A Model for Flexible Service Use and Secure Resource Management

Ken'ichi Takahashi[1], Satoshi Amamiya[2], and Makoto Amamiya[2]

[1] Institute of Systems & Information Technologies/KYUSHU,
2-1-22 Momochihama, Sawara-ku, Fukuoka, 814-0001, Japan
takahashi@isit.or.jp
[2] Faculty of Information Science and Electrical Engineering, Kyushu University,
6-1 Kasuga-Koen, Kasuga-shi, Fukuoka 816-8580, Japan
{roger, amamiya}@al.is.kyushu-u.ac.jp

Abstract. Grid computing is promissing as an infrastructure that allows users to use distributed computer resources by simple connecting a computer to the network without special operation; just like connecting to the electricity, water or gas grid. In this paper, regarding resources as services, we propose a new architecture for realizing an environment in which users can use services that are in various locations through their portable terminals. In this architecture, a service is managed by an *agent*, which has two resource management spaces named the *Public Zone* and the *Private Zone*. The Public Zone is a space for realizing flexible public service use. The Private Zone is a space for protecting private user information. Moreover, agents are organized in a group called the *community* and are managed independently in each community. Thus, we realize both of flexible service use while protecting private information.

1 Introduction

Grid computing is promissing as an infrastructure that allows users to use dispersed computer resources by simply connecting a computer to the network with no special operation; just like connecting to the electricity, water or gas grid. SETI@home[5] and distributed.net[1] are the two well-known grid computing projects. These projects try to search for extraterrestrial intelligence or carry out cryptographic analysis by using CPU resources connected to the Internet. In these projects, a resource is a CPU resource. But a resource is not only a CPU resource, but also data and a service. If we regard a resource as a service, we will be able to realize an environment that allows users to use services provided in various locations through their portable terminals. For example, if a user is in the laboratory, he can use the printer and the copy machine in the laboratory through his portable terminal; if he is in his house, he can use the television, and the audio player there and so on. In this way, users will be able to use services which are based on their locations. To realize such an environment, the following functions are required.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1143–1153, 2005.

Service-Use Mechanism. Each service has a method for its use. For example, when we use a telephone, first, we pick up the telephone receiver and put in coins and dial. In a same way, we must get the method for using a service and use the service according to the method appropriate to it.

Protection of Private Resources. In the real world, various services are provided in exchange for resources/information like money and e-mail address. So, when a user receives a service, he may have to provide some private information. But users don't want to unconditionally make their own information public. Therefore, it is necessary to protect their resources/information.

Decentralized Service Management. In this environment, there are countless services. So it is difficult to manage and dispatch resources to users in a centralized way. Therefore, we need a mechanism for managing resources in a group depending on the location and/or other indicators.

The availability of a service-use mechanism depends on the degree of protection of private resources. If a user does not provide all his information, the services he can use are limited; if he provides more information, he may be able to use more services. Therefore, we need to balance these two functions. In this paper, we propose a new architecture based on two agent systems, named KODAMA[6] and VPC[3]. In this architecture, a service is managed by its *agent*, which has two resource management spaces named the *Public Zone* and the *Private Zone*.

The Public Zone is a space for realizing flexible public service use. The Private Zone is a space for protecting private user information. Moreover, agents are organized in a group called the *community* and are managed independently in each community. Thus, we realize the flexible service use and the protection of private information.

2 The Public Zone and the Private Zone

In this section, we introduce the Public Zone and the Private Zone. In our architecture, agents have a Public Zone and a Private Zone. The Public Zone is for flexible public service use. The Private Zone is for the protection of private resources. An overview of our architecture is shown in Fig. 1.

The Public Zone manages *public resource*s. A public resource is a resource, like a service or information. Public resources are open to the public. A public resource has a *public policy* which consists of a method for its service use and a specification of attributes. A user agent acquires a public policy from the service provider agent and uses its service by behaving according to its method.

A *Security Barrier* exists between the Public Zone and the Private Zone. The Security Barrier has functions for restricting access to private resources and for restricting communications of a program which accesses private resources.

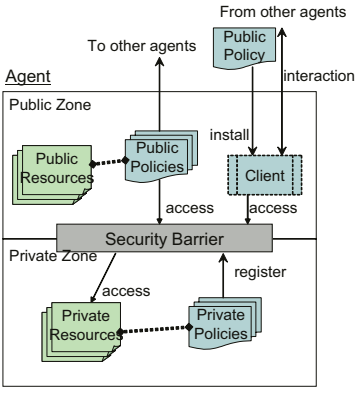

Fig. 1. An Overview of the Public and Private Zone Model

The Private Zone manages *private resources*. An agent cannot directly access private resources, but must access them through the Public Zone. A private resource has a *private policy* which consists of a method for accessing its resources and attributes governing access permission. Private policies are registered with the Security Barrier. The Security Barrier monitors access to private resources and communications of a program which has accessed private resources.

2.1 Public Policies and Private Policies

Each resource has a public policy or a private policy both of which consist of a method and attributes. A method is implemented by a program called the *client program*. The details of the client program are introduced at Sect. 2.3. Attributes consist of common attributes and characteristic attributes.

The common attributes are composed of *owner*, *type*, *description* and *agent_attr*. The owner attribute is the name of the agent which generated the policy. The type attribute shows whether the policy is a public policy or a private policy. The description attribute gives an explanation of the resource. The agent_attr is the list of attributes of the agent which managed the policy.

The characteristic attributes of a public policy are *dependency* and *communication*. The dependency attribute shows what resources are needed for using the resource. The dependency attribute is defined as the list of description attributes of the public policy. When an agent wants to use its resource, the agent must gather resources specified in the dependency attributes in advance. The communication attribute shows whom it is necessary to communicate with for using the resource. The value of the communication attribute can be any one of *no_communication*, *owner_only* or *agent_list*. No_communication means that no communication is required. Owner_only means that it requires only communications with the resource provider. Agent_list is defined as the list of agent name and means that it requires only communications with agents specified in its list.

The characteristic attributes of a private policy are *access* and *allowable_communication*. The access attribute specifies attributes of programs which are permitted the access the private resource. The allowable_communication attribute specifies communications allowed to the client program which accesses the private resource. The value of the communication attribute can be any one of *allow_all*, *agent_list*, *owner_only* and *deny_communication*. Allow_all permits only communications with agents specified in the communication attribute of the client program. Agent_list permits only communications with agent specified in its list. Owner_only permits only communications with the distribution origin (represented by the owner attribute) of the client program. Deny_communication denies all communication.

2.2 The Security Barrier

The Security Barrier is prepared for the protection of private resources between the Public Zone and the Private Zone in each agent. All the access must be done through the Security Barrier. The Security Barrier forbids the access from other agents and also checks the access from programs in the Public Zone. In this architecture, each private resource has a private policy. The private policy is registered with the Security Barrier. The Security Barrier protects private resources by restricting the access to private resources and restricting communications of the client program.

When the client program accesses to a private resource, the access is checked by the Security Barrier. The Security Barrier compares the common attribute (owner, description and agent_attr) of the client program and the access attributes of the private resource. If the access is accepted, the Security Barrier returns its value and registers the client program with the access-list; if it is rejected, IllegalAccessException happens. After that, the Security Barrier monitors the communication of the client program in the access-list. When the client program communicates with other agent, the Security Barrier compares the communication partner and the allowable_communication attributes of the private policy. Then, if its communication is allowed, the client program can communicate; if it is rejected, IllegalAccessException happens. In this way, the Security Barrier protects private resources by restricting the access and the communication.

2.3 The Client and the Service Program

In this architecture, each service is provided by one agent. A user agent uses a service by communicating with the service provider agent according to the method for use of the service. But methods for service use are different for each service. Therefore, it is difficult to implement an agent which is able to ab initio use various services. Therefore, we define the *service program* and the *client program* as a pair.

A service provider agent sets up a service program and a client program (as a part of the public policy) in its Public Zone. A user agent acquires a client program from a service provider agent and invokes it in its Public Zone. Then,

Table 1. Additional Methods for the Service and the Client Program

Result accessToPublicResource(*service_desc*) Call a client/service program specified in *service_desc* on the Public Zone and return its result
Result accessToPrivateResource(*service_desc*) throws IllegalAccessException Call a client/service program specified in *service_desc* on the Private Zone and return its result
void inform(*agt_name*, *method*, *par*, *res_method*) throws IllegalAccessException Send an invocation of the *method(par)* to *agt_name*. When the agent receives a response, it invokes the *res_method(response)*
Result accessTo(*agt_name*, *method*, *par*) throws IllegalAccessException Send an invocation of the *method(par)* to *agt_name* and suspend the program. When the agent receives a response, the program works again
void reply(Result *res*) Send *res* back

the service is actualized by communications, guided by the client program and the service program, between the service provider and user. The service and client program are implemented in Java with additional methods as shown in Table 1.

3 Decentralized Service Management

In an environment with a lot of services, it is difficult to manage and dispatch resources to users in a centralized way. Therefore, a decentralized service management mechanism is required. For that, we define the *community*. Each community manages agents in its community independently. Each community has a *portal* agent which is the representative of each community. A portal agent has *tax policies* that are obligations imposed on agents in its community. Agents are able to use services provided in the community to the extent that they fulfill their obligation.

3.1 The Tax Policy

The tax policy is the obligation imposed on agents who have joined a community. The tax policy is designed for checking the qualification to join the community and/or for obligating to provide the service. The tax policy consists of a client program and attributes. The attributes are the same as for the public policy. When agents join a community, they must install the client program specified in the tax policy in their Public Zone.

When an agent joins a community, it receives tax policies from the portal agent of the community. The agent installs the client programs in its own Public Zone and notifies that to the portal agent. If the community needs to check

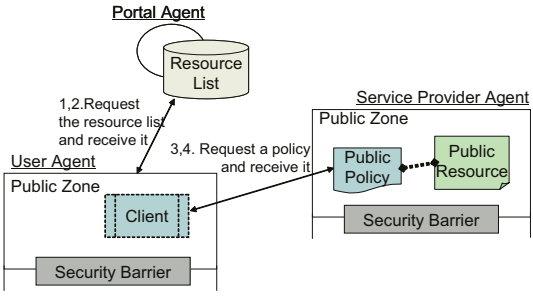

Fig. 2. The Steps for Obtaining a Client Program

the qualification to join the community, the portal agent accesses to the client program in the agent's Public Zone. Then, the client program accesses resources of the agent for checking the qualification to join the community[3] and returns its result. As the result, the portal agent sends a participation permission/refusal notification message. If the agent receives a participation permission notification message, the agent joins the community. Moreover, agents in the community provide the services specified in the tax policies to other agents in the community.

3.2 The Registration of the Service

The portal agent manages resources provided by agents in the community. When an agent joins the community, it sends owner and description attributes to the portal agent. The portal agent registers them in the resource table, allowing an agent to find resources by querying to the portal agent.

3.3 Service Use

An agent acquires a client program from a service provider agent and uses a service. The steps for obtaining a client program are shown in Fig. 2.

1. A user agent sends a resource list request message to a portal agent.
2. The portal agent returns the resource list (which consists of owner and description attributes).
3. The user agent finds necessary resources from the resource list.
4. The user agent requests a public policy from the service provider agent (indicated by the owner attribute in the resource list).
5. The service provider agent returns the requested public policy.
6. The user agent installs the client program detailed in the received public policy.

[3] A method for checking the qualification is not shown in this paper, because it depends on applications.

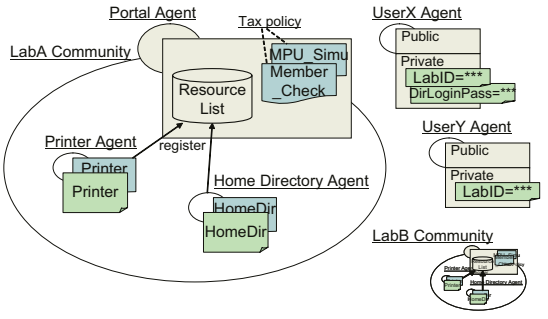

Fig. 3. The Application Overview

In this way, an agent acquires a client program. Subsequently, the user agent acquires public policies indicated in the dependency attribute of the received public policy. Finally, the user agent invokes the client program.

4 An Application Example

In this section, we show an application example in which a member of a laboratory makes use of services provided in the laboratory. In the example, we assume that sensors for detecting user's location are installed in the laboratory and notify the user agent of the community name corresponding to each user's location. We also assume that the problems of tapping, spoofing and masquerading have been solved by importing technologies of encryption, digital signature, Public Key Infrastructure and so on.

4.1 The System Structure

The application overview is shown in Fig. 3.

In this application, there is a printer agent and a home directory agent in the LabA community and the LabB community. A printer agent provides a printer services. A home directory agent provides home directory access services for users who have a login password. Also, LabA is simulating MPU (MicroProcessor Unit) processes that requires more CPU power. Accordingly, the portal agent of the LabA community has a tax policy (*MPU_Sim*) which supplies the CPU resources for the simulation, and a tax policy (*Member_Check*) which confirms whether agents are members of its laboratory or not.

UserX/userY are members of LabA/B, repectively. They have a *LabID* with the following attributes:

```
owner="UserX/Y", type="private", agent_attr=attribute of userX/Y,
description="Belonging to laboratory",
```

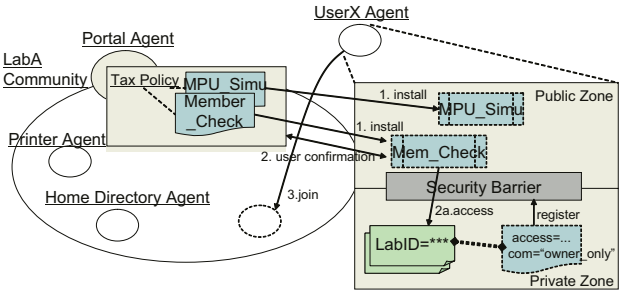

Fig. 4. The Behavior when the UserX agent Enters the LabA community

```
access=agent_attr:{owner="portal of LabA/LabB"},
allowable_communication="owner_only"
```

. Also, userX has a *HomeDirLoginPass* with the following attributes:

```
owner="UserX", type="private", agent_attr=attributed of userX,
description="Home directory login password",
access=agent_attr:{owner="Home Directory Agent"},
allowable_communication="owner_only".
```

4.2 Behavior When Entering the Laboratory

The behavior when userX enters LabA is shown in Fig. 4. When userX visits LabA, its community name (LabA) is notified to his agent by sensors. The agent then receives tax policies (MPU_Simu, Member_Check) from the portal agent of the LabA community and installs their client programs in its own Public Zone; the portal agent accesses Member_Check installed in userX's Public Zone and tries to confirm whether he is a member of LabA. The Member_Check program tries to access the LabID. Then, the access is allowed because attributes of LabID are `access=agent_attr:{owner="portal of LabA"}`, `allowable_communication="owner_only"`. As the result, the confirmation succeeds. After that, the userX agent receives a participation permission notice message and joins the LabA community. On the other hand, even if userY tries to join the LabA community, he can not join because attributes of his LabID are `Access=agent_attr:{owner="portal of LabB"}`.

And agents in the LabA community have the MPU_Simu program in their Public Zone, because MPU_Simu is specified in the tax policy of the LabA community. Therefore, agents in the LabA community supply the CPU resources for the MPU simulation. In this way, we can simulate MPU processes using CPU resources of other agents through their MPU_Simu program.

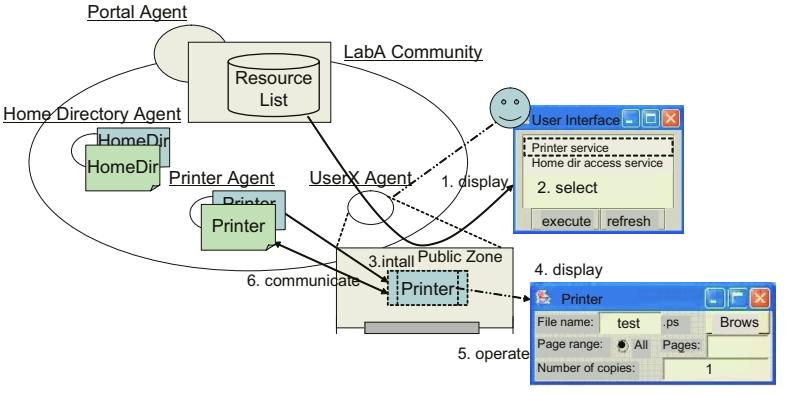

Fig. 5. The Behavior when a User Agent Uses a Service

4.3 Service Use in the LabA Community

The behavior involved in using a service in the LabA community is shown in Fig. 5. A user agent receives the resource list. The user agent shows its description attributes to the user. If the user selects a service he wishes to use, the user agent acquires the public policy and invokes the appropriate client program.

For example, when userX enters LabA, "Printer service" and "Home dir access service" are shown on his portable terminal. If he selects "Printer service", his agent acquires the Printer policy and invokes its client program. This results in the interface for "Printer service" being shown on his portable terminal, and he uses the printer through this interface. Also, if he wishes to use "Home dir access service", his agent acquires the HomeDir policy and invokes its client program. Then, the HomeDir client program tries to access the HomeDirLoginPass. Here, attributes of the HomeDirLoginPass are `Access=agent_attr:{owner="Home Directory Agent"}`, `Allowable_Communication="owner_only"`. Therefore, the HomeDir client program accesses the HomeDirLoginPass and tries to authenticate. If its authentication succeeds, userX can access his home directory; if it fails, he cannot. Of course, only programs generated by the Home Directory agent can access the HomeDirLoginPass .

4.4 The Evaluation of Our System

Service-Use Mechanism. We have defined pairs of service programs which are executed by a service provider agent and client programs which are executed by a user agent. A service provider agent provides service programs to user agents. Therefore, by getting client programs from service provider agents, user agents make use of various services. In the application example, if a user agent does not know a method for the service use in advance, he can use services by getting appropriate client programs from their service provider agents.

We also defined the tax policy, which is the obligation imposed on members of the community. Agents in the community must provide services specified in

tax policies. In the application example, the portal agent confirms whether an agent is a member of the laboratory or not by the Member_Check program, and the agents supplies the CPU resources for MPU simulation by the MPU_Simu program.

Protection of Private Resources. We defined two resource management spaces, the Public Zone and the Private Zone. The Public Zone is a space for flexible public service use. The Private Zone is a space for protecting private resources. All the access to resources in the Private Zone is examined by the Security Barrier according to each private policy. In the application example, we showed that only programs permitted by the private policy can access the LabID and the HomeDirLoginPass.

Decentralized Service Management. We introduce the community, which manages agents independently. In the application example, we defined the LabA and LabB community, each of which manages agents independently.

5 Related Work

UDDI (Universal Description, Discovery and Integration) , WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol) are three key specifications for implementing dynamic web service integration. UDDI offers users a unified and systematic way to find service providers through a centralized registry of services. WSDL is designed to describe the capabilities of any Web Service. SOAP is an XML-based protocol for messaging and Remote Procedure Calls (RPC). Because WSDL description is an interface for RPC or messaging, users must program to use its service. In our architecture, by getting a method for each service, user can use services without programming.

Many researchers are trying to develop security systems for Digital Rights Management and Trusted Computing[2, 4]. However, most of them are principally based on the client-server model or domain-specific distributed environments. Therefore, it is difficult to cover widely distributed environments in which there are a lot of services.

6 Summary

In this paper, we introduced a new architecture which has two resource management spaces: one is the Public Zone for flexible public service uses based on the acquisition of client programs; the other is the Private Zone where private resources are protected under the supervision of the Security Barrier. Therefore, a user agent can use various services while protecting private resources. Future work will clarify the details of the public/private policy attributes and the details of the client/service programs through practical application of the architecture.

References

1. distributed.net. http://distributed.net/.
2. J. S. Erickson. Fair Use, DRM, and Trusted Computing. *Communication of ACM*, Vol. 46(4), pp. 34–39, 2003.
3. T. Iwao, Y. Wada, M. Okada, and M. Amamiya. A Framework for the Exchange and Installation of Protocols in a Multi-Agent System. *CIA2001*, pp. 211–222, September 2001.
4. L. Kagal, T. Finin, and A. Joshi. Trust-Based Security for Pervasive Computing Environments. *IEEE Computer*, Vol. 34(12), pp. 154–157, 2001.
5. SETI@home. http://setiathome.ssl.berkeley.edu/.
6. Guoqiang Zhong, et al. The Design and Implementation of KODAMA System. *IEICE Transactions INF.& SYST.*, Vol. E85-D, No. 4, pp. 637–646, April 2002.