

Light-weight Service-oriented Grid Application Toolkit

Sungju Kwon

School of Computing, Soongsil Univ.
1-1 Sangdo-5Dong, Dongjak-Gu,
Seoul, Korea
+82-2-820-0684

lithlife@ssu.ac.kr

Jaeyoung Choi

School of Computing, Soongsil Univ.
1-1 Sangdo-5Dong, Dongjak-Gu,
Seoul, Korea
+82-2-820-0684

choi@ssu.ac.kr

Kumwon Cho

Korea Institute of Science and
Technology Information,
Dajeon, Korea
+82-42-869-0565

ckw@kisti.re.kr

ABSTRACT

Grid has been focused in a distributed computing community. There has been a lot of research in these areas, especially for design and development of Grid middleware. More recently, the service-oriented architecture based on Web Services rapidly became a major issue. The service-oriented architecture provides a modularized functionality to Grid applications. However, this new technology has some limitations. Web Services basically works with the SOAP protocol, but it is not suitable for massive scientific data. In this paper, we propose MAGE, Modular and Adaptive Grid Environment, which is uses dynamically reconfigurable component architecture with interfaces. MAGE provides several level of transparency to the Grid application development, and it can dynamically reconfigure its architecture to adapt to heterogeneous Grid environments.

Keywords

Service-oriented architecture, Grid, Reconfiguration

1. INTRODUCTION

Grid [6] is not a new issue anymore in the computer science community. Following the success of the initial efforts of Globus [8] and Legion [11, 12], the Grid middleware now merges with Web Services technology to become a service-oriented architecture. The Open Grid Services Architecture (OGSA) [13] has been developed for a 'second generation' distributed computing approach to the Grid middleware. This new architecture provides a more unified and simplified approach to the Grid applications. The current issue in the Grid middleware is service-oriented architecture (SOA). This trend is followed by Web Services technology and is well suited for Internet environments. However, the service-oriented architectures like OGSA are still inefficient in many areas of the Grid applications. The service-oriented architecture has three major limitations for Grid applications. First of all, it is not suitable for advanced network services. The service-oriented architecture is based on SOAP protocol. SOAP protocol is a heavy weight protocol for massive scientific data. It is also not a good choice for different kinds of communication channels. Secondly, SOA strictly

depends on the underlining framework. Application developers must know how a service-oriented architecture works and follow these rules. This constraint restricts integration of diverse system elements into Grid applications. Third, because Grid applications are large, complex, and geographically spread, it is very complicated to manage Grid resources, and it requires easy and unified management interfaces.

In this paper, we propose dynamically reconfigurable middleware for Grid environments called MAGE (Modular and Adaptive Grid Environment). MAGE is a reconfigurable framework, which is a base architecture to provide adaptability, high availability, and scalability. MAGE can change components easily and minimize physical dependency among components. MAGE is a service-oriented architecture to support platform transparency, language transparency, and communication protocol transparency. Based on this framework, application developers can build Grid applications more easily.

A component is a unit of composition with well-defined provided and required interfaces [1]. Like an object, a component encapsulates its internal states and provides services to other components or applications. Components are distinguished from objects in that they are explicitly required interfaces and conformance to a binary standard. All interfaces are managed by the component framework. A component framework is defined by Szyperski as "collections of rules and interfaces that govern the interaction of a set of components plugged into them" [1]. A component framework is a reusable architecture which provides a component interface for communicating with each other in binary format. The framework manages the lifecycle of a component. A component hides its internal states and implementation details from clients and allows itself to be accessed only through published interfaces. Therefore, it is possible to hide implementation details among components. A component-based system has an ability to change its internal structure dynamically without disturbing users. An application can be dynamically configured by adding, removing, and modifying components.

Dynamic reconfiguration [2, 3, 4] is a mechanism that can replace a functional unit of an application with another without stopping the application. A dynamic reconfigurable facility is very useful for adaptable and high availability systems [5]. An adaptable system actively deals with environmental changes. A high availability system provides continuous services without stopping the application even if there is a fault. Adaptive systems and high availability systems are useful for Grid [6], ubiquitous [7], and mobile environments. Currently, this dynamic reconfiguration technology is an active research topic in several areas with their own specific formats. These applications have their own

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'06, April, 23-27, 2006, Dijon, France.

Copyright 2006 ACM 1-59593-108-2/06/0004...\$5.00.

reconfiguration technologies, which have difficulties in adapting to new applications. In order to provide the technology in a more generalized and a platform-independent format, a framework is required to manage its components and provide service interfaces among components. Therefore, the framework should be a component-based architecture.

This paper is organized as follows: Section 2 describes related works including a service-oriented architecture and a component-based grid middleware. We explain the MAGE architecture in Section 3, and a service-oriented flow of the MAGE in Section 4. A component management scheme is explained in Section 5. Section 6 shows an example application. Finally, Section 7 includes the conclusion and future work.

2. RELATED WORKS

The Grid systems should be adaptable and highly available. The Grid middleware systems like Globus toolkit have been studied for more efficient resource usage.

Globus toolkit [8] is an open source software toolkit which provides resource sharing on the Grid environments. Globus toolkit provides services and libraries for resource monitoring, resource management, security, and file management. GGF (Global Grid Forum) defines OGSA (Open Grid Services Architecture) as a service-oriented architecture for distributed computing which provides Grid services to support distributed interaction of services and computing infrastructure. OGSA manages a lifecycle of services through service naming, creation, discovery, and destruction. OGSA extends Web services to manage not only permanent services but also temporary services too. However, OGSA has the following problems. First, it is difficult to achieve high speed network performance. The current Web services mainly use SOAP protocol which is not suitable for scientific applications that generate and handle massive data to compute complicated problems. Second, compared with the component-based middleware, OGSA has more constraints on fault tolerance, continuous state management, automated logging, load balancing, and event distribution. Third, OGSA does not support system reconfiguration according to its state.

GridLab [9] project, which is in progress in EU, provides a unified access to the Grid middleware through the service-oriented interfaces. GridLab consists of user space and capability space for hierarchical structures. These two spaces are integrated by GAT (Grid Application Toolkit). GAT abstracts services that are required by the applications. Therefore, GAT can act as an interface between the application programs and the Grid middleware environments. GAT provides the GAT API for the application developers to access different Grid environments in a unified way. The service-oriented architecture of GridLab provides scalability and adaptability for environmental changes. However, services in an application-level, such as GridLab services, are provided on a large scale and have a problem of reusability and adaptability, compared with services in a component-level. Also, the service interfaces of GridLab are statically linked at compile time. This means that GridLab can not reconfigure running middleware at runtime.

GRIDKIT [10] is a component-based middleware framework for configurable and reconfigurable grid computing. It particularly

focuses on the lightweight component-based technology to construct an extensible family of open and programmable overlay networks. GRIDKIT provides four main domains: Service binding, Resource discovery, Resource management, and Grid security. These four domains of middleware functionality are implemented in GRIDKIT as independent and horizontal frameworks, each of which is highly configurable and reconfigurable. MAGE has a similar functionality like GRIDKIT. However, MAGE provides a network transparency with highly customizable components. Components in MAGE provide a rich set of information for component control and installation. MAGE is capable of installing necessary components automatically based on the component dependency and the versioning information.

3. MAGE ARCHITECTURE

MAGE is a Grid application development toolkit. MAGE provides the facilities to build a component-based architecture with the service interfaces. All functions can be accessed with service interfaces. Figure. 1 shows a basic architecture based on MAGE API.

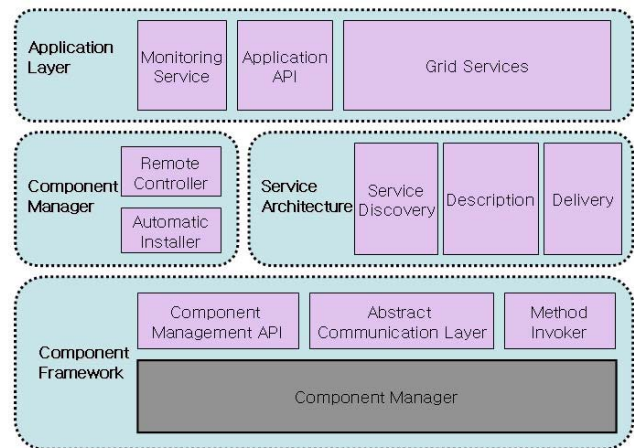


Figure 1. MAGE Architecture

The Component Framework provides a component management facility. All functions must be implemented upon component interfaces that MAGE provides. The components can be loaded dynamically at run-time. The users can control components using a component management API or through a Remote controller within the Component Manager. Every request made for components is delivered by the Component Framework. The Service architecture provides service description, service discovery, and service delivery to each component. Using these two layers, MAGE can provide Grid middle services such as resource management, security, and job management.

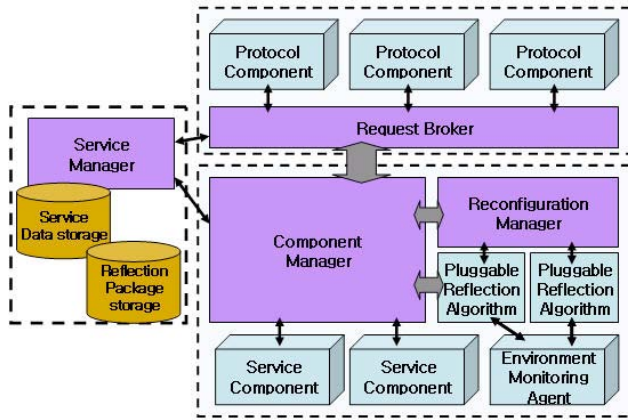


Figure 2. Major elements

MAGE architecture consists of four parts as shown in Figure 2: Request Broker, Component Manager, Service Manager, and Reconfiguration Manager. By using this architecture, MAGE provides transparency with independence of the running platform, the programming language, and the communication protocol. Each element of the architecture has the capability to be located at any location, implemented using any language, and executed on any platform because each element is executed independently from others and communicates with others using the MAGE API.

3.1 Request Broker

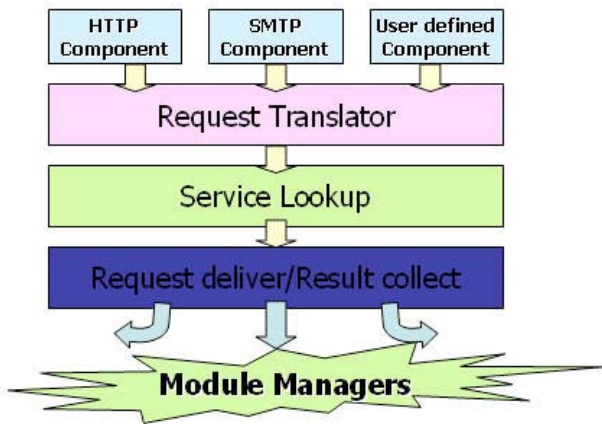


Figure 3. Request Flow inside Request Broker

The Request Broker accepts and translates a client request to an internal message format. The actual communication is performed by a protocol-component inside the Request Broker. MAGE provides the message translation APIs for a protocol-component. A protocol-component can be dynamically installed by a system administrator. Change of a client protocol only replaces a correspondent protocol-component. The Request Broker searches for a matching Component-Manager and delivers user requests to this Component-Manager. Figure 3 shows a request flow within the Request Broker.

3.2 Service Manager

The Service Manager manages a registered service information such as service location, dependency information, and supported operations. The Request Broker and Component Manager use this information to find the required services. The Service Manager provides dependency information of a service to the Component Manager for automatic installation of components.

3.3 Component Manager

The Component Manager maintains a lifecycle of components. It installs components using a descriptor file. The descriptor file contains a service list and a dependency list of components. The Component Manager registers a component information with the Service Manager after completing the installation process. Clients and other services use this information to search appropriate services. A component developer does not need to know about the communication method with the clients. A component only needs to implement the predefined interfaces of MAGE. Actual communications are performed inside MAGE.

3.4 Reconfiguration Manager

The Reconfiguration Manager controls the arrangement of components in the Component Manager according to a reflection algorithm, which can be installed dynamically. A client can install his own reflection algorithm. The reflection algorithm decides configuration of components such as installation, uninstallation, and movement. The Reflection Manager collects that decision and towards it to the Component Manager.

4. SERVICE-ORIENTED INTERFACE

MAGE is a service-oriented architecture based on component framework. A client does not need to know any physical information about the components. A client only needs to know names and operations of the services. This service information does not contain any physical information such as component location, implementation language, and communication method. Even though physical components are moved to other nodes, an application can be executed seamlessly. MAGE provides a common component interface to the developers. MAGE only interacts with components using this interface.

MAGE provides a component interface as shown in Table 1. This component interface consists of the control operations and the service operations.

Table 1. Component Interface of MAGE

Operation	Description
component control operations	
initProvider	called after loading component
startupProvider	called after service starts
stopProvider	called before service stops
destroyProvider	called before unloading component
service operations	
executeMethod	entry point of method invocation
setProperty	entry point of property set
getProperty	entry point of property get

The Component Manager delivers a message to the matching components through the MAGE interface. Each interface must implement actual jobs based on the delivered request messages. The service results go back to the client through MAGE. This approach provides the network neutrality to the component developers. MAGE hides the internal process of a request message from clients and from the components. The client's request message is delivered to the service component through the following steps.

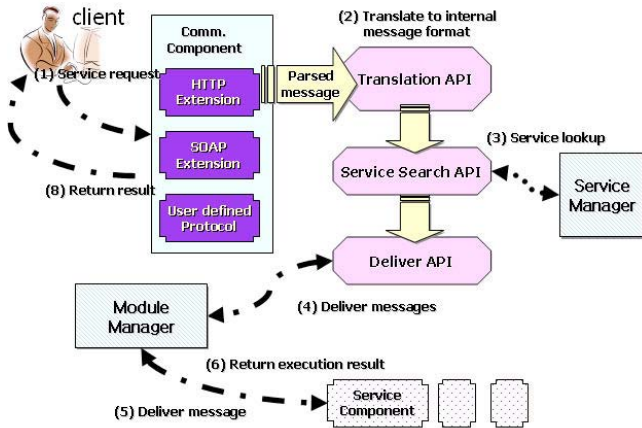


Figure 4. Process Flow of Service

A client sends a request message through the network. MAGE accepts a client request by one of the protocol-components. A protocol-component translates the request message to an internal format using the MAGE translation APIs. The Request Manager searches the corresponding service through the Service Manager. The Request Manager delivers the request message to the Component Manager which contains the appropriate service components. The Component Manager activates the target component and delivers messages. The Component Manager collects the result from the component and sends it back to the Request-Broker. The protocol-component in the Request Broker sends results back to the client.

5. MANAGING THE COMPONENTS

A service component must implement the predefined interfaces of MAGE. A component descriptor file provides information about a component. The major information is listed as follows:

- File name of the Component
- Service name of the Component
- Version information
- Dependence information
- Component activation method
- Service category
- Services list

The Component Manager obtains what kinds of services are provided by each component using component's descriptor file. A descriptor file defines dependency with other components for doing a job correctly. MAGE download other components using this information if necessary. This process will reduce a system

administrator's job. There are two different ways of the component activation.

- Loading at the installation time: MAGE loads the component into the main memory after registering the component with the Service Manager. This method can process client requests faster than loading at the service time.
- Loading at the service time: MAGE does not load the component into the main memory after registering with the Service Manager. Loading will be performed when the client request the services. This approach may be the cause of latency at the first time. However, this method will provide the same performance compared with loading at the installation time once it is loaded. It has a good utilization of memory usage. A service that is not serviced for a long time will be unloaded automatically.

This activation method has no effect on the application functionality. Users can select the most efficient method for a specific job.

6. SIMPLE MONITORING AGENT

Using this framework, we developed the simple monitoring agent as a form of component. This component can gather local system information and response to the client requests. This monitoring agent differs from the others by the automated mobility. The monitoring agent checks system status periodically for conditional movement. The agent can install itself to other systems using the MAGE remote component manipulation API. The agent will install itself to the other node and remove itself from the original node to simulate simple migration simulation.

Figure 5 is an example that shows a remote monitoring. This example also provides a remote control of components. The left side of the user interface shows the component information that is installed at the selected nodes or all nodes. The left upper rectangle displays a list of installed components and the left bottom rectangle shows the detailed information about the selected component. The right side displays the location of an active component. A component moves among nodes by itself, and it provides status information of a node to the client application.

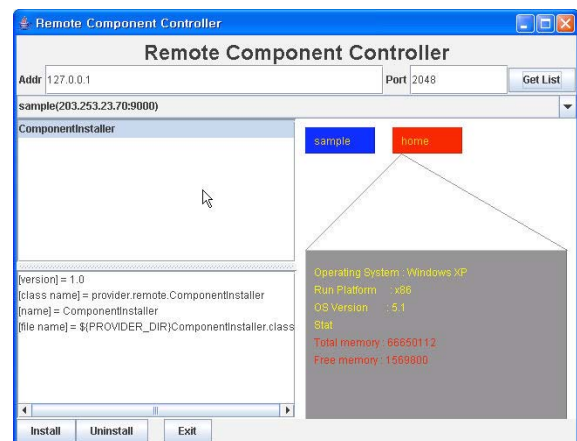


Figure 5. Node Monitoring UI

7. CONCLUSION

In this paper, we introduced a service-oriented architecture based on the component technology. This architecture provides transparency for the running platform, the implementation language, and the communication protocols. It is also easy to reconfigure application architecture without stopping the application.

MAGE has the following characteristics for the Grid environment. First, MAGE is a component-based architecture. An application is composed of functional units called components. Each component works together in MAGE framework and has no information about the physical relationships among the components. Independency among the components provides convenience for application developers and system administrators. Second, MAGE provides transparency for language, platform, and communication protocol. A component does not show any physical information to the clients and provides services with a generalized viewpoint. Third, MAGE is a reconfigurable architecture. A component-based architecture can be reconfigured by itself, by a system administrator, or by a relationship among components at any time. Fourth, MAGE is a service-oriented architecture, which reveals all the functionalities as a service. It hides implementation details from the clients and shows functions as a service with the object-oriented interfaces. Moreover, it controls the life cycle of a service such as service creation, find, operation, and terminations.

In the future, we will focus on the communications transparency for client applications. MAGE will provide a communication component for clients, which is downloadable at runtime before actual communications start.

8. ACKNOWLEDGEMENT

This work is supported by National e-Science Project, the Ministry of Science & Technology (MOST) in Korea.

9. REFERENCES

- [1] C. Szyperski, *Component software: Beyond object-oriented programming*, Addison Wesley, 1998.
- [2] X. Chen and M. Simmons, "Extending RMI to support dynamic reconfiguration of distributed systems," In 22nd International conference on Distributed Computing Systems (ICDC'02), 2002.
- [3] K. M. Goudarzi, *Consistency Preserving Dynamic Reconfiguration of Distributed Systems*, Ph.D. Thesis, Imperial College, 1999.
- [4] J. Hillman and I. Warren, "Quantative analysis of dynamic reconfiguration algorithms," In *Design Analysis and Simulation of Distributed Systems (DASD)*, 2004.
- [5] M. Clarke, G. S. Blair, G. Coulson, and N. Parlavantzas, "An efficient component model for the construction of adaptive middleware," In *IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.
- [6] I. Foster and C. Kesselman, ed., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998.
- [7] Mark Weiser, "Some Computer Science Problems in Ubiquitous Computing," *Communications of the ACM*, July 1993.
- [8] Globus Toolkit, <http://www.globus.org/>
- [9] GridLab, <http://www.gridlab.org/>
- [10] Cai, W., Coulson, G., Grace, P., Blair, G.S., Mathy, L., Yeung, W.K., "The Gridkit Distributed Resource Management Framework," *Proceedings of the European Grid Conference*, Amsterdam, The Netherlands, February 2005.
- [11] Legion, <http://legion.virginia.edu/>
- [12] Anand Natrajan, Anh Nguyen-Tuong, Marty Humphrey, Andrew Grimshaw, "The Legion Grid Portal", *Concurrency and Computation: Practice and Experience*, Grid Computing Environments 2001
- [13] OGSA, <http://www.globus.org/ogsa/>