Grid-Based PDE.Mart: A PDE-Oriented PSE for Grid Computing*

Guoyong Mao
School of Computer Science and Engineering
Shanghai University, Shanghai 200072, China
gymao@mail.shu.edu.cn

Mo Mu
Department of Mathematics,
Hong Kong University of Science & Technology
Clearwater Bay, KL, Hong Kong
mamu@ust.hk

Wu Zhang
School of Computer Science and Engineering
Shanghai University, Shanghai 200072, China
zhang@staff.shu.edu.cn

Xiaobin Zhang
School of Computer Science and Engineering
Shanghai University, Shanghai 200072, China
zxb0412@163.com

Abstract

We investigate the migration of network-based PDE.Mart to the grid platform. The grid architecture of PDE.Mart is presented. Grid-based PDE.Mart is a service-oriented system, and is implemented based on Apache AXIS and Tomcat web service server. Implementation issues are discussed in detail.

1 Introduction

PDE.Mart [10] is a problem-solving environment (PSE) for solving partial differential equations (PDEs) in numerical simulations, academic research, as well as educational settings. Network-based PDE.Mart is a client-server protocol consisting of a web-browser-enabled graphical user interface, PDE-GUI, that runs on client machines to manage the server connection, geometric and model specifications, computational method selection, and post processing; a server system, PDE-Server, to build computational engines and provide PDE solution services on the host machine; and a library, PDE-LIB, that contains building blocks for developing network-based and PDE-oriented PSEs.

The grid-based PDE.Mart project investigates the migration of network-based PDE.Mart to the grid platform. Grid-based PDE.Mart is a service-oriented system [5][7]. It solves a user-supplied PDE problem by using PDE solving services available on distributed grid nodes instead of a single server. The PDE solving objects in PDE-LIB of network-based PDE.Mart are converted to corresponding services and deployed to different grid nodes [8]. Therefore, grid-based PDE.Mart extends network-based PDE.Mart from object integration to service integration.

*Work supported in part by SEC E-Institute: Shanghai High Institutions Grid project and Hong Kong RGC Competitive Earmarked Research Grant HKUST6111/02P.

2 Network-based PDE.Mart

Recall that network-based PDE.Mart consists of three major components: PDE-GUI, PDE-Server, and PDE-LIB. PDE-Server is a Java application running on the PDE.Mart host machine. It serves multiple network clients by concurrent threads and communicates with the instances of PDE-GUI running on client machines via Java sockets. For each socket connection, a computational session is created that exchanges various types of information with the instance of PDE-GUI interactively. A computational engine is built by assembling the PDE solving objects according to instructions from the user. PDE-Server and PDE-GUI share objects in PDE-LIB as building blocks.

The major components involved in a typical PDE computation are illustrated by a flow chart as shown in Figure 1. The system design is fully object-oriented. The computational process starts with a geometric domain and a PDE model, goes through a mesh, a discrete system, a reordering index mapping, and finally arrives at a solution. At each step, a numerical procedure shown as a box, such as mesh generation, discretization, indexing, and algebraic solution, is executed to generate an output state shown as a rectangle by taking one or more previous states as the input. All of the components in the flow chart are treated as objects and classified as descriptive objects and processing objects with the former corresponding to the rectangles

and the latter corresponding to the boxes, respectively. Abstractly speaking, a descriptive object encapsulates all the information necessary for describing the characteristics and behavior of the object such that other objects can use the encapsulated information in a typical PDE-oriented computation. On the other hand, a processing object implements a numerical procedure that takes a list of descriptive objects as the input and generates a target descriptive object. The Java classes implementing these objects are called descriptive and processing classes, respectively.

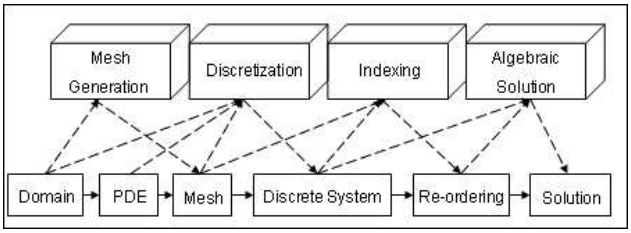

Figure 1. Flow chart for a typical PDE computation consisting descriptive and processing objects

The overall structure of network-based PDE-Server is shown in Figure 2. The server class is mainly responsible for managing the client-server protocol. Once running, it keeps listening to clients on the network to provide PDE solution services. Upon each client's connection, a socket is created on the server to establish the private communication channel between the client and the server. The engine builder is then invoked to create a computational session for building the client's own computational engine.

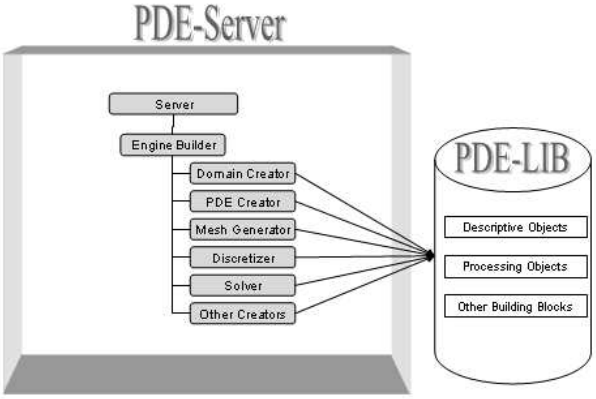

Figure 2. Structure of network-based PDE-Server

The engine builder class is responsible for creating a computational session. Computational sessions are multi-threaded and concurrent. Each client owns an individual computational session that communicates with the instance of PDE-GUI through the assigned socket. The major components of the engine builder include Domain Creator, PDE Creator, Nonlinear PDE Solution Creator, Mesh Generator, Discretizer, Indexer, Solver, and Blackbox Creator. The invocation of these creators is interactively controlled by the client via a request query. A creator further invokes an appropriate numerical procedure from the collection of processing objects in PDE-LIB to create a PDE solving object as requested by the user.

3 Grid-Based PDE.Mart

Note that in a grid environment, the processing objects are usually distributed among different grid nodes, although some of them might reside in the local node that hosts PDE-Server. Therefore, PDE-LIB now turns to be a virtual library in grid-based PDE.Mart. Correspondingly, those processing objects available on the remote nodes are converted to services. To make use of the virtual library, a centralized agent system PDEServiceAgent is introduced, where a resource database is maintained for all the registered, both local and remote, processing objects. With the help of the agent system, only the creators in PDE-Server of the network-based version have to be converted to the corresponding grid-based counterparts, and for simplicity we keep the names unchanged for all these creators. Each invocation to a numerical procedure in a creator is now directed to the centralized agent system. The agent then determines whether and how a remote service or a local procedure is invoked in order to generate the user-requested target object, receives the generated object from the service if it is a remote invocation, and finally returns the object to the requesting creator. The overall structure of grid-based PDE-Server is shown in Figure 3.

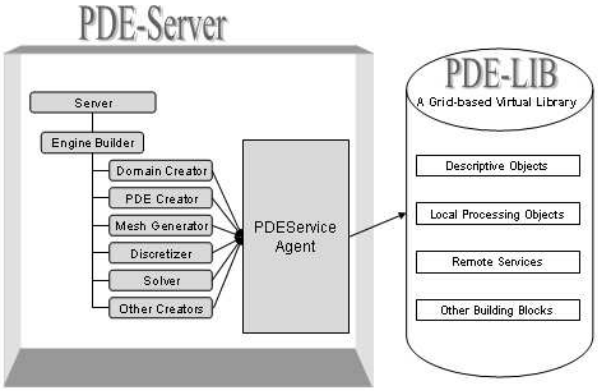

Figure 3. Structure of grid-based PDE-Server

4 Implementation

Web service [2] is one of the most popular distributed technologies to support service-oriented software development. The web services technology allows applications to communicate with each other in a platform- and programming language-independent manner. A web service is a software interface that describes a collection of operations that can be accessed over the network through standardized XML messaging. It uses protocols based on the XML language to describe an operation to execute or data to exchange with another web service. A group of web services interacting together in this manner defines a particular web service application in a Service-Oriented Architecture (SOA).

Conceptually, web services can be viewed as units of work, each handling a specific functional task. The web services architecture permits the development of web services that encapsulate all levels of functionality. The architecture also allows multiple web services to be combined to create new functionality. A web service application can then be created by invoking a series of web services through remote procedure calls (RPC) or a messaging service to provide some or most of the application's logic. A published web service describes itself so that developers can locate the service and evaluate its suitability for their needs.

The web services technology uses SOAP (Simple Object Access Protocol) [9] for the XML payload and uses a transport such as HTTP to carry the SOAP messages back and forth. SOAP messages are actually XML documents that are sent between a web service and the calling application.

We use Apache AXIS [3] and Tomcat web service server [4] to implement grid-based PDE-Server. Apache AXIS is an implementation of the SOAP submission to W3C. It is the third generation of Apache SOAP.

Apache AXIS supports the Web Services Description Language (WSDL) [6], which allows users to easily build stubs to access remote services, and to automatically export machine-readable descriptions of deployed services from AXIS. A WSDL document, written in XML, describes a web service and how to call it. A WSDL document exposes the web service's method signature, protocol to be used, network address, and data format. Apache AXIS offers the JAVA2WSDL tool to generate the WSDL document for a java class and the WSDL2JAVA tool for the code generation for the server-side and application-side implementation [1].

Furthermore, Apache AXIS supports all basic types, and a type mapping system for defining new serializers and deserializers. It also allows automatic serialization/deserialization of Java Beans, including customizable mapping of fields to XML elements/attributes. Therefore, the Apache AXIS implementation can still keep grid-based PDE.Mart object-oriented, where all the arguments in a service invocation are encapsulated descriptive objects.

We now describe the implementation of grid-based PDE.Mart in more detail.

4.1 Generation of WSDL document for a JAVA class

Suppose we would like to convert a Java class FivePSDiscretizer in PDE-LIB, as shown in Figure 4, to a web service. This processing class contains a method discretizePDE that implements a numerical procedure Five-Point-Star for discretizing an elliptic PDE. The method takes a list of input arguments and returns an object of type BeanDiscretization as the generated discrete algebraic system.

```
package Disserviceclient;
public class FivePSDiscretizer {
    public BeanDiscretization dis;
    ......
    public FivePSDiscretizer() {}
    public BeanDiscretization discretizePDE(BeanGlobalControl gc,
                BeanMesh mes,BeanNonrectdom nrdom,
                String  pdestring, boolean isdomrect )
    {
      ......
      return dis;
    }}
```

Figure 4. Structure of FivePSDiscretizer

JAVA2WSDL takes this class as the input and generates a WSDL document that describes the operation of the service, including its name, arguments from the request, and output from the response. The following segment shows the corresponding part in the generated WSDL document.

```
<wsdl:message
name="discretizePDERequest">
<wsdl:part name="in0"
type="impl:BeanGlobalControl" />
<wsdl:part name="in1"
type="impl:BeanMesh" />
<wsdl:part name="in2"
type="impl:BeanNonrectdom" />
<wsdl:part name="in3"
type="soapenc:string" />
<wsdl:part name="in4"
type="xsd:boolean" />
</wsdl:message>
<wsdl:message
name="discretizePDEResponse">
<wsdl:part name="discretizePDEReturn"
type="impl:BeanDiscretization"/>
</wsdl:message>
<wsdl:portType
```

```
name="FivePSDiscretizer">
<wsdl:operation name="discretizePDE"
parameterOrder="in0 in1 in2 in3 in4">
<wsdl:input
message="impl:discretizePDERequest"
name="discretizePDERequest"/>
<wsdl:output
message="impl:discretizePDEResponse"
name="discretizePDEResponse"/>
</wsdl:operation>
</wsdl:portType>
```

Note that AXIS follows the XML Schema standards to provide automatic serialization for all basic data types. User-defined serializer and deserializer can also be used with the extension interface of AXIS. However, one cannot send arbitrary Java objects over the wire and expect them to be understood at the far end. AXIS only sends objects for which there is a registered AXIS serializer. Therefore, we use BeanSerializer to serialize any class that follows the JavaBean pattern of accessor and mutator. For class Discretization, for instance, we actually generate a corresponding bean class BeanDiscretization, which only contains variables, getter and setter methods.

Furthermore, we often need to pass to a service an object, such as a domain and PDE object, which contains methods that are not simply getter and setter. This is particularly the case where the service needs to invoke certain methods of a Java object in the argument list. In this case, the application cannot pass such an object to the service simply by a corresponding bean class. In the above example, the Five-Point-Star discretization service needs to use certain methods of the PDE object from the application side. For this purpose, we currently pass the defining parameters via a key-value variable pdeinputline to the service and then use it to rebuild the PDE object on the service side.

4.2 Deploying a service on a grid node

For deploying a service, say FivePSDiscretizer, on a grid node, by using the WSDL document FivePSDiscretizer.wsdl above, the WSDL2JAVA tool generates the corresponding WSDD (Web Services Deployment Descriptor) files deploy.wsdd and undeploy.wsdd with the following command:

```
java org.apache.axis.wsdl.WSDL2Java
--server-side  FivePSDiscretizer.wsdl
```

These two WSDD files are used for deploying and undeploying the corresponding service on the grid node, respectively. Note that WSDL2JAVA can also be used more conveniently via a graphic development tool such as JBUILDER2005. By copying FivePSDiscretizer.class and all the related classes to the service destination, the service can now be deployed on the grid node with the following command:

```
java org.apache.axis.client.AdminClient
deploy.wsdd
```

Note in particular that the generated WSDD file deploy.wsdd describes the deploy information of the implementation class, and contains extra meta data describing the operations and parameters of the implementation class. Here is part of the WSDD file for FivePSDiscretizer:

```
<service name="FivePSDiscretizer"
type=""
regenerateElement="true"
provider="java:RPC" style="rpc"
use="encoded" validate="true">
<parameter name="scope"
value="Request"
regenerateElement="false"/>
<parameter name="className"
value="Disserviceclient.FivePSDiscretizer"
regenerateElement="false"/>
<parameter name="allowedMethods"
value="*"
regenerateElement="false"/>
<namespace>http://Disserviceclient/
</namespace>
<typeMapping
encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
qname="ns4:BeanDiscretization"
languageSpecificType=
"java:Disserviceclient.BeanDiscretization"
serializer=
"org.apache.axis.encoding.ser.
BeanSerializerFactory"
deserializer=
"org.apache.axis.encoding.ser.
BeanDeserializerFactory"
name="BeanDiscretization"
regenerateElement="true"
xmlns:ns4="http://Disserviceclient/"/>
```

It is seen that the serialization and deserialization of bean object BeanDiscretization follow the standards of BeanSerializerFactory and BeanDeserializerFactory.

The virtual library PDE-LIB is then created by deploying a collection of web services on distributed grid nodes.

4.3 Application side implementation

To consume a web service in grid-based PDE-LIB, on the other hand, several Java classes must also be generated

on the application side, which is again done with the help of the WSDL2JAVA tool. More specifically, WSDL2JAVA takes a WSDL file, say, FivePSDiscretizer.wsdl and the corresponding URL information for where the service is deployed, and generates the following four programs:

```
FivePSDiscretizerSoapBindingStub.java
FivePSDiscretizerServiceLocator.java
FivePSDiscretizerService.java
FivePSDiscretizerPort.java
```

Note that an application program does not instantiate a stub directly. It instead instantiates a service locator and calls a get method which returns a stub. The Service Definition Interface (SDI) is an interface derived from a WSDL's portType. This interface is used to access the operations on the service. A stub class implements the SDI. It contains the code that turns the method invocations into SOAP calls using the AXIS Service and Call objects. It stands in as a proxy for the remote service, letting the application program call it exactly as if it were a local object. In other words, users do not need to deal with the endpoint URL, namespace, or parameter arrays that are involved in dynamic invocation via the AXIS Service and Call objects. The stub hides all that work for users.

This application program is the centralized agent system PDEServiceAgent. It accepts service requests from various creators in PDE-Server; instantiates appropriate service locators according to the resource database, receives service responses, and finally pass them back to the requesting creators.

4.4 Workflow of Grid-based PDE-Server

To develop and consume web services, JAX-RPC (the Java API for XML-based Remote Procedure Call) is needed. JAX-RPC uses an XML-messaging protocol, such as SOAP, to transmit a remote procedure call over a network. Specifically, a web service receives a SOAP HTTP request containing a method call from the client. Using JAX-RPC, the service extracts the method call from the SOAP message, translates it into a method call, and invokes it. Then the service uses JAX-RPC to convert the method response back into SOAP and send the results back to the client. The client receives the SOAP message and uses JAX-RPC to translate it.

To call a service and get the result, all input parameters are serialized into the XML format at the client side and passed to the server via HTTP. On the server side, the XML document is deserialized to retrieve the original information. The serialization/deserialization procedure is similar for the output from the service. The application-service workflow for a web service invocation is shown in Figure 5.

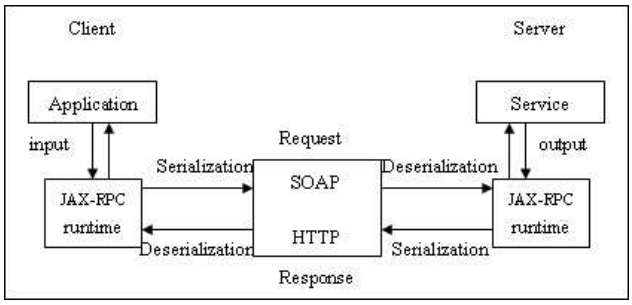

Figure 5. The application-service workflow for a web service invocation

For illustration, Figure 6 and 7 show the corresponding request and response information for a service invocation to FivePSDiscretizer as displayed by the SOAP monitor tool.

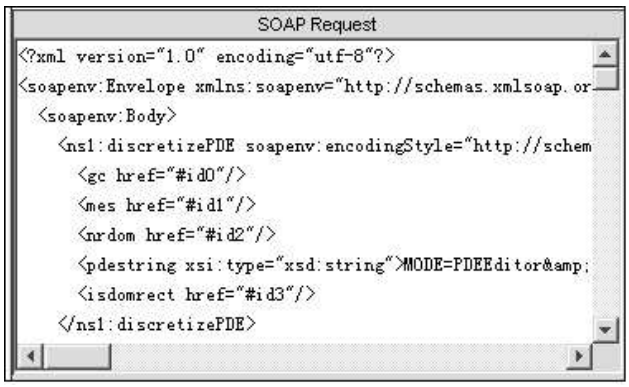

Figure 6. SOAP request information

5 Conclusions

In this paper, we analyze the system architecture of network-based PDE.Mart and propose the architecture of grid-based PDE.Mart. The service-oriented system is implemented based on Apache AXIS and Tomcat web service server. Related implementation issues are discussed in detail. The system is still under further development and evolution.

References

[1] The jakarta project, ant—A java-based build tool. http://jakata.apache.org/ant/.

[2] Soap and web services. http://www-136.ibm.com/developerworks/webservices.

[3] The apache AXIS project. http://ws.apache.org/axis/.

[4] The apache jakarta project. http://jakarta.apache.org/.

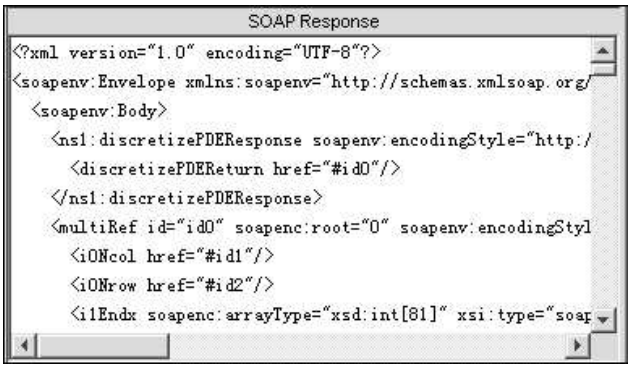

Figure 7. SOAP response information

[5] M. Cannatara, C. Comito, and A. Conguista. Grid-based PSE toolkits for multidisciplinary applications. 2003. FIRB "Grid.it" WP8 Working Paper.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. W3C note 15. 2001. http://www.w3.org/TR/wsdl.

[7] I. Foster, C. Kesselman, J. M.Nick, and T. Steven. Grid services for distributed system integration. *IEEE computer*, 35(6):37–46, 2002.

[8] I. Foster, C. Kesselman, J. M.Nick, and S. Tuecke. The physiology of the grid and open grid services architecture for distributed systems integration. 2002. http://www.globus.org/research/papers/OGSA.pdf.

[9] M. Gudgin, M. Hadley, J.-J. Moreau, and H. F. Nielson. SOAP version 1.2, W3C working draft 9. 2001. http://www.w3c.org/TR/SOAP12/.

[10] M. Mu. PDE.Mart: A network-based problem solving environment for PDEs. *ACM Trans. Mathematical Software*, 31(4), 2005.