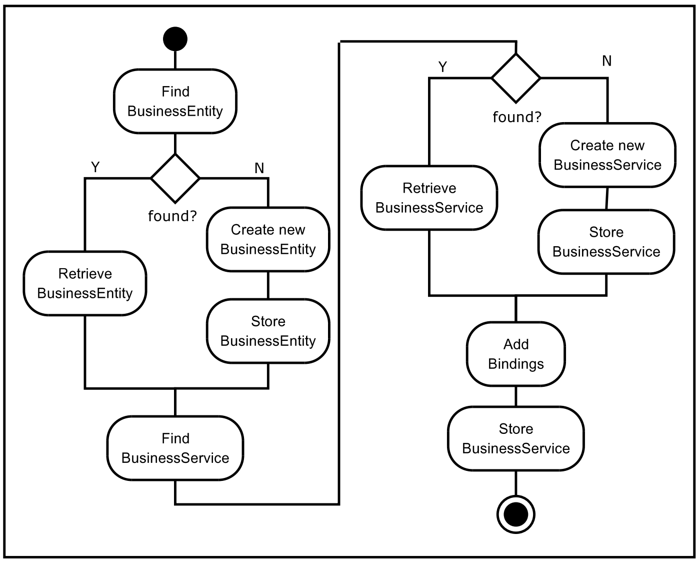

Fig. 3. Registering a service

4.8 Removal Algorithm

Using the service key stored in the data structure mentioned above, the UDDI is queried for this service's entry. Using the binding keys, the WSDL and GSDL bindings created by this instance of the pluglet are searched and removed. If after this step, there are still bindings left, the updated service entry is stored in the UDDI, otherwise the service entry is removed. This algorithm is depicted in Figure 4.

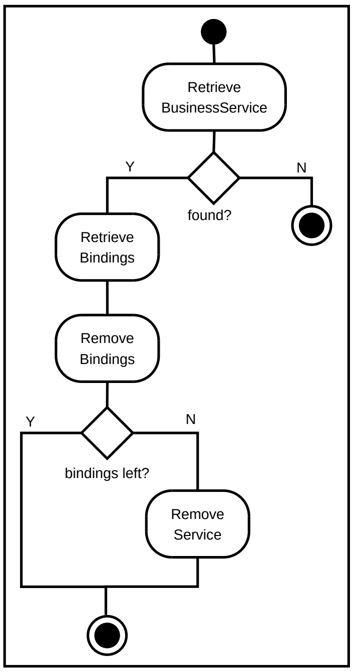

Fig. 4. Unregistering a service

5 SUMMARY

In this contribution we have presented a framework to export the pluglets of the H2O computational system as Web/Grid Services and register them with a UDDI registry. We have thereby enhanced H2O's Web Service/OGSA compatibility.

The full implementation of all operations defined by the OGSI specifications have not been implemented yet. Currently, only the proxy and method routing infrastructure is provided. Other features of OGSA compliance and the impact of the shift from OGSA to the WS-RF Web Service Resource Framework) are a matter of future work.

REFERENCES

[1] Foster, I.—Kesselman, C.—Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. The International Journal of Supercomputer Applications, 153, 2001.

[2] Foster, I.—Kesselman, C.: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputer Applications and High Performance Computing, Vol. 11, 1997, No. 2, pp. 115–128.

[3] Natrajan, A.—Humphrey, M. A.—Grimshaw, A. S.: Grids: Harnessing Geographically-Seperated Resources in a Multi-Organisationel Context. In 15th Annual International Symposium on High Performance Computing Systems and Applications, 2001.

[4] Grid2: Blueprint for a New Computing Infrastructure. Eds. I. Foster and C. Kesselman. Morgan Kaufmann Publishers, 2004.

[5] Grid Computing. Eds. F. Berman, G. C. Fox, A. J. G. Hey., J. Wiley, 2003.

[6] Sunderam, V.—Kurzyniec, D.: Lightweight Self-Organizing Frameworks for Metacomputing. In 11th International Symposium on High Performance Distributed Computing, 2002.

[7] Juhasz, Z.—Andics, A.—Pota, S.: JM: A Jini Framework for Global Computing. IEEE International Symposium on Cluster Computing and the Grid, 2002.

[8] Foster, I.—Kesselman, C.—Nick, J.—Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, 2002.

[9] Walsh, A. E.: UDDI, SOAP and WSDL: The Web Services Specification Reference Book. Prentice Hall, 2002.

[10] Open Grid Service Infrastructure (OGSI): `http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf`.

[11] Globus Toolkit 3: `http://www-unix.globus.org/toolkit/download.html`.

[12] Stuer, G.—Sunderam, V.—Broeckhove, J.: Towards OGSA Compatibility in Alternative Metacomputing Frameworks. Lecture Notes in Computer Science 3036, p. 51–58.

[13] Kurzyniec, D.—Wrzosek, T.—Drzewiecki, D.—Sunderam, V.: Towards Self-Organizing Distributed Computing Frameworks: The H2O Approach. In Parallel Processing Letters, Vol. 13, 2003, No. 2, pp. 273–290.

[14] The H2O Project Home Page: `http://www.mathcs.emory.edu/dcl/h2o/`.

[15] The RMIX Project Home Page: `http://www.mathcs.emory.edu/dcl/rmix/`.

[16] Apache UDDI Registry. `http://ws.apache.org/juddi/`.

[17] Sun's Java Web Services Developer Pack. `http://java.sun.com/webservices/downloads/webservicespack.html`.

[18] The Java API for XML Registries. `http://java.sun.com/xml/jaxr/overview.html`.

[19] UDDI4J Project. `http://www.uddi4j.org`.

[20] Sun's Java Web Services Developer Pack – Release Notes `http://java.sun.com/webservices/docs/1.3/ReleaseNotes.html`

[21] Web Services Resource Framework. `http://www.globus.org/wsrf/`.

Gunther Stuer is a post-doctoral fellow in the Department of Mathematics and Computer Science at the University of Antwerp (UA), Belgium. His research interests include network protocol design, Java lightweight grid systems and distributed computing. In 2003 he received his PhD. in Computer Science, at the University of Antwerp (UA), Belgium.

Vaidy Sunderam is a professor of computer science at Emory University. His research interests are in wireless networks and mobile computing systems, parallel and distributed processing systems, and infrastructures for collaborative computing. His prior and current research efforts have focused on system architectures and implementations for mobile computing middleware, collaboration tools, and heterogeneous metacomputing, including the PVM system and several other frameworks such as IceT, H2O, and Harness. Professor Sunderam teaches computer science at the beginning, advanced, and graduate levels, and advises graduate theses in the area of computer systems.

Jan Broeckhove is a professor in the Department of Mathematics and Computer Science at the University of Antwerp (UA), Belgium. His research interests include computational science, quantum physics and distributed computing. In 1982 he received his PhD. in Physics, at the Free University of Brussels (VUB), Belgium.