

Protocol transparent application framework for Grid

Sungju Kwon¹, Jaeyoung Choi¹, Jysoo Lee²

¹*School of Computing, Soongsil University,
1-1 Sangdo-5Dong, Dongjak-Gu, Seoul, 156-743, Korea
{lithlife, choi}@ssu.ac.kr*

²*Supercomputing Center, KISTI, Daejeon, Korea
jysoo@kisti.re.kr*

Abstract

A component is defined as a functional unit with well-defined interfaces. It encapsulates its internal states and provides services to other components or applications. By modularizing required functions into components, a component-based system can easily reuse those components and provide a flexible application structure with dynamic reconfiguration facility. In this paper, we propose a component-based middleware, called MAGE, which uses a service-oriented interface to provide platform transparency, implementation language transparency, and location transparency. The MAGE can dynamically reconfigure its architecture to adapt to Grid environments.

1. Introduction

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 a 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 means of enforcing architectural properties and provides a component interface for communicating with each other in binary format. Also, the framework manages a 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 dependency 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 applications. 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 applications 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 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.

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.

This paper is organized as follows: Section 2 describes related works including a service-oriented architecture and a component-based 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. Finally, Section 6 includes the conclusion and future work.

2. Related Works

2.1. A component-based middleware

A component-based system requires functions such as self-reflection [8] and dynamic reconfiguration to provide flexibility. It is possible to reduce dependencies among functional units by using component technology, therefore it is possible to add, remove, and modify functional units easily. A self-reflective system provides openness and flexibility to a component-based middleware. Self-reflection is a capability in which a system can adapt itself to a failure of its components and changes of its resources. By using reflection, a system can hold its functional list and its current state of components. The system can find a specific component easily, which consists of several functional units, and can construct a service dynamically. A component-based self-reflective middleware provides the ability to react to dynamic changes of environments.

There are several component-based distributed computing environments such as CCA (Common Component Architecture) [9], and CIM/WBEM [10, 11]. CCA is developed in CCA forum and is composed of a framework and components. It defines rules to build a component, and interfaces to send/receive messages among components. CCA provides implementation language independence. However, CCA can not support reconfiguration of components. Although a framework is implemented according to CCA specification, it might not use a component which is developed for other CCA-based platforms since there is no standard interface among different CCA-based frameworks.

CIM (Common Information Model), which is developed by DMTF (Distributed Management Task Force) [12], provides a common definition of management information for generalized modeling of information management for systems, networks,

GridLab [14] project, which is in progress in the EU, provides a unified access to Grid middleware through service-oriented interfaces. GridLab consists

applications, and services. Companies can extend the modeling according to their own necessity. CIM supports compatibility among different platforms with plenty of resource information. CIM uses an object-oriented approach to standardize modeling of resources. CIM/WBEM is a web-based management architecture using CIM as resource modeling. It uses CIM for data representation, XML for data encoding, and HTTP protocol for transferring data. CIM/WBEM is a component-based environment which is constructed with providers. CIM/WBEM provides adaptability by using a component concept and object-oriented service concept. However, it uses an HTTP extension only to communicate with clients, so that it does not provide a communication protocol transparency. The resource access with the abstracted model provides implementation transparency, but its usage is restricted to specific applications like resource management. Current CIM/WBEM software does not provide transparency of implementation language.

2.2. Grid middleware

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

Globus toolkit [13] is an open source software toolkit which provides resource sharing on Grid environments. The 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. However, the OGSA has three 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, it does not support system reconfiguration according to its state.

of user space and capability space for hierarchical structure. These two spaces are integrated by GAT (Grid Application Toolkit). The GAT abstracts

services that are required by applications. Therefore, GAT can act as an interface between application programs and Grid middleware environments. The GAT provides GAT API for 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, GridLab's service interface is statically linked at compile time. This means that it can not change services at runtime.

3. MAGE Architecture

MAGE acts as a component management framework. MAGE does not provide application specific functions. MAGE can be executed as a standalone application or with a legacy Grid middleware through interface components. Fig.1 shows the relationship between MAGE and legacy grid middleware.

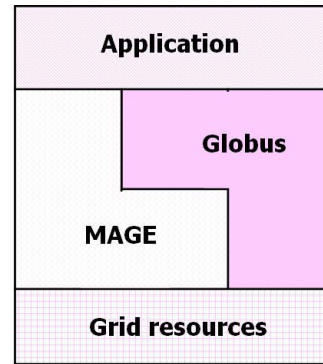


Fig. 1 MAGE and Globus Relationship

MAGE architecture consists of four parts as shown in Fig.2: Request-Broker, Component-Manager, Service-Manager, and Reconfiguration-Manager. By using this architecture, MAGE provides transparency independent of the running platform, programming language, and communication protocol. Each element of the architecture has 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 internal protocols. Each element can exist as a 1:1 relationship or as M:N relationship. This relationship can be arranged by the system administrator. This separation of elements provides flexibility for application framework.

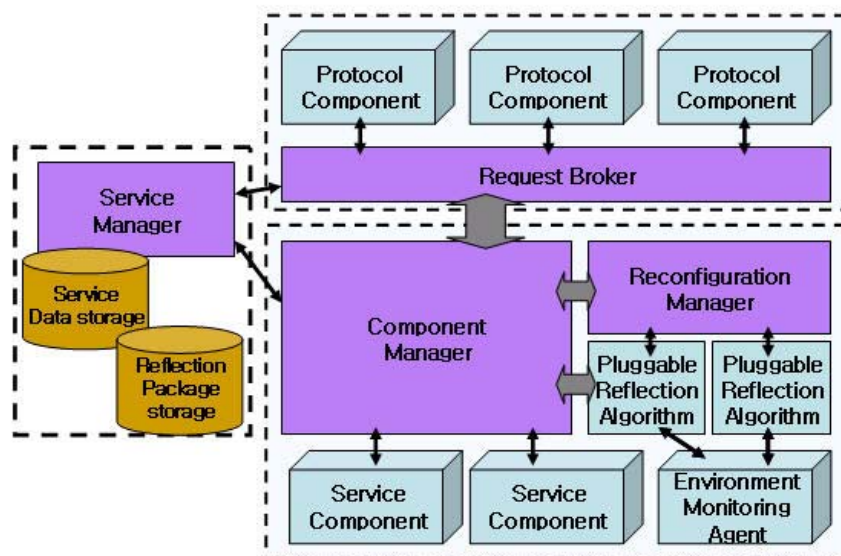


Fig. 2 Reflective MAGE Architecture

3.1. 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 APIs of message translation for a protocol-component. A protocol-component can be dynamically installed by a system administrator. Client can change communication protocol without effecting MAGE services. MAGE only need to change corresponding protocol-component. The Request-Broker searches for a corresponding Component-Manager and delivers user requests to this Component-Manager. Fig. 3 shows a request flow inside the Request-Broker.

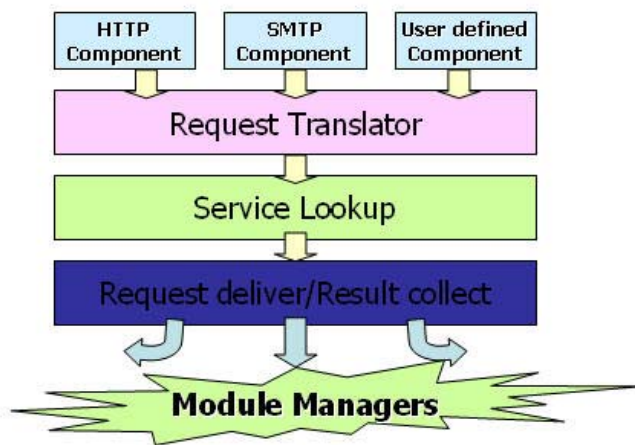


Fig. 3 Request Flow inside Request-Broker

3.2. Service-Manager

The Service-Manager manages registered service information such as service location, dependency information, and supported operations. The Request-Broker and Component-Manager use this information to find appropriate 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. A component can be installed using the descriptor file. The descriptor file contains the service list and dependency list of components. The Component-Manager registers 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 clients. A component only needs to be implemented with predefined interfaces of the MAGE. Actual communications are performed inside of the MAGE.

3.4. Reconfiguration-Manager

The Reconfiguration-Manager arranges components within Component-Manager. A client can install reflection algorithm packages at runtime. The reflection algorithm package consists of one reflection algorithm component and several environment monitoring components. Environment monitoring components collect system information and provide this information to the reflection algorithm component. Using this information, the reflection algorithm component changes the installed component list on the Component-Manager. And users can install several reflection algorithms with priority.

4. Service-oriented Interface

The MAGE is a service-oriented architecture based on component framework. A client does not need to know any physical information about components. A client only needs to know names and operations of 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 another node, an application can be executed seamlessly. The MAGE provides a common component interface to developers. The MAGE only interacts with components using this interface.

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

Table 1. MAGE Component Interface

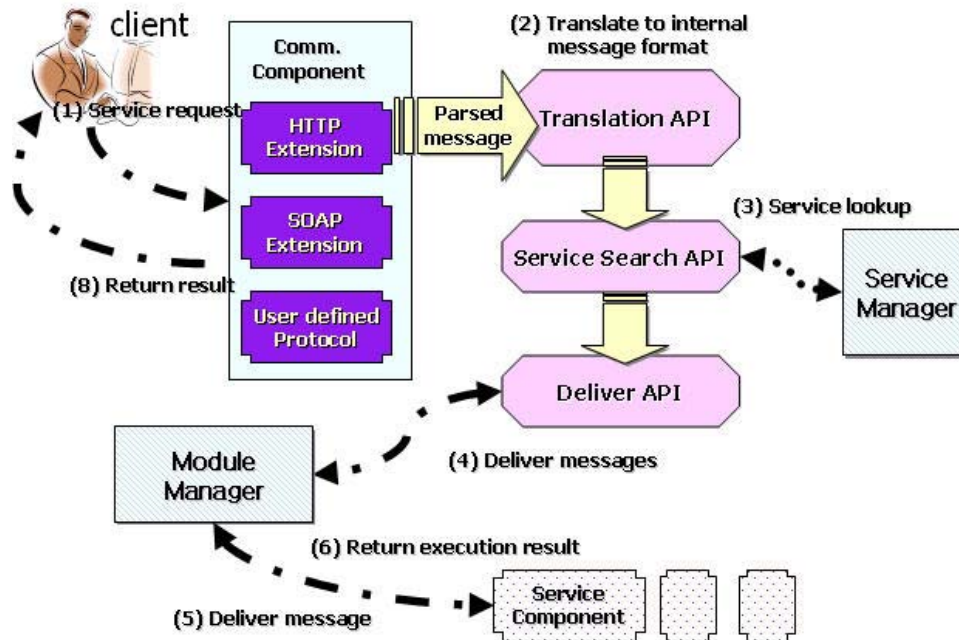


Fig. 4 Process Flow of Service

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

The Component-Manager delivers a message to corresponding components through the MAGE interface. Each interface must implement actual jobs based on delivered request messages. The service results go back to the client through MAGE. This approach provides network neutrality to component developers.

The MAGE hides the internal process of request message from clients and from components. The client's request message is delivered to the service component through following procedures.

1. A client sends a request message through the network. The MAGE accepts a client request by one of the protocol-components.
2. A protocol-component translates the request message to the internal format using MAGE translation APIs.
3. The Request-Manager searches the matching service through the Service-Manager.
4. The Request-Manager delivers the request message to the Component-Manager

which has the requested service component.

5. The Component-Manager activates the target component and delivers messages.
6. The Component-Manager collects the result from the component and sends it back to the Request-Broker.
7. The protocol-component in the Request-Broker sends results back to the client.

5. Management of Service Components

A service component is provided with the predefined interfaces of MAGE. A component descriptor file provides information about a component. Major entities are listed as follows:

- A component file name
- A component service name
- Version information
- Dependence information
- Component activation method
- Service category
- Services list

The Component-Manager understands what kinds of services are provided by each component using its descriptor file. A component can describe a dependent

component list in order to execute its job correctly. The MAGE will download other components using this dependence information. This process will reduce the system administrator's job. There are two different ways of component activation.

Loading at installation time: the MAGE loads the component into the main memory after registering the component with the Service-Manager. This method can more quickly process client requests.

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

A component developer can decide which one is better for the job. This activation method does not affect application functionality at all. It is just a convenient way.

6. Conclusion

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

The MAGE has the following characteristics for the Grid environment. First, the MAGE is a component-based architecture. An application is composed of functional units called components. Every component works together using the MAGE framework and does not know about physical relationships among components. Independency among components provides a convenient way for application developers and system administrators. Second, the MAGE provides transparency about language, platform, and communication protocol. A component does not show any physical information to clients and provides services as a generalized viewpoint. Third, it is a reconfigurable architecture. A component-based architecture can reconfigure itself at any time, can be reconfigured by a system administrator, or by the relationship among components. Fourth, the MAGE is

a service-oriented architecture which reveals all functionality as a service. It hides implementation details from clients and shows functions as a service with object-oriented interfaces. The MAGE controls the life cycle of a service such as service creation, discovery, execution, operation, and terminations.

In the future, we will focus on communications transparency for client applications. The MAGE will provide a communication component for clients and it must be downloadable at runtime before actual communications start.

Acknowledgement

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

8. 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] G. Kiczales, J. Rivieres, and D. Dobrow, The art of the metaobject protocol, In MIT Press, 1991.
- [9] CCA Forum, <http://www.cca-forum.org/>
- [10] CIM Standards, <http://www.dmtf.org/standards/cim>
- [11] WBEM Initiative, <http://www.dmtf.org/standards/wbem>
- [12] DMTF, <http://www.dmtf.org/>

[13] Globus Toolkit, <http://www.globus.org/>

[14] GridLab, <http://www.gridlab.org/>