Early Experience of Remote & Hot Service Deployment with Trustworthiness in CROWN Grid[1]

Hailong Sun+, Yanmin Zhu*, Chunming Hu+
Jinpeng Huai+, Yunhao Liu*, Jianxin Li+

+ School of Computer Science, Beihang University, Beijing, China
{sunhl, hucm,huaijp,lijx }@act.buaa.edu.cn

* Department of Computer Science, Hong Kong University of Science & Technology, Hong Kong
{ zhuym, liu}@cs.ust.hk

Abstract. CROWN Grid aims to empower in-depth integration of resources and cooperation of researchers nationwide and worldwide. In such a distributed environment, to facilitate adoption of services, remote and hot service deployment is highly desirable. Furthermore, when the deployer and the target container are from different domains, great security challenges arise when a service is deployed to the remote container. In this paper, we present ROST, an original scheme of Remote & hOt Service deployment with Trustworthiness. By dynamically updating runtime environment configurations, ROST avoids restarting the runtime system during deployment. Moreover, we adopt trust negotiation in ROST to assure the security of service deployment. We conduct experiments in a real grid environment, and evaluate ROST comprehensively.

Keywords: Service grid, CROWN, Remote and hot deployment, ROST, Trust Negotiation Agent (TNA)

1 Introduction

Grid computing promises to enable coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [1]. In recent years, service-oriented grid architecture is introduced, which is widely considered as the future of grid computing [2]. Built on web services, OGSA [3] is the de facto standard for building service grids, in which various resources are encapsulated as services with uniform user interfaces.

The main goal of our key project, CROWN (**C**hina **R**&D Environment **O**ver **W**ide-area **N**etwork) Grid, is to empower in-depth integration of resources and cooperation of researchers nationwide and worldwide. CROWN project was started in late 2003. A number of universities and institutes, such as Tsinghua University, Peking University,

[1] This work is partially supported by the National Natural Science Foundation of China under Grant 91412011, China Ministry of Education under grant CG2003-CG004 & GP004 & GA004 and Microsoft Research Aisa.

Chinese Academy of Sciences, and Beihang University, have joined CROWN, with each contributing several computing nodes. More universities and institutes will be invited to join CROWN Grid by mid 2005.

In the past years, many key issues in grid computing have been extensively studied. However, remote and hot service deployment has not been fully addressed. Before a service is ready for invocation, it must be deployed in a service container which provides a runtime environment. A grid is a highly distributed environment, in which numerous domains could be involved. The domains are usually geographically dispersed. It is highly desirable for a user to deploy its services into remote service containers for multiple purposes. For example, in CROWN Grid for bioinformatics application, there are many computing intensive applications such as BLAST. A computing node could easily be over-loaded when multiple jobs arrive in a short period. The heavy load can be balanced if the node is able to deploy one or more BLAST service replica to remote nodes and then redirect some jobs. Similar requirements also exist in many other grid applications.

Traditionally, remote service deployment is supported in a cold fashion, which means, to deploy a new service, the runtime environment need to be restarted. This results in many disadvantages because previously running services must be stopped, and they may have to resume or even restart their jobs, causing significant overhead. Therefore, hot service deployment has become increasingly important, which does not need to restart the runtime environment while deploying services. With the availability of remote and hot service deployment, many applications will benefit, such as load balancing, job migration and so on.

Service deployment is actually not a new issue. Similar demands also exist in mobile agents [4] and active networks [5]. To the best of our knowledge, however, there is no successful solution to enabling remote and hot service deployment in grid systems. The most updated Globus Toolkit version 4 [6], the de facto standard for grid middleware, does not provide the function of remote and hot service deployment yet. This may be due to the great security challenges arising when a user deploys a service to a remote container. Here we call a node *deployer*, which intends to deploy a service, and the remote service runtime environment *target container*, which is responsible for running and managing services being deployed. Without proper security mechanisms, a service provided by a deployer may be malicious, and the target container may be rogue or fragile. Also, the security policies of the deployer and the container could be incompatible. In an open grid environment, we can not expect any deployer and the corresponding target container to set up required trust relationship in advance. Moreover, it is too costly to build the trust across domains based on the traditional PKI infrastructure every time during remote deployment.

In this paper, we present our original work, ROST (**R**emote and h**O**t **S**ervice deployment with **T**rustworthiness), which achieves its goal by dynamically updating the runtime environment configurations. ROST avoids restarting runtime systems during remote deployment. Moreover, we include trust negotiation in ROST scheme, which greatly increases flexibility and security of CROWN. Major contributions of this work are as follows:

- We identify the necessity of remote and hot service deployment in service grids, and their challenges.
- We propose an effective approach, ROST, to enable remote and hot service deployment. Also, we add trust negotiation into the scheme to meet general security requirements for grid environments.
- We implement ROST in CROWN Grid and evaluate the performance of ROST by comprehensive experiments.

The rest of this paper is organized as follows. We discuss related work in Section 2. In Section 3, we introduce the design and implementation experiences. We present experimental methodology and performance evaluation of ROST in section 4. And in section 5, we conclude this work.

2 Related Work

Globus Toolkit is the most famous grid middleware and it has begun to support service-oriented grid computing based on OGSA since version 3. But even in the updated release version 4, remote and hot service deployment is not supported. Grid service is actually built on Web service, and extended to include functions such as state and life cycle management. For Web services, several middleware, such as Apache Axis [7], JBOSS [8] and Microsoft .NET [9], have partly implemented dynamic service deployment, i.e., deploying a local service without restarting service containers. However, Web service is much simpler than grid service, e.g. web services are normally stateless, so web service middleware can not apply to grid environments. Also, most of them only consider local deployment.

Friese et al. [10] proposed a method for hot service deployment in an ad hoc grid environment based on OGSI which is now replaced by WSRF. To ensure security, they make use of sandbox which can restrict the service function. DistAnt [11] extends the Apache Ant build file environment to provide a flexible procedural deployment description, and provides a solution to remote and hot service deployment based on Globus Toolkit 3. It does not provide any security mechanism for remote deployment. Baude et al. [12] proposed a solution for deployment and monitoring of applications written using ProActive, which is a Java-based library for concurrent, distributed and mobile computing. It does not consider grid service deployment issues.

3 ROST Design and Implementation

CROWN consists of numerous organizations with each of them forming a domain, as illustrated in Figure 1. Domains are usually connected by the Internet. CROWN, as a service-oriented grid, encapsulates various resources as services. In CROWN, a computer must be installed a Node Server (NS), a CROWN middleware. An NS contains a service container which provides runtime environment for various services. Each NS usually belongs to a security domain. Every domain has at least one RLDS (Resource

Locating and Description Service) to provide information service. RLDS maintains dynamic information of available services.

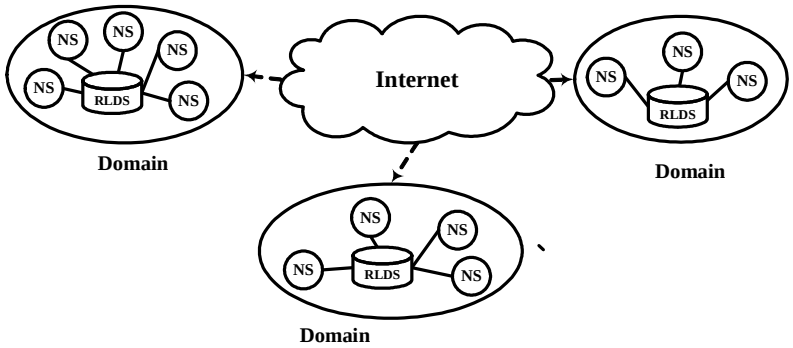

Fig. 1. Resource organization in CROWN

Remote service deployment is needed when a deployer needs to deploy a service on an NS in a different domain. In this paper, we refer to deploying a service to an NS and deploying a service to a container interchangeably, which means the same. A service is basically an entity that consists of an executable program, a description file, and several configuration files. Before a deployer's services can be ready for invocation in the remote NS, two key issues must be addressed. The first is security, namely, how to guarantee the service provided by the deployer is not malicious and the environment provided by the remote container is safe to the service. The second is how to enable the service to be available without restarting the remote container.

In CROWN, services follow the WSRF specifications [13]. A complete service consists of several files, as shown in the following.

- Executable programs. Such as Java classes, scripts, EJBs, etc.
- One or multiple WSDL files. Description of interfaces and access protocols of a service.
- A WSDD file. Web Service Description Descriptor, description of service configuration for the service container.
- BPEL files. Description of composed services which are described in BPEL4WS (Business Process Execution Language for Web Services).
- A JNDI configuration file. Description of WSRF resources of a service.
- A security configuration file. Description of authorization approach and other security related information.

To facilitate the transportation and protection of services, we compress a service into one single file. By far, we have adopted Globus Toolkit's GAR file format. In addition, we have extended GAR so that it is able to contain multiple types of executable programs and description files.

3.1 ROST Architecture

As shown in Figure 2, ROST is composed of several components while our discussions will focus on the two major ones, i.e., TNA and RHD.

TNA is responsible for trust establishment between a pair of deployer and container, and RHD is for remote and hot service deployment. The SCC (Service Container Configuration) is the abstract of various configurations of service containers. Indeed, each deployment operation results in an update to SCC.

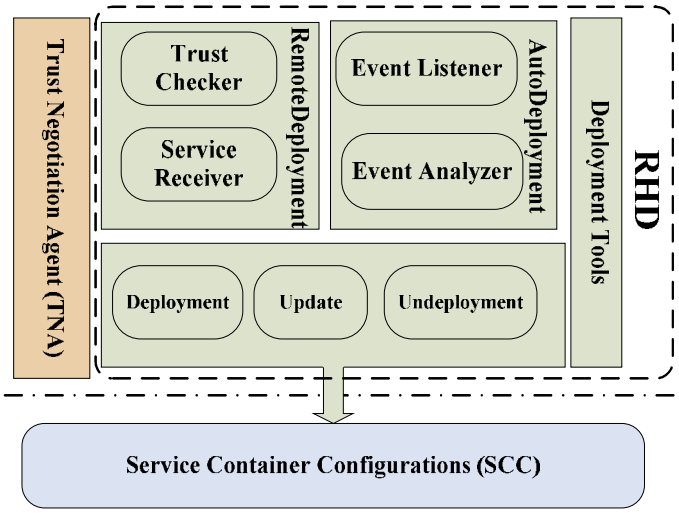

Fig. 2. ROST components

The procedure of service deployment can be divided into two phases: trust negotiation by TNA and deployment by RHD. To be more specific, the workflow of ROST is depicted as follows:

Step 1: the deployer sends a deployment request to a remote NS;

Step 2: the remote NS checks locally whether it can afford the new service; if yes, goes to Step 3;

Step 3: the remote NS checks whether the deployer has been trusted according to the local domain controller or the history information. If yes, sends a trusted notification; otherwise, initiates trust negotiation;

Sep 4: the deployer checks whether the remote NS has been trusted. If yes, sends a trusted notification, and goes to Step 5; otherwise, initiates trust negotiation;

Step 5: if the negotiation successfully sets up the desired trust, the deployer initiates service deployment by transferring the service to the remote NS;

Step 6: the remote NS performs hot deployment of the service.

Step 7: the remote NS acknowledge the success of the deployment.

3.2 TNA: Trust Negotiation Agent

3.2.1 ATN Technology

Several security infrastructures have been proposed for grid computing. For instance, in Grid Security Infrastructure (GSI) [14], every user or computer is uniquely identified by a X.509 certificate, which is issued by a Certificate Authority (CA). This fashion provides very limited capability of security control and it is rarely possible to deploy such a global hierarchy of CAs in an open environment like CROWN.

ATN (Automated Trust Negotiation) [15-19] is a new approach to access control in an open environment, which, in particular, successfully protects sensitive information while negotiating a trust relationship. With ATN, any individual can be fully autonomous. Two individuals try to set up a trust relationship by exchanging credentials according to respective policies.

Based on the above observations, we solve the trustworthiness problem in ROST by adding a Trust Negotiation Agent (TNA) , which is generally based on ATN technologies.

3.2.2 Trust Negotiation in ROST

As illustrated by Figure 3, TNA has mainly four components as follows.

- **TrustTicket Manage**r: The Access Mediator is responsible for issuing new TrustTickets for requesters and validating TrustTickets based on local Ticket Repository.
- **Strategy Engine**: The negotiation strategy [20] is used to determine when and how to disclose local credentials and policies. Also, it makes decisions to update the negotiation states, including success, failure or continuance.
- **Compliance Checker**: This component determines which local credentials satisfy the requester's policies or whether the requester's credentials satisfy local policies.
- **Credential Chain Discovery**: For trust negotiation in open networks, access control decision often involves finding a credential chain that delegate authority from the source to the requester, when the credentials are not stored locally. The main function of this component is to discover and collect necessary credentials.

In ROST, TNA is deployed on both sides of deployers and target containers. If a requestor has a valid *TrustTicket*, then the access mediator will call *TrustTicket* Manager to make access decision. Otherwise, trust negotiation will be triggered. When the requestor discloses its policies, the Strategy Engine decides whether the negotiation should continue. If so, the Access Mediator will call Compliance Checker to make corresponding verification to ensure which credentials should be provided, then responds with the necessary credentials and policies. In some cases, if the credentials are not available in local Credential/Policy Repository, Credential Chain Discovery is called to dynamically retrieve necessary credentials. Similarly, when the requester submits its credentials, the Access Mediator will call Compliance Checker to make corresponding verification to ensure whether the credentials satisfy local policies and make access decisions.

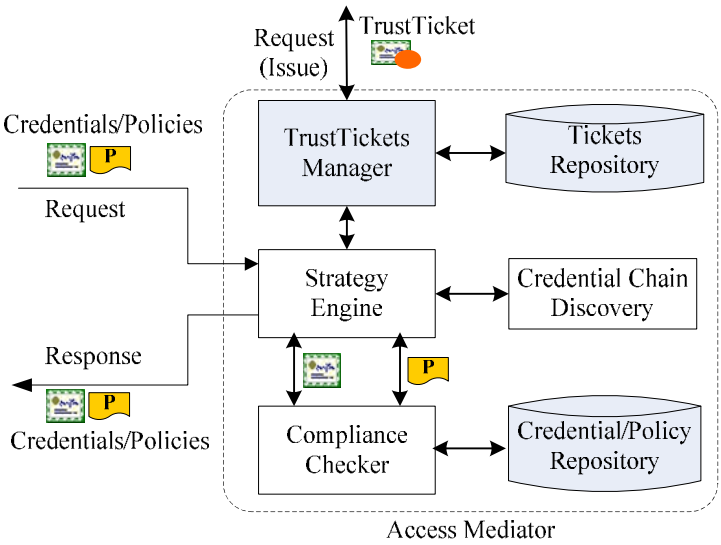

Fig. 3. TNA structure

In TNA, we adopt refined RTML (Role-based Trust Management Language Markup Language) to represent both access control policy and attribute-based credentials. When credential storage is distributed, the goal-directed algorithm [16] ensures that all credentials available can be discovered and collected. In ROST design, the *TrustTicket* takes the form of <*subject, issuer, subject, valid date, expiration data, signature*>. It is an identity assertion represented with XML with short lifetime assigned by the issuer.

In addition, negotiation information exchange between participants must rely on a secure communication protocol such as SSL/TLS to prevent eavesdropping, man-in-the-middle attacks, replay attacks, etc. Our ROST implementation conforms to WS-Security and WS-Conversation specifications for SOAP message protection.

3.3 RHD

After a negotiation successfully sets up desired trust, the container receives the service from the deployer and begins to deploy it.

RHD enables remote hot deployment as well as providing a convenient way for local hot deployment. RemoteDeployment and AutoDeployment, as shown in Figure 2, are respectively responsible for remote and local service deployment.

3.3.1 RHD APIs

We design APIs for both remote and local deployment, through which users are able to develop high level middleware and applications. There are basically three types of deployment operations: deploy, update, and undeploy. We define nine APIs to support these deployment operations as follows.

(1) *deploy* (String *garFilePath*)
(2) *deployByFTP* (URL *garFileURL*, String *user*, String *password*)
(3) *deployBySOAPAttach*(String *garFilePath*)
(4) *update* (String *garFilePath*)
(5) *updateByFTP* (URL *garFileURL*, String *user*, String *password*)
(6) *updateBySOAPAttach*(String *garFilePath*)
(7) *undeploy*(String *garFileName*)
(8) *undeploy*(String *serviceName*)
(9) *getAllDeployedServices*()

Note that (1)-(3) are three interfaces for deploying a service, while (1) is for deploying a service locally; and (2)(3) provide two different interfaces for remote deployment. The (4)-(6) defines three interfaces for updating deprecated services. The (7) and (8) defines two interfaces for removing services from service containers. We define (9) for querying all services deployed in a service container.

3.3.2 Remote Deployment

After mutual trust is successfully established, *ServiceReceiver* is called to receive the GAR file and uncompress it by *GARUnzipper*. Then the underlying deployment functions are called to perform corresponding operations.

A service container must include various configurations of the deployed services. Indeed, the key to hot deployment is to update the configuration of SCC dynamically. Relevant configurations include executable programs, WSDL description, WSDD, and JNDI configuration. For example, when a new service implemented with JAVA needs to be deployed, we have to let SCC load JAVA classes of the service.

For updating or un-deploying an existing service, it should be careful since other services or users might be using it. Simply updating or undeploying a service without adding special measures may lead to unexpected service interruption to users. To solve this problem, we add a reference counter for each deployed service. The initial value of a counter is zero, and the value increases/decreases by one each time when the service is invoked/completed. When an update or undeployment request comes, we first check the counter of the service. A service is ready to be updated or undeployed only if the reference counter is equal to zero.

3.3.3 Auto Deployment

Besides remote and hot deployment, RHD component also provide a convenient method to hot-deploy services to local containers.

A file folder is specified to receive GAR files and an *EventListener* keeps listening to the events associated to the folder. The *EventListener* is interested in two types of events: arrival of new files and deletion of existing files.

Suppose an event *e* caught by *EventListener* is passed to the *EventAnalyer* for analysis and further process. Based on contents of an event, the *EventAnalyzer* will call underlying different deployment functions. In the following, we provide the pseudo code of this process.

```
if (e is arrival of a new file){
  if (file type is GAR){
        if ( the file already exists){
             while(reference number > 0){
            sleep(2000 milliseconds);
      }
         update the corresponding service;
         }else{
        deploy the corresponding service;
         }
 }else{
       remove the file;
 }
}else if ( e is file deletion){
     while( reference number > 0){
           sleep(2000 milliseconds);
   }
undeploy the corresponding service;
}
```

As a result, users may deploy/update/undeploy a local service by simply storing/replacing/removing its GAR file to/in/from a folder. They need not to care about underlying processes, and services are deployed/undeployed automatically and transparently.

4 Performance Evaluation

ROST is implemented as a core component of CROWN middleware. We evaluate the performance of ROST by comprehensive experiments in real grid environments.

4.1 Experimental Environment

The experiments are conducted across two domains connected by the Internet. The deployer resides in Tsinghua University, while the target NS's (i.e., target containers) are located in Beihang University. The deployer has a Pentium III 1.6Ghz CPU and 512M memory, with a 10M bps connection to the Internet. Remote NS's reside in a 32-node cluster with each has two Intel Xeon 2.8GHz CPUs and 2G memory. The cluster is connected to the Internet through a 100M bps connection. No other tasks are running on each node except the necessary CROWN middleware.

4.2 Performance Metrics

We use the following metrics to evaluate ROST.

- *Deployment response time*. It is important that a remote service deployment introduces shorter response time. When multiple concurrent deployment requests are sent to a single NS, the deployment response time increases.
- *Task execution time*. A task here means a collection of independent jobs, while a job means an invocation of a specific service. Given a task, we concern its total execution time.

4.3 Experimental Results and Analysis

We execute the experiment 100 times and report the average.

In the first experiment, we evaluate the performance of ROST in terms of deployment response time. The deployer in Tsinghua University issues concurrent deployment requests to a node server in Beihang University. We vary the ways of service GAR file transfer, FTP and SOAP attachment. Each GAR file has a size of about 6K bytes.

Figure 4 shows the average deployment response time as a function of the number of concurrent requests. When there is only one request each time, the response time of ROST is as short as seven seconds. In contrast, the cold deployment needs as long as 30 seconds to merely stop and restart the service container so as to load a new service. With increasing number of concurrent requests, the average response time increases roughly linearly. When the number of concurrent requests reaches 30, the average response time is about 52 seconds. We also observe that SOAP transfer has similar performance with FTP mechanism.

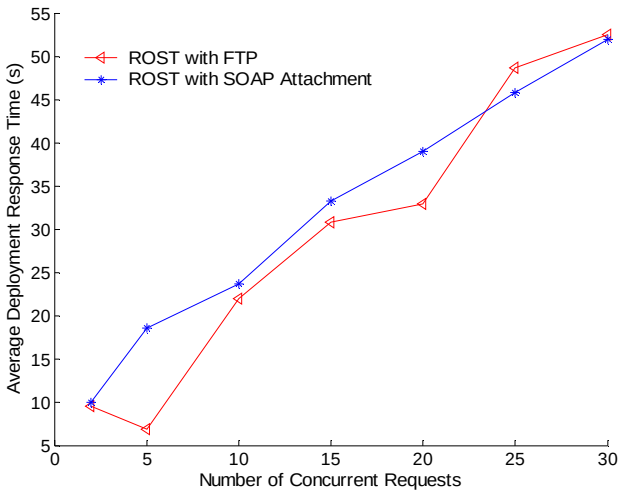

Fig. 4. Average deployment response time v.s. Number of concurrent requests

We then study how well ROST can help to achieve load balancing. In the second experiment, two schemes are compared, *with* and *without* ROST. There are 20 NS's

available for processing jobs, while initially only a fraction of the nodes are deployed with the required service.

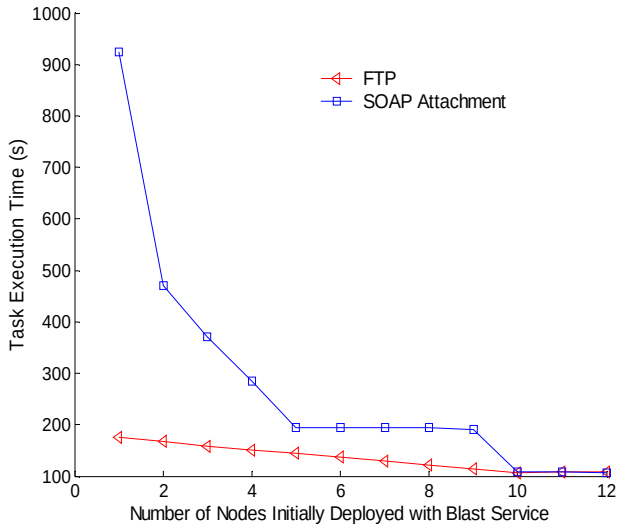

Fig. 5. Task execution time vs. number of nodes initially deployed with Blast service

Figure 5 plots the task execution time with different number of nodes initially deployed with the service. With ROST, the task execution time is significantly reduced, as a node may easily deploy its service to other relatively idle nodes. In some specific cases, the maximum improvement can be four times faster. When the fraction of nodes initially deployed with the service increases, the effect of time reduction becomes less.

5 Conclusions and Future Work

CROWN Grid aims to integrate nationwide and worldwide valuable Internet resources. In CROWN, remote and hot service deployment is highly demanded. In this paper, we present early design and implementation experience of remote & hot service deployment with trustworthiness (ROST). With ROST, services can be deployed to a remote container in a different security domain in a hot and secure fashion, which significantly improves service efficiency and quality. The experiments in real grid environment demonstrate the effectiveness of ROST.

In future work, we will perform more experiments, explore more relevant trust mechanisms, and further improve trust negotiation and deployment efficiency. Additionally, we will further integrate ROST with other CROWN middleware to handle real application problems such as load balancing and job migration.

References

1. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organization," The International Journal of High Performance Computing Applications, vol. 15, pp. 200-222, 2001.
2. I. Foster and C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure. San Francisco: Morgan Kaufmann, 2003.
3. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid Services for Distributed System Integration," IEEE Computer, vol. 35, pp. 37-46, 2002.
4. L. Bernardo and P. Pinto, "Scalable Service Deployment using Mobile Agents," presented at the Second International Workshop on Mobile Agents, 1998.
5. M. Bossardt, A. Muhlemann, R. Zurcher, and B. Plattner, "Pattern Based Service Deployment for Active Networks," presented at the Second International Workshop on Active Network Technologies and Applications, 2003.
6. "The Globus Toolkit: http://www.globus.org/toolkit/."
7. "Apache Axis: http://ws.apache.org/axis/."
8. M. Fleury and F. Reverbel, "The JBoss Extensible Server," presented at ACM/IFIP/USENIX International Middleware Conference, 2003.
9. "Microsoft.NET: http://www.microsoft.com/net/."
10. T. Friese, M. Smith, and B. Freisleben, "Hot Service Deployment in an Ad Hoc Grid Environment," presented at International Conference on Service Oriented Computing, 2004.
11. W. Goscinski and D. Abramson, "Distributed Ant: A System to Support Application Deployment in the Grid," presented at the Fifth IEEE/ACM International Workshop on Grid Computing, 2004.
12. F. Baude, D. Caromel, F. Huet, L. Mestre, and J. Vayssiere, "Interactive and Descriptor-based Deployment of Object-Oriented Grid Applications," presented at the 11th IEEE International Symposium on High Performance Distributed Computing, 2002.
13. WSRF Specifications, http://www.oasis-open.org/committees/tc_home.php.
14. I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," presented at the 5th ACM Conference on Computer and Communications Security, 1998.
15. W. H. Winsborough, K. E. Seamons, and V. E. Jones, "Automated Trust Negotiation," presented at DARPA Information Survivability Conference and Exposition, 2000.
16. N. Li, W. H. Winsborough, and J. C. Mitchell, "Distributed Credential Chain Discovery in Trust Management," presented at the 8th ACM Conference on Computer and Communications Security, 2001.
17. W. H. Winsborough and N. Li, "Towards Practical Automated Trust Negotiation," presented at the 3rd International Worshop on Policies for Distributed Systems and Networks(POLICY 2002), 2002.
18. W. H. Winsborough and N. Li, "Safety in Automated Trust Negotiation," presented at IEEE Symposium on Security and Privacy, 2004.
19. M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu, "Negotiating Trust on the Web," IEEE Internet Computing, vol. 6, 2002.
20. T. Yu and M. Winslett, "A Unified Scheme for Resouce Protection in Automated Trust Negotiation," presented at IEEE Symposium on Security and Privacy, 2003.