Preliminary Experience of ARMS: A Web Service Based Automatic Resource Management System

Yong Yan, Bo Shen
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2002-295
October 15th , 2002*

multimedia services, resource management, web services, authentication, access control, Kerberos, grid computing, remote visualization

To build a rich-media service grid for automatically managing the life-cycle activities of services, it is essential to have an automatic resource management system to provide a cost-bounded capacity guarantee illusion to services over distributed heterogeneous resources. In this paper, we describe our preliminary experience in the design and implementation of the ARMS (Automatic Resource Management System), a Web service based resource management system. The emerging web service model with single SOAP-based RPC interface provides a good way for the ARMS to uniformly abstract underlying resources and hide the heterogeneity of resources and services.

In the ARMS, all heterogeneous workstations and management functions are abstracted as web services based on an abstraction mechanism, named service agent. The service agent provides a SOAP-based RPC messaging engine for uniforming the service interoperation and web-service publish mechanisms for easing the service deployment. To achieve automatic resource management, a secure, scalable, and self-managing distributed resource container, called DMS, is proposed in order to provide resource capacity guarantee for a rich-media service. An DMS can be customized by service-specific policies and cooperates with the global resource management service (GRMS) to achieve demand-driven management automation. The GRMS is responsible for managing global free resources.

In order to protect geographically distributed resources and services in the system, the ARMS has implemented an extensible security service that seamlessly integrates Kerberos-like authentication with a role-based access control model. To achieve enforcement transparency, two secure mechanisms: passport and service guard have been introduced to transparently integrate the authentication and access control functions into the web service model.

Preliminary Experience of ARMS: A Web Service Based Automatic Resource Management System

Yong Yan and Bo Shen

Client and Media Systems Lab in HP Labs, Palo Alto 94304

***Abstract*—To build a rich-media service grid for automatically managing the life-cycle activities of services, it is entailed to have an automatic resource management system to provide a cost-bounded capacity guarantee illusion to services over distributed heterogeneous resources. In this paper, we describe our preliminary experience in the design and implementation of the ARMS (Automatic Resource Management System), a Web service based resource management system. The emerging web service model with single SOAP-based RPC interface provides a good way for the ARMS to uniformly abstract underlying resources and hide the heterogeneity of resources and services.**

In the ARMS, all heterogeneous workstations and management functions are abstracted as web services based on an abstraction mechanism, named service agent. The service agent provides a SOAP-based RPC messaging engine for uniforming the service interoperation and web-service publish mechanisms for easing the service deployment. To achieve automatic resource management, a secure, scalable, and self-managing distributed resource container, called DMS, is proposed in order to provide resource capacity guarantee for a rich-media service. An DMS can be customized by service-specific policies and cooperates with the global resource management service (GRMS) to achieve demand-driven management automation. The GRMS is responsible for managing global free resources.

In order to protect geographically distributed resources and services in the system, the ARMS has implemented an extensible security service that seamlessly integrates Kerberos-like authentication with a role-based access control model. To achieve enforcement transparency, two secure mechanisms: passport and service guard have been introduced to transparently integrate the authentication and access control functions into the web service model.

***Index terms*—Multimedia services, resource management, web services, authentication, access control, Kerberos, Grid computing, remote visualization.**

1 INTRODUCTION

Internet provides a pervasive interconnect infrastructure for aggregating all kinds of heterogeneous distributed computing resources together to support a wide range of information sharing, collaboration, and distributed computing. This makes it possible to build global information infrastructure to realize virtual organizations for the global sharing and management of resources and services as envisioned by the Grid Computing [11], the EOS [28], and the AMESE [29].

Driven by the scientific computing and technical computing, the grid computing forum comes up with a set of core technologies as a software toolkit (GT2.0), for global data movement, LDAP based information sharing, global resource scheduling, and certification based authentication [14]. However, the GT2.0 is not flexible and general enough to address interoperability and sharing among heterogeneous resources and applications. Recently, the emerging of the Web-service standard [16][27] attracts the grid forum to rebuild the grid foundation on top of the Web-service model, which will enable the grid technology to support the general commercial computing [13]. The web service model provides a general and portable software architecture for universal application integration and messaging based on the XML-based SOAP RPC messaging mechanism and the WSDL-based extensible service representation. The generality of the web service architecture also provides a standard way to abstract heterogeneous resources and services. As a general platform for supporting all kinds of applications and services, the grid infrastructure still have lots of challenge problems to solve before its success.

In parallel with the research and development effort in the grid computing, the rich media service grid, denoted as rmsGrid, is an effort in HP labs with the aim to come up with a general automatic management infrastructure for rich media pervasive commercial services. The rich media content requires that the rmsGrid guarantees service level agreements in order to provide guaranteed end-to-end QoS to customers. As a service grid, the rmsGrid is further required to be secure, scalable and reliable. Guided by our previous research result [29], the rmsGrid design aims at (1) maximizing management automation to minimize the complexity and cost of the system, and (2) taking advantage of portable and extensible standard technologies, such as the Java, J2EE, and Web-service standard.

In the rmsGrid, a global intelligent resource management system is entailed to seamlessly harvest distributed heterogeneous resources to meet various kinds of dynamic resource demands of rich media services. In order to provide a uniform resource programming interface over heterogeneous resources, a global resource management system usually virtualizes heterogeneous resources to a common resource representation. This motivates us to design and develop a global resource management system, named ARMS (Automatic Resouce Management Systems) based on the web service standard. This paper mainly introduces the design and implementation progress of system mechanisms and functional components of the ARMS. The policies running to make the system smarter will be separately addressed in the future.

In the ARMS, all resources and management functions are uniformly designed and implemented as web services based on a general abstraction mechanism: service agent. The service agent provides a SOAP-based RPC messaging engine for uniforming the service interoperation and web-service publish mechanisms for automating the service deployment.

The ARMS currently consists of five types of services: management service, directory service, security service, workstation service, domain service and client service. The workstation service is the only platform-dependent service which performs platform-dependent monitoring and resource control while abstracting an underlying physical resource into a virtual resource.

In order to support self-managing rich-media services, the ARMS comes up with a self-managing resource management mechanism: domain management service (DMS). An DMS is a secure, scalable, and self-managing resource container, which is dynamically generated to provide resource capacity guarantee for a rich-media service. An DMS can be customized by service-specific policies. The DMS cooperates with the global resource management service (GRMS) to achieve demand-driven management automation. The GRMS is responsible for managing global free resources.

In order to protect geographically distributed resources and services in the system, the ARMS has an extensible security service that seamlessly integrates Kerberos-like authentication with a role-based access control model. The authentication system is modified from the Kerberos so that (a) the authentication system can be integrated into the resource management sysem, instead of working as an external component, for better management capability; (b) the client has single access point to the resource management system which allows stateless client interface. To achieve enforcement transparency, two secure mechanisms: passport and service guard have been introduced to transparently integrate the authentication and access control functions into the web service model. Passport and service guard are two distributed security plugins in the web service's messaging engine, cooperatively enforcing principal authentication, message integrity checking, and service access control for distributed services and resources.

This paper is organized as follows: we compare our work with related work in next section. Then, we present the high-level picture of the rmsGrid and describe the requirements on the ARMS in the rmsGrid in Section 3. In Section 4, we describe the system architecture of the ARMS: automatic resource management system. In Section 5, we introduce our abstraction mechanism of services and resources. The automatic resource management is described in Section 6. The security infrastructure of our system is introduced in 7. In Section 8, we introduce the implementation status of our system. Then, we conclude the paper in Section 9.

2 RELATED WORK

In the grid computing, different kinds of resource management systems have been proposed [17]. However, non of them has addressed how to automatically manage resources based on the resource demand of a service. Recently, the automatic resoruce management has been addressed by Muse [4], DDSD [31], and Océano [2] in the context of the utility data center. The ARMS aims at automating the resource management in an open grid infrastructure. The DMS is a distributed and secure service prioviding resource capacity guarantee for end-to-end services. However, the ARMS can indeed benefit from existing results in the Grid computing and those in the utility data center. In addition, The ARMS is among the first effort to explore the methodology of building portable, extensible, easily manageable resource management system based on the Web service model.

To protect the system, the Globus toolkit uses a certification based authentication system [4]. Unlike the Globus toolkit and other methods [8][9][10], the security infrastructure of the ARM adopts a Kerberos-like authentication which makes it possible to achieve both the stateless client interface and the easy manageability. With the support of two security mechanisms: passport and service guard, the integrated security infrastructure has achieved enforcement transparency and the separation of enforcement and policy management in the Web-service model.

3 RMSGRID AND THE DESIGN REQUIREMENTS FOR THE ARMS

The media service Grid (denoted as msGrid) is visioned as an automatic management infrastructure for multimedia services and resources with the aim to deliver reliable, QoS guarantteed, trustworthy, and scalable multimedia services to end users. The msGrid differentiates itself from other types of Grids by its strong committment on achieving QoS guaranttee. The msGrid automatically manages the end-to-end activities in the lifecycle of a media service, which is a concrete instance of the general automatic management framework explored by the AMSES [29].

Conceptually, the msGrid can be considered to consist of three layers: multimedia user interfaces on the top, self-managing multimedia services in the middle, and an automatic resource management system on the bottom, as shown in Figure 1.

On the user end, a multimedia virtual operating environment (MVOE) is provided to access all kinds of multimedia services, such as remote collaboration, video-conference, streaming media, gaming, remote-visualization, etc. The MVOE should be a mix of text, audio, video, and graphics. The same MVOE should be runnable on all kinds of devices from desktop, thin client, to PDA. The MVOE should be stateless to

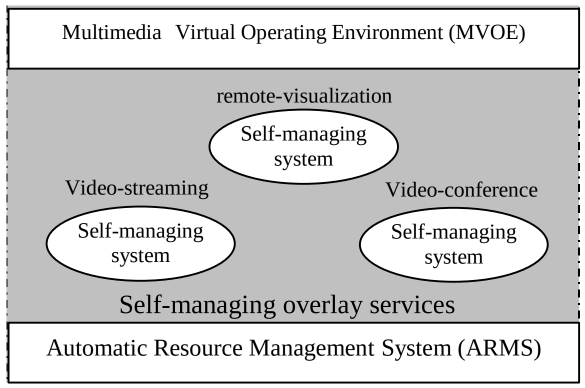
Multimedia Virtual Operating Environment (MVOE)
remote-visualization
Self-managing system
Video-streaming
Self-managing system
Video-conference
Self-managing system
Self-managing overlay services
Automatic Resource Management System (ARMS)

Figure 1. Conceptual picture of msGrid.

make services pervasive. Besides the usage interface, the MVOE also provides multimedia management interface for those self-managing services.

The middle layer of the msGrid consists of self-managing overlay services where an overlay service may be composed of other overlay services. The overlay service is self-managed based on service level agreements (SLAs) contracted between the user and the msGrid. The msGrid provides each service with common self-managing mechanisms that are customized based on service specific policies to deliver a multimedia service to the user. The self-managing is conducted through monitoring SLAs, detecting SLAs violations, and adjusting resource demands. A self-managing service derives its resource demaind from its SLAs and requests resources from the automatic resource management system. The resource demaind maybe dynamically generated with the change in the service load. For example, the number of concurrent sessions in a global video conference dynamically changes with the entering and leaving of participants in the mid. A self-managing service should hide the impact of the load pertubation to deliver the QoS guaranteed service, which may dynamically change the resource demand of the service.

The automatic resource management system, denoted as ARMS, is mainly responsible for providing resource capacity guarantee for services based on the resource demand of services where a resource contract made between a service and the ARMS is given by a resource demand description. To fulfill this overall goal, the ARMS must meet several requirements. First, the resources managed by the ARMS may be geographically distributed across multiple administration domains and have heterogeneous types. The ARMS must provide a uniform resource abstraction for services to program resources and extract resource information. For example, a service can use the same way to deploy its service components or conduct file operations on heterogeneous computer nodes without knowing the resource heterogeneity. Second, to achieve resource capacity guarantee, the ARMS needs to monitor resources, detect resource failure/degradation, and automatically adjust resource allocation of a service. Third, when resources are sharing among multiple services, the ARMS must provide performance isolation on shared resources. Fourth, the ARMS should provide resource protection at service boundaries. At last, the pervasive services and geographically distributed resources entail the scaleability and reliability of the ARMS.

The ARMS has two types of APIs: (a) one for services to request, query, acquire, and release resources; and (b) another for the administrator to manage the ARMS. The current design and implementation of the ARMS mainly focus on the management of self-owned resources.

4 SYSTEM ARCHITECTURE OF THE ARMS

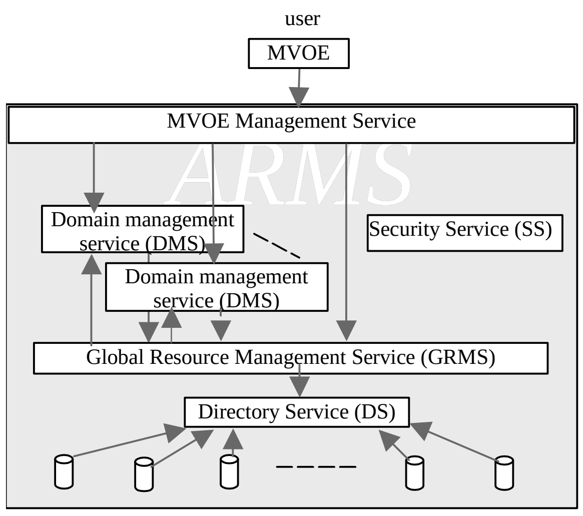
user
MVOE
MVOE Management Service
ARMS
Domain management service (DMS)
Domain management service (DMS)
Security Service (SS)
Global Resource Management Service (GRMS)
Directory Service (DS)

Figure 2. Architecture of the ARMS.

In the ARMS, all management functions are designed as web services. The system architecture of the ARMS is shown in Figure 2, which consists of six different types of services: MVOE management service, doman management service, security service, global resource management service, directory service, and workstation service. All the services run on top of service agents which virtualize the underlying service hosting node (the servie agent is described in the next section).

From the user's point of view, a virtual operating environment is provided for a user to access ARMS functions or for an administrator to manage the ARMS. In our long-term view, MVOE, a multimedia extension to the VOE [29], will be an universal multimedia interface for accessing, programming, and managing multimedia services in the msGrid. The current focus is on resource and user management functions. The MVOE management service provides the server-side management of user-profile customized MVOEs.

The security service is the only trusted identity in the system which provides authentication and access control management for other services. The ARMS uses a modified Kerberos authentication system and a rule-based access control system. The detail is explained in Section 7.

All dynamic and static information about free resources are kept by the directory service. Besides maintaining the static profile of a resource, the directory service is also responsible for monitoring resources. The directory service may maintain a distributed directory hierarchy for scaleability based on the distribution and the number of resources. In current implementation, only one directory is used. The directory service uses publish-subscrible model for workstations to publish their information into the directory and for the global resource management service to allocate resources.

Each domain management service is dynamically generated based on a resource demand description that is submitted by a resource user (or a self-managing service). The global resource management service will allocate resources from the global free resource pool into a domain management service. The domain resource management will be responsible for achieving the resource capacity guarantee by dynamically monitoring the resource capacity and adaptively adjusting its resource allocation from the global resource management service. Finally, the global resource management service will be responsible for dynamically adjusting the resource allocations among domain management services, trying to maxmizing overall system performance and profit.

During the deployment process, the security service is the first one to be deployed, which provides necessary access control and authentication for subsequent deployment of other services. For example, when the global resource management service is deployed, it must get a service ticket to the directory service from the security service before it can serve any resource requests.

5 WEB-SERVICE BASED RESOURCE VIRTUALIZATION

To manage various kinds of heterogeneous resources, it is critical to have a uniform abstraction to hide the resource heterogeneity to minimize the management complexity. The loosely-coupled, language-neutral, and platform independent features of the emerging web service standard [16][27] are critical to a uniform abstraction, which motivates us to abstract all resources and functions as Web services. To support the web service abstraction, we designed and implemented a general Web-service messaging and deployment engine, called service agent as shown in Figure 3.

The design of the service agent aims at achieving the extensibility, deployment automation, enforcement transparency, and resource virtualization for the ARMS.

In the service agent, the SOAP RPC message marshaling/de-marshalling engine, the service publish interface, and the local service publish engine constitute a core for supporting the Web-service model. The core is mainly responsible for publishing a Java class as a Web-service by generating its WSDL and providing it a SOAP RPC messaging interface.

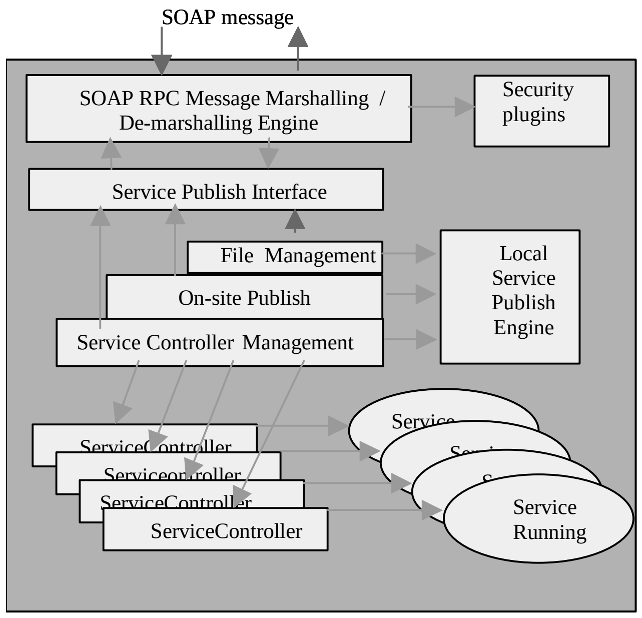

Figure 3. Architecture of the service agent.

To support automatic deployment, the service agent incorporates a file management service to provide platform-independent file operations on heterogeneous resources. The file management service provides basic operations for accessing files and directories. Based on the file management service, the on-site publish service moves a local service or a local Java class to a remote node and publish it there to execute.

To support the self-management capability of high-level services, the service controller management service is responsible for managing service controllers where each controller controls and monitors the execution of a service. The monitoring of a service is platform-dependent, such as reading resource usage.

6 AUTOMATIC RESOURCE MANAGEMENT

One of the major design goals of the ARMS is aiming at optimally achieving a cost-bounded capacity guarantee illusion for high-level services by dynamically adjusting the resource allocation to hide failures and load pertubation on resources. The ARMS uses a two-level automatic resource management approach:

the global resource management service and the domain (resource) management service, which is illustrated in Figure 4.

6.1 Domain management service

The domain management service works like an exclusive resource container for the self-managing multimedia service hosted by it. By cooperating with the global resource management service, a domain management service dynamically requests or trades resources with the aim to deliver the capacity guarantee at the minimal cost of the self-managing service. The domain management service is a distributed service which enforces a single sign-on secure domain using the security infrastructure presented in Section 7. The domain management service is a general management mechanism with a programmable interface for the high-level self-managing service to set management policies.

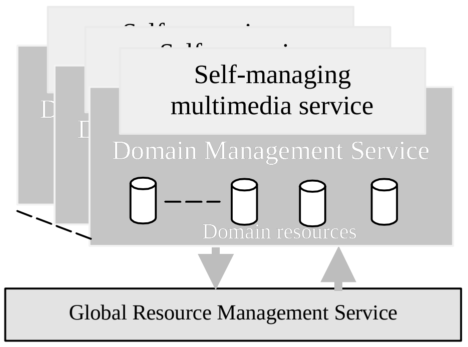
Self-managing multimedia service
Domain Management Service
Domain resources
Global Resource Management Service

Figure 4. A two-level resource management architecture.

The resource management automation in the domain management service is fulfilled by four service components: monitoring, capacity analysis, adjustment decision-making, and profile. The automation framework is given in Figure 5.

The monitoring component periodically collects local resource health information and returns a health report to the capacity analysis. What a health report should contain is programmable. The monitoring complexity mainly comes from the resource sharing. For a sharing computing node, the underlying operating system support is necessary. For network resources, end measurement is used when network switches cannot be fully controlled.

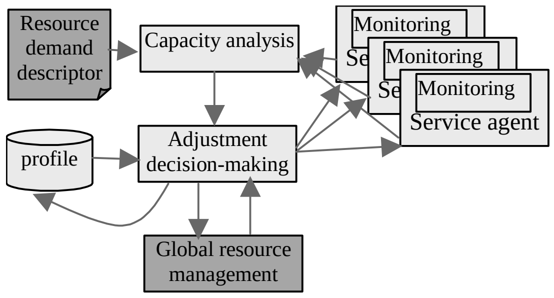
Resource demand descriptor
Capacity analysis
Monitoring
Service agent
profile
Adjustment decision-making
Global resource management

Figure 5. Automation framework of the domain management service.

The capacity analysis component collects health reports from all domain resources and periodically takes a snapshot of the domain resource capacity. If the capacity snapshot doesn't meet the capacity demand, it sends a capacity deflicit report to adjustment decision-making component.

The adjustment decision-making component estimates a new resource request to the global resource management service based on the profile and the capacity deficit report. The capacities of some sharing resources, such as network and commodity operating systems, are nondeterministic. Even though the resource reservation may be a solution, it is not cost effective or impractical in the grid which aims at support huge number of services in a global area. In addition, the resource fault is another nondeterministic factor to make it hard to guarantee resource capacity. Here, the profile is used to keep previous records of adjustment decisions and capacity deficits. The decision-making component enhances its decision precision based the profiled experience.

6.2 Global resource management service

The resource requests of domain management services are finally executed by the global resource management service (GRMS). The GRMS mainly conducts three functions through three components:

1. Scheduler: The scheduler allocates free resources from the directory service. A resource request could be any combination of different kinds of resources. The allocation is derived by specific performance metrics that are programmable. For example, the performance metric can be cost, profit, or the number of self-managing services.

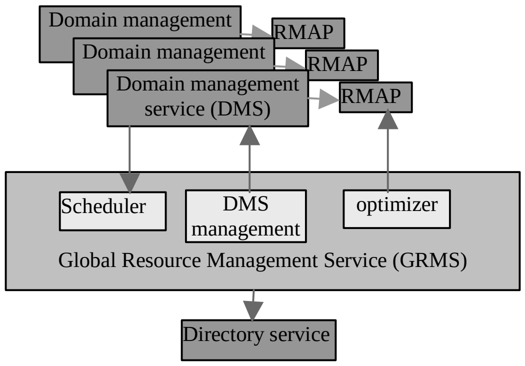

Figure 6. Component architecture of the global resource management service.

2. DMS management: The DMS management component is responsible for creating a DMS for an incoming domain request, conducting DMS placement, and monitoring DMS.
3. Optimizer: The optimizer aims at improving the resource allocation from the global view by periodically finding better resources in the global space or switching resources among different DMSs. A “better resource” is measured by a programmable metric.

7 SECURITY INFRASTRUCTURE

In the ARMS, services and resources are geographically distributed and tend to be vulunerable to all kinds of attacks. For a wide-area system, the security is a crucial system requirement. In order to provide a secure shell around the ARMS over geographically distributed resources and services while keeping a simple and stateless client, a security infrastructure composed of a Kerberos-like authentication system and a rule-based access control system is designed and implemented to achive the following goals:

- *Integrating authentication service into the ARMS system*.
- *Separating authentication infrastructure from ARMS functionality.*
- *Separating policy management from enforcement.*
- *Enforcing security transparently.*

7.1 Authentication

Authentication methods fall into two categories based on their key cryptography techniques: the symmetric key cryptography based authentication, such as the Kerberos authentication[4][24][26], and the asymmetric (or public) key cryptography based authentication, such as the certification based authentication [8]. The major differences between these two categories of authentication methods as summaried by B. Tung [26] are (a) the asymmetric key cryptography takes orders of magnitude longer to encrypt and decrpt the same amount of information than it does with symmetric key cryptography; (b) comparing to the primes based asymmetric key ciphers, such as RSA, symmetric key ciphers are much stronger; (c) the asymmetric key cryptography is more scalable than the

symmetric key cryptography in key distribution; (d) the asymmetric key cryptography needs some way to do key certification while symmetric key cryptography needs not; (e) in the asymmetric key cryptography based authentication method, a large number of certifications usually needs to be maintained by each service, which is not the case for the symmetric key cryptography based authentication method. These motivate us to build our authentication infrastructure based on the symmetric key cryptography.

Our Kerberos-like authentication system is a modification to the Kerberos authentication system with aims to facilitate a simple and consistent global user interface of the system while keeping the security strength of the Kerberos. Our authentication system uses Kerberos credentials: the ticket and the authentication. A ticket carries the KDC-proof identity information of a client for a specific service. An authenticator is a proof that the ticket is originally issued to the client, not stolen.

Different from the Kerberos, our authentication system uses intermediate-server based authentication protocols, which is required to build a single global system view. The functionalities of our authentication system are partitioned into three parts: client component, admin service, and KDC (key distribution center) service. The detail component functionalities and authentication protocols are described in this section.

7.1.1 Component Functions

To keep the Web service model as our single software development model, the KDC service is designed to be a Web service that manages two transaction-protected databases: authentication database and tickets database. The KDC service is the unique trust identity in the ARMS. All other principals are authenticated againist it. To ease the management of the authentication system, the KDC service is only accessible to the admin principals of the KDC. Initially, a default admin principal is registrated in the authentication database of the KDC. Admin principals are used by admin services that are integrated with the instances of the ARMS service.

The KDC service mainly performs three functions: (a) *Principal registration and update*. For each principal registration request, the KDC verifies the uniqueness of the principal name and generates a password-based private key into the authentication database. Initially, the KDC service registers itself into the authentication database and gets a password-based private key that is used for encrypting and decrypting database entries. (b) *Session ticket generation*. For a session request between a client and a service, the KDC verifies the principal names of the client and the service, and generates a random session key and a ticket for the communication between the client and the service. (The same reason as that in the private key generation, the instance addresses of a multi-instance service are not used in the session key generation.) (c)*Ticket renew*. The KDC renews expired tickets and invalidates old tickets in the ticket database.

To facilitate the uniform global user interface of the system, an admin service functions like a proxy of the KDC service by residing in the ARMS service. The admin service will be replicated while the ARMS service is replicated over multiple hosts for scalability. The admin service fulfills all the KDC service functions: principal registration and update, ticket request, and ticket renew for other services. Even though the admin service is integrated into the ARMS service, the admin service and the ARMS service are two different services where the later is authenticated through the former.

In the deployment, the first deployed instance of the admin service uses the default admin service name in the KDC service to get a service ticket to the KDC service. Then, the first instance updates the admin service name to include all instances.

The client component takes care of all client-side authentication functions: maintaining acquired service tickets and principal name, generating authenticators for requests, and renewing tickets.

7.2 Access Control

For management flexibility and scalability, the ARMS system uses the role-based access control model [22] in the web service model. In the role-based access control model, principals are separated from permissions on protected operations through roles so that principals and permissions on operations can be independently changed and managed.

Figure 7 shows the role-based access control model in the ARMS. Under the web service model used in the ARMS, the service is the basic access control unit in the model. The model has five major parts: principals, permissions, roles, permission-to-role mapping, and principal-to-role mapping. Permissions to a service refer to operations of the service.

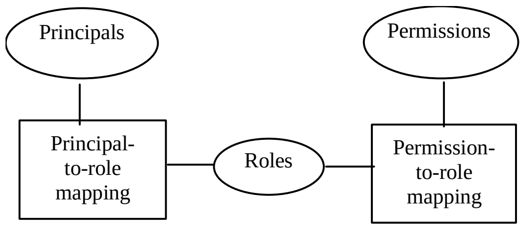

Figure 7. Access control model of the ARMS.

7.2.1 Roles

Each service has an 1-dimensional role space that contains all roles specific to the service. In the ARMS system, the role spaces of all services are kept in a role file in the directory service that is only accessible to the admin of the ARMS system.

7.2.2 Principal-to-Role Mapping

If A principal is able to access a service, the principal must have been assigned to a role in the role space of the service. The assignment of principals to roles is managed by the ARMS while principals request service tickets. The ARMS system embeds the assignment of a principal to roles in the service ticket. The service ticket is transferred in an encrypted form so that the assigned roles can’t be changed by malicious users.

7.2.3 Permission-to-Role Mapping

Access control on the operations of a service is guarded by an 2-dimensional permission-to-role mapping matrix that defines which roles can perform which operations. The permission-to-role mapping matrix is managed as a service permission file by the ARMS service.

In a service, access control is conducted only when the authentication verification has been passed because the roles are embeded in the service ticket.

7.3 Security Mechanisms: Passport and Service Guard

To support distributed authentication and access control, two districuted mechanisms: passport and service guard have been introduced. To achieve enforcement transparency and the separation of policy and enforcement in Web service messaging engine, both mechanisms are implemented as security plug-ins in the messaging engine as shown in Figure 8.

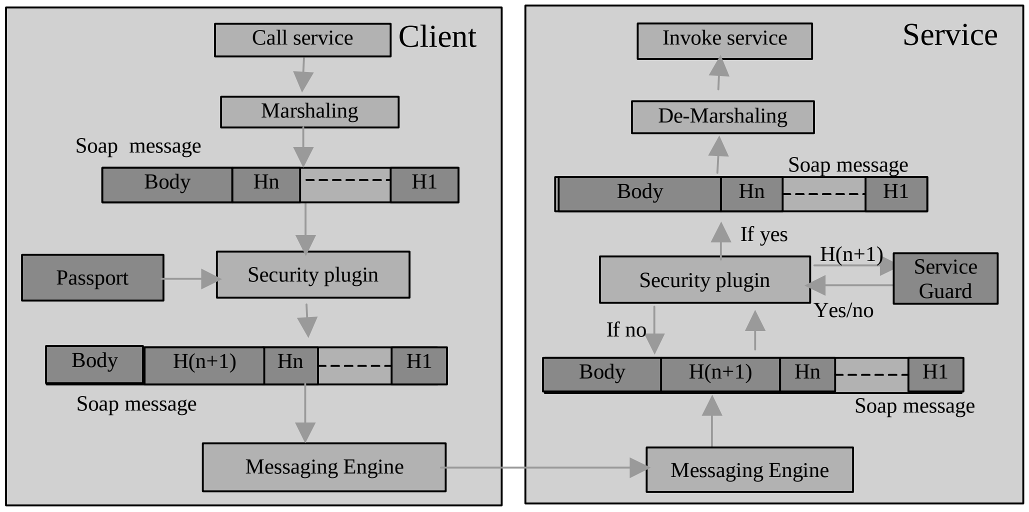

Figure 8. Transparent authentication and access control procedure in Web service messaging.

7.3.1 Passport

The passport is the client-side security module which consists of an authentication information table (AIT) for all the services requested by the client and an message processing engine.

For each service requested by the client, the AIT maintains an entry of the service URL, the service session key, and the service ticket. The current service entry is pointed by the current service index.

The message processing engine is responsible for encapsulating authentication information into the request message. For each service request, the message processing engine first extracts the session key and the service ticket from the AIT for the requested service. Secondly, it generates an authenticator encrypted by the session key where the authenticator contains client name, client IP address, and the MD5 checksum of the message. Last, it encapsulates both the encrypted authenticator and the service ticket in a SOAP message header (which is shown as the header H(n+1) in Figure 8).

7.3.2 Service Guard

The service guard is the service-side security module which is responsible for the authentication and the role-based access control for the service. The service guard consists of the service private key, a service permission table, an authentication engine, and an access control engine. The service permission table is set by the ARMS service based on the permission file of the service, which assigns permissions (or operations of the service) to roles. This separates permission management from the access control for better manageability and scalability.

For an incoming SOAP request message to a service, the service guard of the service is triggered to conduct authentication and access control.

First, the authentication engine extracts the authentication message header from the incoming SOAP message. It decrypts the service ticket using its private key, decrypts the authenticator using the session key in the ticket, and conducts authentication by performing the following verification:

- *Data integrity verification*: the authentication engine computes the MD5 checksum of the message and compares it with the checksum in the authenticator. If results don't match, the service guard sends authentication failure exception back to the client.
- *Client principal authentication*: the authentication engine validates the authenticator and the service ticket based on their timestamps. If valid, the client name and IP address in the authenticator are verified against that in the service ticket. If failed, an authentication exception is sent back to the client.

Secondly, after the authentication passes, the access control engine extracts the roles embedded in the service ticket. It conducts access control by verifying the roles against the service permission table. When the access is allowed, the requested service operation is finally invoked.

8 IMPLEMENTATION STATUS

The ARMS is a work-in-progress project. The first version of the ARMS system was implemented using pure JAVA based on the Glue 2.3 Web service engine from The Mind Electric Inc. [15]. GLUE was designed and built 100% around web services standards like XML, SOAP, WSDL and UDDI [16]. The first prototype is for the proof of concept. Right now, we are rebuilding the system based on open technologies: Globus toolkit, Java, J2EE, and Apache technologies.

The directory service was implemented using the LDAP with the web-service interface using the GLUE. The directory service maintains resource profiles (only workstation resources have been considered) and user profiles.

The MVOE was a swing-based interface which could run as an independent application or an applet in the browser.

Our Kerberos-like authentication system was built based on the modification to The Kerberos V5.0 [18]. We adopted the basic mechanisms: authenticator, ticket (with an extension to include principal roles for role-based access control), password-based private key generation algorithm, session key generation algorithm, and checksum generation algorithm. We rebuilt the key distribution center (KDC) and implemented our new authentication protocols. The KDC was built as a Web service using Glue and Bekelery-DB 4.0.14 [4]. The KDC contains two full transaction protected databases: authentication database for storing private keys and ticket database for storing tickets and session keys. Both databases support locks for consistency control. All data entries were encrypted using the private key of the KDC service.

The access control functionalities were implemented in three system components: the MVOE management service, the KDC, and the service guard. The MOVE management service manages the role file and service permission files in the directory server. The assignment of principals to roles was implemented in the MOVE management service. The KDC encapsulates the assignment of a principal to roles in the service ticket. The access control were enforced in the service guard of the service.

The passport and the service guard were plugged into the SOAP messaging engine using Glue's interceptors. By registering an instance of the interceptor using appropriate Context object, the interceptor instance can intercept SOAP message and invokes the processing engines of the passport and the service guard.

The interceptor instance with a passport is registered with the sendRequest of a client so that each outstanding request from this client is intercepted and processed by the passport engine. The interceptor instance with a service guard is registered with the receiveRequest of a service so that each incoming SOAP message is intercepted and processed by the service guard.

The implementation of the domain management service is in progress, which will be fully reported with some performance data in the future.

9 CONCLUSION

In this paper, we introduce the design and current implementation status of an Web-service based automatic resource management system, named the ARMS. This system is designed to seamlessly harvest distributed heterogeneous resources to automatically meet various kinds of dynamic resource demands of rich media services. This system aims at providing general resource management mechanisms which can be programmed using service specific policies.

In order to support self-managing rich-media services, the ARMS comes up with a self-managing resource management mechanism: domain management service (DMS). The DMS cooperates with the global resource management service (GRMS) to achieve demand-driven management automation. The GRMS is responsible for managing global free resources.

In order to protect geographically distributed resources and services in the system, the ARMS has an extensible security service that seamlessly integrates Kerberos-like authentication with a role-based access control model. The authentication system is modified from the Kerberos so that (a) the authentication system can be integrated into the resource management sysem, instead of working as an external component, for better management capability; (b) the client has single access point to the resource management system which allows stateless client interface. To achieve enforcement transparency, two secure mechanisms: passport and service guard have been introduced to transparently integrate the authentication and access control functions into the web service model.

This system is still working-in-progress. Many problem still remains to be solved, such as those policies and security in the resource management.

ACKNOWLEDGEMENTS

Here, I would like to thank Bruce Blaho and Micheal Goss for their suggestions and discussion on the security work.

REFERENCES

[1] Apache Software Foundation. Apache web server. http://www.apache.org.

[2] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and M. Kalantar, "Océano – SLA based management of a computing utility," *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[3] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, D. Simon, Web Service Security (WS-Security): Version 1.0 05, April 2002. ftp://www6.software.ibm.com/software/developer/library/ws-secure.pdf.

[4] Berkeley DB Tutorial and Reference Guide, Version 4.0.14. *http://www.sleepycat.com/docs/reftoc.html*.

[5] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch, "A national-scale authentication infrastructure," *IEEE Computer*, 33(12):60-66, 2000.

[6] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP),* October 2001.

[7] B. Clifford Neuman and Theodore Ts'o. "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications*, 32(9):33-38. September 1994.

[8] T. Dierks and C. Allen, "The TLS protocol: Version 1.0," *RFC2246*, January 1999.

[9] R. Fielding , J. Gettys, J. Mogul, H. Frystyk , T. Berners-Lee. "Hypertext Transfer Protocol—HTTP/1.1," *RFC2068*, January 1997.

[10] J. Franks , P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart. "An Extension to HTTP: Digest Access Authentication," *RFC2069*, January 1997.

[11] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, 15(3), 2001.

[12] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," *http://www.globus.org/research/papers/ogsa.pdf*, January, 2002.

[13] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, "Grid Services for Distributed System Integration," *IEEE Computer*, June 2002.

[14] I. Foster, C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit**,”** *Intl J. Supercomputer Applications*, 11(2):115-128, 1997.

[15] Glue, The Mind Electric Inc. *http://themindelectric.com/docs/glue/ guide/index.html*.

[16] S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, and R. Neyama. Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, *Sams*, 2001.

[17] K. Krauter, R. Buyya, and M. Maheswaran, “A Taxonomy and Survey of Grid Resource Mangement Systems for Disitributed Computing,” *Software – Practice and Experience*, 32(2):135-164, 2002.

[18] J. Kohl and C. Neuman, “The Kerberos Network Authentication Service (V5),” *RFC1510*, September 1993

[19] J. Novotny, “The Grid Portal Development Kid,” Concurrency: Practice and Experience,

[20] J. S. Park and R. Sandhu, “Role-Based Access Control on the Web,” *ACM Transactions on Information and System Security*, Vol. 4, No. 1, February 2001.

[21] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, “Virtual Network Computing,” *IEEE Internet Computing*, Vol. 2 No. 1, Jan/Feb 1998, pp. 33-38.

[22] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. youman, “Role-based Access Control Models,” *IEEE Computer*, Feb. 1996.

[23] Security in a Web Services World: A Proposed Architecture and Roadmap. A joint security whitepaper from IBM Corporation and Microsoft Corporation: Version 1.0, April 2002. ftp://www6.software.ibm.com/software/developer/library/ws-secmap.pdf

[24] J. G. Steiner, B. Clifford Neuman, and J. I. Schiller. “Kerberos: An Authentication Service for Open Network Systems,” *Proceedings of the Winter 1988 Usenix Conference*. February, 1988. (Version 4) text , postscript .

[25] Stone Cold Software. Apache Kerberos Module. *http://modauthkerb.sourceforge.net/*.

[26] B. Tung. Kerberos: A Network Authentication System. Published by Addison Wesley Longman, Inc., 1999.

[27] Web Service Architecture Requirements, W3C Working draft 29, *http://www.w3.org/TR/2002/WD-wsa-reqs-20020429#N100CB*, April 2002.

[28] J. Wilkes, J. Janakiraman, P. Goldsack, L. Russell, S. Singhal, A. Thomas, “Eos, the Dawn of the Resource Economy,” *HotOS’2001*, May 2001.

[29] Y. Yan and Z. Xu, “Functional Architecture of AMSES: An Automatic Management System for E-service Systems,” *HP Lab Technical Report*, HPL-2001-185, July 25, 2001.

[30] Y. Yan and X. Zhang, “Profilt Effective Paralle Computing,” *IEEE Concurrency*, April/June issue, 1999.

[31] Huican Zhu, Hong Tang, and Tao Yang, “Demand-driven service differentiation in cluster-based network servers,” *Proceedings of IEEE Infocom 2001*, April 2001.