Exposing MPI Applications as Grid Services

E. Floros and Y. Cotronis

National and Kapodistrian University of Athens,
Department of Informatics and Telecommunications
{floros,cotronis}@di.uoa.gr

Abstract. This paper presents a Grid Services programming framework for the virtualization and composition of mesh-based high performance MPI applications and their interaction with other Grid Services. Applications are abstracted using a Uses / Provides port scheme where ports represent access points to data quantities. Quantities are modeled using Service Data Elements and Grid Service Handles. Clients can query services based on provides quantities and subscribe to related notification sources. Service clients execute MPI applications through a customized application management service, passing requirements regarding the mesh-topology and the execution environment. The framework defines and provides reference implementations of core portTypes used to instantiate and control the execution of a parallel application. Finally services can be composed using their Uses / Provides Quantities based on service workflow descriptions.

1 Introduction

The advent of grid computing has stimulated development of a new breed of applications targeted for deployment in highly distributed and heterogeneous computing platforms, exploiting disperse computational resources. Nevertheless practice has proved that Grids are hard to program and currently a universal Grid programming model remains a highly desired goal. While it may be possible to build grid applications using established programming tools, they are not particularly well-suited to effectively manage flexible composition or deal with heterogeneous hierarchies of machines, data and networks with heterogeneous performance [8].

Recently there is a shift of grid programming towards the Service-oriented paradigm for application development. The Open Grid Services Architecture (OGSA) [3] leverages Web Service technologies and introduces the notion of Grid Services. A Grid service is a Web service that conforms to a set of conventions relating to its interface definitions and behaviors [13].

An area of interest for applying Grid Services is the virtualization of legacy high-performance applications. Virtualization is a common approach for exposing and extending the functionality of software assets. Various approaches have been introduced applying primarily Object Based and Service Oriented technologies. Among them Common Component Architecture (CCA) defines a component model tailored towards high performance applications [1]. XCAT [10] extends CCA by introducing a Web Services based framework extensively utilizing XML and outlining Grid-oriented Application Factories. XCAT is a predecessor to OGSA providing different

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 436–443, 2004.

service functionality and semantics. Pardis [6] and GridCCM [12] propose CORBA-based frameworks and extensions to IDL and ORB. In these frameworks component stubs are used to implement parallel servers and clients, each instance corresponding to a single MPI processes.

One class of such legacy applications is the high performance parallel applications developed using MPI [11] and in particular mesh based simulation models like Meteorological, Hydrological, Pollution, Fire Propagation etc. Such models are usually developed as isolated MPI applications, typically applying the SPMD programming paradigm. Data processed form 2D or 3D meshes which map directly to real-world coordinates. Models most of the times can benefit from their interoperation (for instance a Hydrological model can interact with a Meteorological model) to produce more accurate results. The composition of these applications is not a straightforward process since source codes are developed by separate teams which are difficult or unwilling to cooperate. As a result ad-hoc approaches are followed.

In this paper, MPI applications are exposed as OGSA Grid Services. The virtualization of an MPI application with Grid Service semantics and tools introduces new potentials since there are obvious benefits both in terms of functionality and interoperability. Programmers can use the well defined value-added infrastructure of Grid Services to search for applications (e.g. in UDDI repositories), retrieve formal descriptions in a standard defined XML (WSDL) document and easily bind their functionality to diverse clients. These client applications can be developed in various languages relieving the programmer from the burden of understanding the inner engineering of the application or even of the MPI message-passing semantics per se. Diverse clients may leverage the capabilities of high performance applications and utilize high-end expert code which till now was isolated due to inherent complexity of MPI and lack of high-level composition semantics. Moreover multiple MPI services can be put together using workflow-based composition, in order to exchange data and interoperate.

The rest of the paper is organized as follows. Section 2 presents the proposed MPI application virtualization model and introduces the notion of Uses and Provides Quantities. Section 3 describes the inner details of the framework and provides examples of simple client-service interaction and more complex, workflow-like service composition. Finally section 4 provides conclusions and future directions of this research.

2 Virtualization of MPI Applications

To demonstrate the framework we use a real life application scenario based on Flood Forecasting [5]. In this scenario three different models cooperate, in a cascading pattern, to execute a flood crisis application (Fig. 1). A Meteorological Model (parallel) provides quantitative precipitation forecasts to a Hydrological simulation (sequential) which in turn feeds a Hydraulic (parallel) simulation with Hydrographs. The three models interoperate to provide weather forecasts, discharge forecasts and flood scenarios.

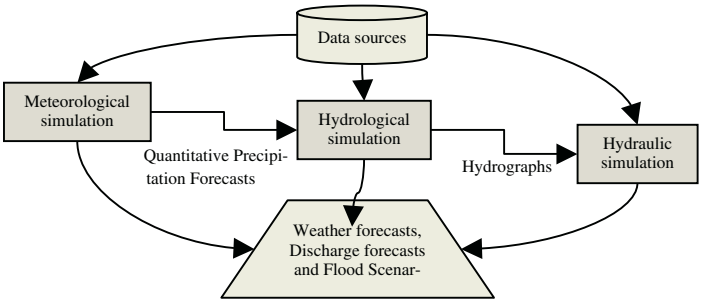

Fig. 1. Sample scenario with three cascading simulation models

2.1 Data-Centric Virtualization

In order to expose a parallel MPI application as a Grid Service one has to bridge two different distributed programming worlds: the service-oriented, operation-centric world of OGSA Grid Services and the two-sided communications world of MPI. The common denominator for these two approaches is the data that are being exchanged between two communication parties. In service-oriented and object-oriented approaches like OGSA, Web Services, CORBA etc., data are bound to operations that receive them as method parameters and return them as results of method execution. In MPI, messages do not trigger explicit method calls but carry data and also act as synchronization mechanism.

We can follow two approaches to wrap a Grid Service around the MPI application: wrapping of processes or wrapping of data. In the first approach the Grid Services runtime interacts directly with the processes that comprise the MPI application. This requires the Grid Service processes or at least a subset of them, to immerse inside the MPI context that the MPI processes define and be an MPI program itself.

A more general approach is to have the Grid Service directly access the output data that a given MPI application produces. In this approach the implementation of the service is independent of the implementation and execution details of the underlying MPI system. In this paper we have followed the later approach since not only is the more general one but also because the current implementations of OGSA support mainly Java based servers, which cannot participate in an MPI communication world.

We abstract an MPI application by defining the following: The application implements an algorithm (e.g. a Meteorological or Hydraulic model) that accepts as input a set of external data in the form of *uses quantities* executes and produces a set of result data in the form of *provides quantities*. For instance Precipitation is a *Provides Quantity* of the Meteorological model and a *Uses Quantity* of the Hydrological model.

The application may also utilize a set of meta-data that define execution requirements of the algorithm and other behavioral aspects and execution details. *Uses* and *Provides Quantities* may be available in various forms and mediums: a simple binary or text file in the local file system, a remote file replicated in various hosts and controlled by a replication service, a database in a network available RDBMS or a networked server application that provides or consumes data by exchanging them in a message-passing fashion.

Following the above approach a Grid service can describe public interfaces of the MPI application by exposing the uses and provides quantities of the application, together with the required meta-data for execution. The implicit operation that the MPI application implements, and is being exposed as a Grid Service operation is the execution of the algorithm.

2.2 Modeling of Provides and Uses Quantities

We model *Provides Quantities* using Service Data Elements (SDEs). Service Data is a structured collection of information that is associated to a Grid Service. This information is easy to query, so that Grid Services can be classified and indexed according to their Provides Quantities. Although originally intended to provide attributes and meta-information of a Grid Service, the SDE conception is in accordance with the Provides Quantity notion.

For example in the Flood crisis scenario the Meteorological model may provide a Precipitation Quantity which is comprised by precipitation related information on a specific 2D x-y axis. The *Precipitation* SDE will have the following simplified XML Schema definition (XSD):

```
<complexType name="Precipitation">
  <complexContent>
    <extension base=”Quantity2D”>
      <sequence>
       <element name="value" type="float" maxOccurs=”unbounded”/>
       <element name=”time” type=”float” maxOccurs=”unbounded”/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

The above defines a Provides Quantity named “Precipitation” which extends a framework defined Quantity2D SDE with two additional fields: the *value* of precipitation and the relevant *time* step. Quantity2D defines an abstract quantity that is enclosed in the mesh rectangle (*xmax,ymax*) and (*xmin,ymin*):

```
<complexType name="Quantity2D">
   <sequence>
      <element name=”xmax” type=”float” maxOccurs=”unbounded”/>
      <element name=”ymax” type=”float” maxOccurs=”unbounded”/>
      <element name=”xmin” type=”float” maxOccurs=”unbounded”/>
      <element name=”ymin” type=”float” maxOccurs=”unbounded”/>
   </sequence>
</complexType>
```

A service may implement a variable number of Provides Quantities. Clients can search for services which provide specific quantities and acquire a reference to them (in the form of a GSH). An implementation code (e.g. a JavaBean if Java is the target language) is generated automatically by the above XML. The service programmer has to customize it and extend it to retrieve and prepare the Quantity information from the application output data. In most cases the MPI application source code should not require any modification to be used from the service. The programmer having knowledge of the result dataset format can derive with little effort the requested data and return them to the client.

A *Uses Quantity* is modelled using a {GSH, QuantityName} pair that uniquely identifies the name of the quantity (e.g. "Precipitation") and the grid service that provides it. For instance the *Precipitation Uses Quantity* of the Hydrological application may have the form: *{http://www.gsmpi.org/models/Meteo, "Precipitation"}*

An application (simple client or another grid service) may define zero or more {GSH,QuantityName} pairs whose values can be set during the preparation of the application or dynamically during the execution time. A null value for a given GSH means that this *Uses Quantity* is not available and the algorithm should perform all required computations without taking advantage of them.

3 Programming Framework

3.1 Standard PortTypes

The framework defines and implements a set of standard portTypes and respective operations, which can be extended by the application developer. These portTypes are:

MPIAppPrepare: Provides the required functionality that handles pre-execution details of the MPI application, such as the setup the execution environment, the definition of the application topology, the definition of special requirements for computational resources (memory, cpu, disk space) etc.

MPIRun: Provides the *MPIRun::mpiRun* operation that is invoked in order to actually start the MPI application. This operation consults the requirements defined previously, prepares the underline MPI system and executes the *mpirun* command. Multiple versions of the operation can be implemented in order to support different MPI environments (e.g. MPICH, LAM etc.).

MPIAppMonitor: Implements basic operations for job monitoring and management. Operations can be used to query the state (*starting*, *running*, *completed* and *aborted*) of the application and control its life-cycle (*kill* or *restart* the application).

3.2 Notifications

The framework extensively utilizes the Notification facilities provided by Grid Services. Clients can register as notification sinks requesting from the service to be informed when a *Provides Quantity* has changed; usually when a computation has finished and the *Provides Quantity* is ready to be retrieved, or when intermediate results are available. When a *Provides Quantity* is changed the event is propagated to all interested parties. Clients are able to pull available results by sending an appropriate request to the SDE. Programmers may also choose to implement a quantity push schema where the service application itself assumes the responsibility to communicate the event together with the relative data to the client application. This ability is especially useful for supporting flexible service composition based on uses / provides quantities.

3.3 Programming in the Framework

The framework is applied as follows: First the programmer writes the *GWSDL* description of the service and the *XSDs* of the Provides Quantities. The service extends one or more of the framework's *portTypes* to generate a custom MPI Application Management Factory. The programmer edits and extends the SDE stubs (which in Globus Toolkit 3 are materialized in the form of JavaBeans) generated from the XSD specifications of the *Provides Quantities*. This is the most important and laborious part of the framework. The programmer has to implement in the JavaBean the data access logic in order to extract and deliver from the complete output data set the quantity information that this SDE provides. The service is then deployed in a Grid Services container (e.g. Apache Tomcat). The stubs generated from the *GWSDL* file are used to implement service clients.

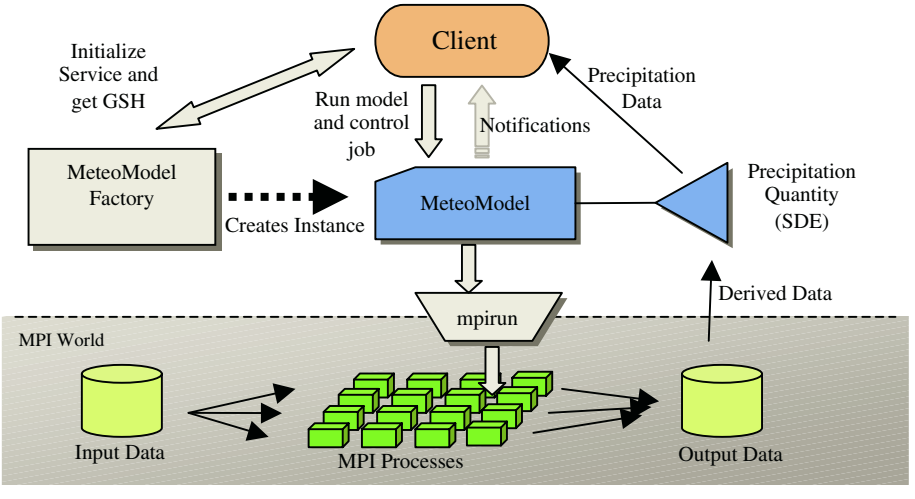

Fig. 2. Overview of Client / Service interaction within the framework

Fig. 2 depicts a sample application execution. A client application uses the *MeteoModelFactory* to instantiate a new *MeteoModel*. Then it passes the initialization parameters of the application. For instance a client may request a 2D dimensional mesh topology within a given range of X and Y coordinates. The client is not concerned with how many processes will be created and where. These details are derived from the requested mesh topology and are handled by the service. Additional requirements may be passed such as a time limit of execution or minimum model error.

The application is executed by calling *MPIRun::mpiRun* operation. Currently the service takes the initial requirements and produces an RSL file to be passed as a parameter to MPICH-G2 *mpirun* script. Issues of security and credentials delegation are handled using GT3 transport level and message level security.

The client may poll periodically the status of the application using *MPIAppMonito::mpiAppGetState* operation or can be notified using Notifications when results are available. To avoid having to extend from all these portTypes mpiRun can use default execution values thus an application can be started by extending only the *MPIRun* portType and issuing a single call to the *MPIRun::mpiRun*.

Moreover a client may request premature end of an application (*MPIAppMonitor::mpiAppKill* operation) either by keeping any intermediate results up to then or by flashing all output rendering them useless in order to start a new simulation.

Since Grid Services can be state-full many clients can dynamically acquire a reference of the running *MeteoModel* instance, connect and retrieve results from the executing model. For example there may be two service clients the first being the Hydrological model and a second visualization client that retrieves and displays graphical precipitation images.

3.4 Service Composition

The described framework facilitates the composition of virtualized MPI applications with other Grid Services (either MPI or non-MPI). Composition can be performed both in space and in time [4]. In the first case the two composed services either have prior knowledge of the *Uses Quantities* each other exposes or the {GSH,QtyName} pair is passed during the execution of the service as a parameter (using for instance a Perl script to instantiate them).

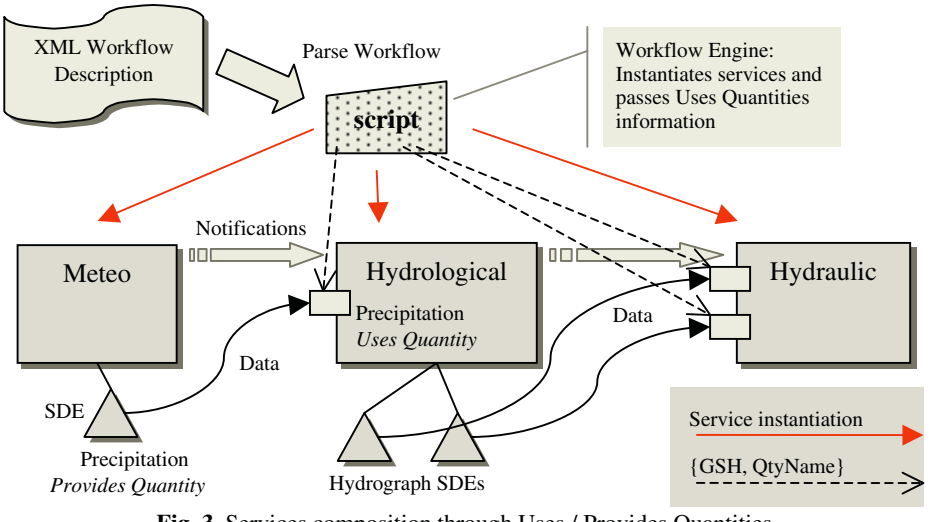

Fig. 3. Services composition through Uses / Provides Quantities

The framework can further be extended to support workflow based composition (composition in time). Currently, workflow systems for Grid and Web Services are evoking a high degree of interest, with initiatives such as WSFL [9], and Grid Services Flow Language (GSFL) [7] investigating the various aspects of workflow in their respective domains. Our approach of Quantities-based composition resembles the GSFL *notificationModel* which is the recommended solution for peer-to-peer, high-performance inter-service bulk data exchange.

In the example (Fig. 3) a simple workflow engine parses a description of the service composition in an XML format, extracts the Uses/Provides Quantities information and instantiates the services by passing the required Uses Quantities information.

4 Conclusions and Future Work

In this paper we have presented a programming framework for exposing high-performance parallel MPI applications as OGSA Grid Services. We have introduced the notion of *Uses* and *Provides Quantities* which are used to describe an abstract application interface, permit data exchange between applications and facilitate the composition of two or more applications in a service workflow.

The framework is currently work-in-progress and is being implemented on top of Globus Toolkit 3 as the Grid Services middleware and MPICH-G2 as the MPI execution environment. The recent refactoring of OGSA that has led to the introduction of the Web Services Resource Framework (WSRF) [2] is pushing for a similar refactoring of all research activities in this area. As a result our imminent steps will be to adapt our framework to WSRF semantics. Moreover, we are evolving the framework at various levels: Application execution semantics are extended to support interactivity and check-pointing. Service composition is enhanced with formal workflow syntax and extended to support semantic information and compatibility assertions. Furthermore, we investigate the capability of Web Services and MPI programs to co-operate at the process level. Finally, we plan to further evolve the dynamic capabilities of the framework especially in the context of dynamic workflow transformations.

References

1. Armstrong R., Gannon D., et al: Towards a Common Component Architecture for High-Performance Scientific Computing. 8th IEEE International Symposium on High Performance Distributed Computation, August 1999.
2. Czajkowski K. et al.: The WS-Resource Framework (WSRF) v1.0, GGF, March 2004.
3. Foster I., Kesselman C., Nick M. 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, June 2002.
4. Gannon D., et al..: Grid Web Services and Application Factories. In: Grid Computing: Making the Global Infrastructure a Reality, p251-p264. Willey, April 2003.
5. Hluchy L. et al.: Problem Solving Environment for Flood Forecasting. 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), July 2003.
6. Keahey K. Gannon D.: PARDIS: A Parallel Approach to CORBA. IEEE 6th International Symposium on High Performance Distributed Computing, August 1997.
7. Krishnan S. Wagstrom P. von Laszeswki G.: GSFL: A Workflow Framework for Grid Services. Argonne National Laboratory, Preprint ANL/MCS-P980-0802, August 2002.
8. Lee C., Talia D.: Grid Programming Models: Current Tools, Issues and Directions. In: Grid Computing: Making the Global Infrastructure a Reality, p555-p576, Wiley, April 2003.
9. Leymann F. Web Services Flow Language (WSFL 1.0), IBM Software Group. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, May 2001.
10. Madhusudhan G., Krishnan S., Slominski A., Merging the CCA Component Model with the OGSI Framework. Proc. of CCGrid2003, May 2003.
11. Message Passing Interface Forum: MPI: A Message Passing Interface Standard, June 1995.
12. Perez C, Priol T., Ribes A.: A Parallel CORBA Component Model, INRIA/RR-4552. September 2002.
13. Tuecke S. et al.: Open Grid Services Infrastructure (OGSI). Version 1.0, Global Grid Forum, June 2003.