THE SECURE ABELS BROKERING SYSTEM

Linda F. Wilson, W. Riley Lochridge, and G. Ayorkor Mills-Tettey
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755-8000 USA
E-mail: Firstname.Lastname@dartmouth.edu

KEYWORDS

Distributed simulation, brokering systems, security, dynamic information exchange, software agents, simulation tools.

ABSTRACT

The Agent-Based Environment for Linking Simulations (ABELS) system is a software framework whose goal is to enable independent simulations and other data resources to exchange information dynamically without prior knowledge of each other. Specifically, it enables the dynamic formation of a "cloud" of simulations and other networked resources such as sensors and databases. This data and simulation cloud uses a distributed brokering system to match data consumers in the cloud with appropriate data producers, based on registration information submitted by the various participants in the cloud.

In a system of interacting independent resources, there are several security concerns, including how to prevent undesirable entities from joining and participating in the cloud, how to protect sensitive information, and how to ensure the integrity of the cloud so that it functions reliably. This paper presents the redesign of the ABELS brokering system to incorporate security features that address these concerns.

1. INTRODUCTION

The Agent-Based Environment for Linking Simulations (ABELS) system is a software framework whose goal is to enable independent simulations and other data resources to exchange information dynamically without prior knowledge of each other (Kumar et al. 2002; Mills-Tettey and Wilson 2003a; Mills-Tettey and Wilson 2003b; Wilson et al. 2001). The ABELS system allows a collection of independent networked resources to associate with each other in what is referred to as a data and simulation *cloud*. The autonomous resources are producers and/or consumers of data. Individual cloud participants join and exit the data and simulation cloud as needed and have no prior knowledge of the other cloud participants. A distributed brokering system is used to match data producers to consumers and initiate communication between cloud participants, but it does not control the independently-designed participants in any way. Each organization participating in an ABELS cloud is responsible for determining what resources it makes available to the cloud and which data consumers are eligible to access its services.

The ABELS system is designed for loosely-coupled interactions between participants. That is, participants in the cloud are not required to conform to a common stringent standard, and consumers are not statically linked to particular producers of information. In addition, there are no tight interdependencies among cloud participants.

There are many security concerns inherent in the ABELS system. These include concerns about how to protect undesirable entities from joining and participating in the data and simulation cloud, how to protect sensitive information produced by services in the cloud from being accessed by unauthorized entities, and how to ensure the integrity of the cloud as a whole so that it functions reliably. To mitigate the security threats in the system, the ABELS architecture must include mechanisms for access control, privacy and integrity, and logging. In addition, each cloud participant must be able to define its own security needs and capabilities, and the cloud must guarantee that each participant's security requirements are met.

As discussed in Mills-Tettey and Wilson (2003a; 2003b), we have recently redesigned the ABELS system to incorporate appropriate security mechanisms. Security experts have noted that security must be designed into a system's architecture rather than added at a later date. Thus, security features have been integrated throughout the ABELS system.

This paper describes the redesign and implementation of the secure ABELS brokering system, which is responsible for maintaining a database of cloud participants and matching data consumers with appropriate data producers. Section 2 describes the various components of the ABELS system, while Section 3 discusses related systems and Sun Microsystems' Jini technology, which is used by ABELS. Section 4 presents the design of the secure brokering system while Section 5 discusses its implementation. Finally, Section 6 presents conclusions and areas for future work.

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)

ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

2. THE ABELS SYSTEM

2.1. Overview

The Agent-Based Environment for Linking Simulations (ABELS) is a software framework that enables the dynamic formation of a cloud of autonomous simulations and other data resources, which can then interact without prior knowledge of other cloud participants. As shown in Figure 1, the ABELS system architecture consists of three basic types of components: user entities, generic local agents (GLAs), and a distributed brokering system. An optional user interface is provided to permit human interaction with the system via the GLAs.

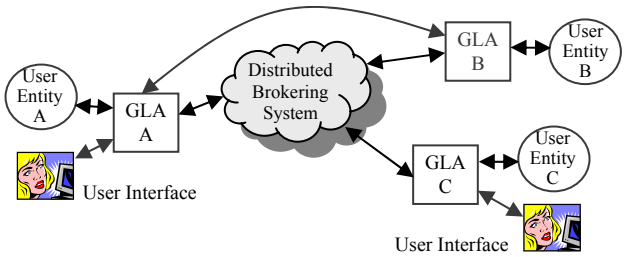

Figure 1: Basic Framework Connecting Elements in the Cloud

The cloud participants are the *user entities* which are producers and/or consumers of data. For example, a cloud might include simulations, databases, and sensors. A data producer provides a *service* with one or more *service functions*, while a data consumer makes requests or *queries* for information. User entities join and exit the data and simulation cloud as needed and have no prior knowledge of the other cloud participants. Each organization participating in an ABELS cloud is responsible for determining what resources it makes available to the cloud and describing those services accurately.

A user entity connects to the cloud via a software agent known as the *generic local agent* (GLA). Each GLA serves as the cloud interface for one or more producer and/or consumer entities, owned and operated by the same organization. The user entity uses its GLA to join or exit the cloud, register its services and service functions, and make queries for data. A data producer is said to interface with the cloud via a *producer GLA*, while a data consumer participates via a *consumer GLA*. The GLA is responsible for adapting information to the specific needs of its producer or consumer entities. For example, a consumer GLA is responsible for handling any data format, unit, and file conversions that are necessary between its consumer and the producer that is serving it. A producer GLA is responsible for passing input data to a desired service, executing the desired service function, and returning the output data to the corresponding consumer GLA.

The *distributed brokering system* forms the core of the cloud and is responsible for managing cloud resources and linking consumers to appropriate producers of information. In particular, it stores descriptions and remote references or *proxies* for all of the resources in the system, and it uses a system of leases to determine which participants are still in the cloud. Furthermore, it matches and ranks those service functions that may satisfy a given consumer's query. Once the brokering system links a consumer GLA with a corresponding producer GLA, communication occurs directly between the GLAs without going through the brokering system. In Figure 1, GLAs A and B have a direct connection, indicating that a match occurred between a producer on one end and a consumer on the other end. The brokering system is described in detail beginning in Section 4.

A key feature of ABELS is its runtime matching of consumers with suitable producers, and this matching is based on textual descriptions of the participants. A service definition includes the name, location, and description of the service along with detailed information about the functions it provides. A service definition also specifies one or more high-level categories or *groups* that characterize the service. For example, a service would belong to the "weather simulations" group if it provides information about the weather conditions in Hanover, New Hampshire. A query is defined as the ideal function desired by the consumer. For a given consumer query, ABELS must evaluate all service functions that may match the query and rank those functions so that the most suitable one is chosen to resolve the query. A query is resolved using the best-available service function, so a query resolved by one function today may be resolved by a different function next week.

For better accuracy and efficiency, there are two levels of matching in the ABELS system. The first-level matching is based on the groups of interest for the given query. That is, the brokering system generates a list of those services that belong to the groups specified in the query definition. In the second-level matching, the details of the query are compared with all of the service functions belonging to the list of services, and the service functions are ranked according to their suitability for the query. Details of the matching and ranking process can be found in Peteet et al. (2003). Additional information on the ABELS system can be found at http://thayer. dartmouth.edu/~abels.

2.2. Security Issues

As discussed in Mills-Tettey and Wilson (2003a; 2003b), we have recently redesigned the ABELS system to incorporate various security features. In particular, the participants may have varying security needs, and the cloud must guarantee that each participant's security requirements are met.

The specific security capabilities in ABELS can be classified as access control capabilities, privacy and integrity capabilities, and logging capabilities. Access control consists of the related processes of authentication and authorization. Authentication verifies the identity of an entity while authorization determines what permissions are granted to the authenticated entity. Privacy protects sensitive information from being viewed by unauthorized parties, while integrity provides the ability to detect and protect the information from tampering. Finally, logging is used to detect and audit a security breach.

Security features in ABELS appear at two levels: *centralized* aspects are handled by the brokering system, while *decentralized* aspects are managed by the individual GLAs in the cloud. The brokering system security focuses primarily on ensuring the integrity of the cloud, while the GLAs provide the primary means of protecting the entities participating in the cloud. In particular, a GLA may authenticate the users that wish to use it to communicate with the cloud, to prevent access by unauthorized users. In addition, a producer GLA may place limits on the consumer GLAs that wish to use its services. The brokering system may authenticate GLAs before allowing them to join the cloud, in order to prevent unauthorized participation by an illegal GLA. Encryption can be used to protect sensitive information passed between a GLA and its user entities, between two GLAs, or between a GLA and the brokering system. Finally, both the brokering system and the GLAs log all successful and unsuccessful attempts to add GLAs to the cloud or connect user entities to the GLAs.

The complete ABELS security framework is discussed in Mills-Tettey and Wilson (2003a; 2003b). This paper presents the architecture of the secure brokering system, and specific security details are given in Section 4.3.

3. RELATED WORK

3.1. Other Approaches

ABELS is not the only system designed to create a dynamic, distributed community of heterogeneous entities, simulations, and services. Other approaches include the High Level Architecture (HLA) and the web services architecture.

The High Level Architecture (HLA) (Dahmann et al. 1998) is a software architecture for creating a federation of simulations communicating across a single runtime infrastructure (RTI). HLA is designed to support tightly-coupled interactions between simulations. A set of rules specifies the object models to which federations (communities of simulations) and federates (individual simulations) must conform in order to work with other entities in the system. To participate in an HLA federation, simulations must be written to conform to the federation object model (FOM).

Unlike HLA, the web services architecture (Curbera et al. 2002) provides a mechanism for communication and data exchange between loosely-coupled entities. WSDL (Web Service Description Language), UDDI (Universal Description, Discovery and Integration), and SOAP (Simple Object Access Protocol) are three XML-based protocols that enable the description, registration, and invocation of web services. XML provides a language- and platform-independent communication mechanism that is an important feature of the web services architecture. Missing in the current architecture, however, is a mechanism for the runtime brokering between information consumers and producers that would allow the transparent replacement of one service provider with another one.

The ABELS system described in this paper targets loosely-coupled simulations and data resources, requiring little or no changes to existing simulations. Developed with Java and Jini, ABELS is designed to be platform-independent and currently is being developed on both UNIX and Windows systems. A brokering system is used to perform runtime matching of data producers and consumers, and software agents act as interfaces through which entities can participate in the ABELS cloud. These software agents enable the ABELS system to adapt to a wide range of simulations and resources without requiring these entities to be written in a specific language.

3.2. Jini

Our distributed brokering system is developed in part using Sun Microsystems' Jini technology (Kumaran 2002). Jini is a protocol-independent, Java-based programming model that simplifies the development of a system of distributed services. In particular, the Jini application programming interface (API) handles tasks associated with the discovery and lookup of distributed services and the description of service attributes. Jini also provides event listeners and leases that are used to manage the networked resources. The ABELS brokering system uses the building blocks provided by the Jini architecture and its reference implementation to manage the cloud resources and match data consumers with suitable producers.

4. THE SECURE BROKERING SYSTEM

4.1. Overview

The ABELS system links loosely-coupled data resources to form a distributed data and simulation cloud. The brokering system serves as the central framework through which the cloud is formed and maintained. To ensure efficiency and fault tolerance, the brokering system is designed using distributed database principles.

As discussed in previous work (Kumar et al. 2002; Wilson et al. 2002), the ABELS brokering system must serve as a database of participating services and match consumers with suitable producers. However, a better analogy compares the brokering system to a public or school library. Just as a library maintains and organizes its collection of books, the brokering system stores and categorizes each of the services registered within it. Just as books are organized into broad subject categories, services are organized into high-level categories called *groups*. Just as a librarian may take a reader's description of interests and recommend an appropriate book, the brokering system uses a detailed description of the desired service to match the consumer to the producer that best meets its needs. Furthermore, just as a borrower must obtain a borrowing card and present it to check out a book, so must a producer or consumer undergo identity-based authentication and authorization before joining or using services of the cloud.

The brokering system consists of the *broker*, the *matching and ranking system,* and the *keyword* and *conversion databases*. The broker manages the resources in the cloud, the matching and ranking system evaluates producers on behalf of consumers, and the keyword and conversion databases provide information needed for the matching process. The broker is implemented using Java and Sun's Jini technology, and the other components are implemented using Java.

The current implementation of the ABELS brokering system includes several Jini lookup services, each of which can run on a separate machine and supports several high-level categories called groups. The lookup service plays the role of registrar, or card catalog, for the cloud, managing a persistent database of all cloud participants by storing their service descriptions and proxies, which are described in Section 4.2.

The lookup service facilitates automated system startup and cloud formation by utilizing Jini's *discovery* and *join* protocols. It does this by providing its proxy to all GLAs as they join the cloud, allowing them to initiate contact at any later time. The lookup service also provides the means by which ABELS maintains a robust and self-healing network.

Due to the potential volatility in any distributed network, the ABELS cloud must be long-lived and resistant to sudden changes in the system that could initiate a system crash. The lookup services help ensure this by storing registration information for all GLAs in the cloud and discarding service proxies of lost GLAs. The lookup services support the join protocol, which assigns unique service IDs to all cloud entities. All GLAs in the cloud must conform to this protocol and maintain, for each service, the service ID, the list of groups it wishes to support, its Jini lookup entries, and the set of locations of the lookup services in which it has registered (Li 2000). The lookup services also grant Jini *leases* for an entity's GLA once it has joined the cloud. A lease represents the GLA's proof of interest, allowing the entity to remain registered in the cloud as long as its lease has not expired. The GLA must renew its lease within the finite lease duration, according to the terms set by the lookup service. Note that the administrative overhead of handling leasing is distributed away from any one central component by placing it in the lookup services, thereby increasing fault-tolerance and reducing the potential for bottlenecks.

4.2. Services and Queries

Having joined the ABELS cloud, a producer GLA may register any of its services with multiple Jini lookup services, based on the groups the services wish to join. To locate the lookup services with which it wishes to register, the GLA contacts the broker's communication module, which finds and returns the locations of the lookup services that support the desired groups. Before a service can be registered, the GLA must provide the lookup service with the following information: the group(s) it wants to support, a service description, and a service proxy object that is used by a consumer GLA to access the service. Within each service description there may be multiple function descriptions. In each function description, the user defines a sequence of input and output variables, including information on data types, units, ranges, and subsets. A query is a description of the ideal service desired and is in the form of a function description.

Similarly, a consumer GLA that has joined the cloud can obtain the locations of lookup services that support its desired groups. After receiving the locations of the lookup services, the consumer GLA contacts them to obtain information on the services belonging to the groups of interest. This is the first level of the two-level process that matches producers to consumers. Specifically, the consumer GLA receives the service registration and service proxy information for each service that might meet the needs of the consumer's query. A standing request is also left with the broker; if a service that supports the desired group(s) later joins the cloud, the system informs the consumer GLA via remote event handlers. Thus, the consumer GLA knows at all times which services of interest are currently in the cloud. With this information, the consumer GLA interacts with the matching and ranking system for the second-level matching. In order to understand this process, a brief description of the matching and ranking system is in order.

Matching and ranking determines the service function that best resolves a given query (Peteet et al. 2003). The matching and ranking process is designed as part of the brokering system but is implemented in the GLA in order to optimize performance and facilitate cloud scalability. The matching and ranking process is

based on the service description described above, making syntactic consistency critical. Accordingly, the human user can use the keyword database to determine which keywords are appropriate, and this will increase the likelihood of a suitable match.

The loosely-coupled nature and runtime brokering capability of ABELS differentiate it from related approaches like HLA and the web services framework. ABELS is a loosely-coupled system in which a service function is matched to a query exclusively on the basis of inputs, outputs, and a user-defined description. As a result, the matching process abstracts both services and queries from their implementation details, thus allowing a multitude of different implementation methods. The runtime brokering of services to queries allows the system to adapt to changes in network availability, providing a consumer with the best possible service available at the time of query resolution.

For each service function in the groups of interest, the matching and ranking system provides the consumer GLA with a numerical rank between 0.0 and 1.0, with 1.0 being a perfect match with the function. The rank is a weighted average of several factors that reflect various aspects of the fitness of a function for the particular query. While default weights are typically used, the human user may also customize the factor weights for his particular needs. To augment the ranking, a user may designate a service as *preferred, deprecated,* or *unsuitable.*

When resolving a query, the consumer GLA first attempts to use the highest-ranked preferred service. If that service is unavailable, it goes through the preferred services in descending order by rank. If no preferred services are available, the consumer GLA tries to connect to unmarked services in descending order of rank. If the query is still unresolved, the process continues through the list of deprecated services. If the query is still unresolved at this point, it will fail; unsuitable services are never used to resolve the query.

Once the consumer GLA has connected to a producer GLA using the provided service proxy, communication occurs directly between the two GLAs. If the service goes down during the process of information exchange, the consumer GLA will once again attempt to resolve the query, starting with the best-available service function.

4.3. Brokering System Security

As described in Mills-Tettey and Wilson (2003a; 2003b), security in ABELS is designed with a two-tier approach. The brokering system provides *centralized* security, restricting access to the cloud and maintaining cloud integrity. The brokering system is responsible for determining which GLAs may join the cloud and what permissions they have as cloud participants. This determination is known as *access control* and is comprised of the related concepts of authentication and authorization. Brokering system authentication entails the verification of a GLA's identity, while authorization determines the specific permissions granted to the GLA. The GLA itself provides *decentralized* cloud security that prevents malicious use of the GLA as an entry point for attack.

An ABELS cloud is classified as *open, semi-open, semi-closed,* or *closed*, depending on the security protocols established when the cloud is initially created. The human user creating the cloud, known as the cloud administrator, is responsible for selecting the cloud's security protocols. The cloud administrator is assisted in this process by using a graphical user interface (GUI).

In the *open* cloud, all producers and consumers must conform to specified IP address restrictions. For example, an academic institution may wish to allow any user connecting from within its IP domain to connect to the cloud while all IP addresses outside this domain are denied. While restrictions are required, they can be set to allow all IP addresses.

All cloud classifications use IP restrictions, and additional forms of authentication are used in the semi-open, semi-closed, and closed clouds. In these three cloud classifications, the default authentication method is the use of X.509 digital certificates issued by a trusted authority (usually the brokering system, although this is customizable).

In a *semi-open* cloud, all producer GLAs must undergo identity-based authentication and authorization by presenting a digital certificate. The digital certificate is verified by the broker's access control module, which is described in Section 5.1. Once the producer GLA is authenticated, the access control module checks the IP address restrictions in place, and if the entity conforms to the restrictions, it joins the cloud. Consumer GLAs, however, are subject only to IP restrictions when joining the cloud. Within such a cloud, the brokering system effectively guarantees that data obtained from the cloud is from known producers while consumers may participate under less stringent requirements. This is analogous to the access control used for a professional organization's web site, such as IEEE XPlore (ieeexplore.ieee.org), in which a data producer is guaranteed by the organization, while any user logging in from a member institution's IP domain is granted access.

In the closed and semi-closed clouds, both producers and consumers must authenticate themselves using digital certificates. Both clouds require identity-based authorization, but it is in the requisite GLA security functionality that they differ. In a *semi-closed* cloud, a participant GLA authenticates all its human users, but does not necessarily authenticate all its

consumer entities or encrypt service input and output information, as described in Mills-Tettey and Wilson (2003a; 2003b). This provides an environment in which the cloud guarantees the legitimacy of information provided but not necessarily its privacy.

In a *closed* cloud, the GLA authenticates all entities, and all information passed between producers and consumers is encrypted. The GLA's mandatory authentication of all entities adds legitimacy to the service registration and other information the GLA provides to the cloud. The encryption ensures that unauthorized users do not obtain sensitive data. Closed clouds are appropriate for environments where security and privacy of data is of great importance, such as at a military research agency.

When creating the cloud, the cloud administrator may select from a list of available authentication methods. As described in Mills-Tettey and Wilson (2003a; 2003b), ABELS currently supports digital certificates, username/password pairs, Windows, Unix, and Kerberos authentication methods. Additional authentication features may be added at a later date.

Once approved by the brokering system for admission to the cloud, the GLA receives a Kerberos-style ticket, called an entry ticket, which is presented to any ABELS component or entity in all subsequent interactions. This ticket proves that a user has been authenticated and authorized to join the cloud, and it functions in the same way as a library card, verifying the entity's identification and authorization and enabling it to participate within the cloud and carry out its actions. This process is shown in Figure 2, in which GLA A and the access control module must authenticate with one another using digital certificates. Once authenticated, the access control module checks that GLA A conforms to the IP restrictions and, if authorized, returns an entry ticket to the GLA. This figure also mentions the lookup service references (proxies), which would be returned to the GLA in response to a given query, as described in Section 5.3. Finally, with the lookup service proxies in hand, the GLA contacts Lookup Service 1 to either register a service or perform a first-level lookup after first presenting its entry ticket to the lookup service.

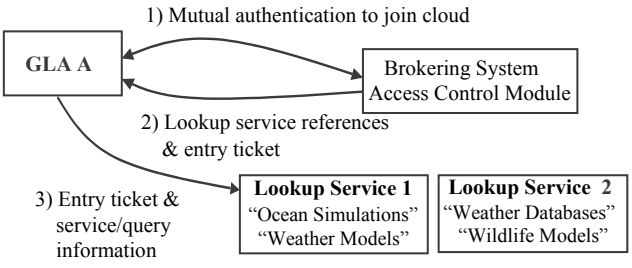

Figure 2: GLA Authentication, Authorization, and Cloud Participation

As mentioned previously, the brokering system has been redesigned to incorporate various security features. This redesign includes a shift from multicast functionality to an exclusively unicast system. Unless otherwise specified, Jini uses multicasting in its join and discovery protocols. Multicasting is a broadcast-based form of communication in which any computer listening on a prespecified port (e.g., 4160 is used by Jini) will receive all packets sent. While multicast allows for easy bootstrapping and discovery of available resources by new entities, it represents a major security hazard in that any entity listening to port 4160 would receive Jini *announcement* packets describing the current cloud composition in terms of lookup service locations and available groups. Because of the information leak this represents, we determined that only unicast communication should be used in ABELS. The security benefit in switching from multicast to unicast is analogous to a person announcing his whereabouts and activities on a telephone versus a megaphone. While a telephone (and unicast) are not invariably secure, using a single channel approach requires that someone proactively tap the phone line (or intercept packets) in order to hear what is being shared. The unicast functionality is one aspect of the comprehensive, two-tier security framework for the cloud and its participants.

5. IMPLEMENTATION AND DESIGN

5.1. Components of the Broker

The broker is designed in a modular manner, where each component has a distinct and specific functionality. These components interact with one another, the lookup services, and the rest of the ABELS system on an as-needed basis. With the modular design of the broker, future changes in the functionality may be easily implemented by adding a component without changing the entire structure of the system. As shown in Figure 3, the core implementation of the broker consists of six components: communication module, access control module, load balancing module, data recovery module, resource manager module, and meta lookup service. Figure 3 also shows the Jini lookup services (LUS) used by the broker.

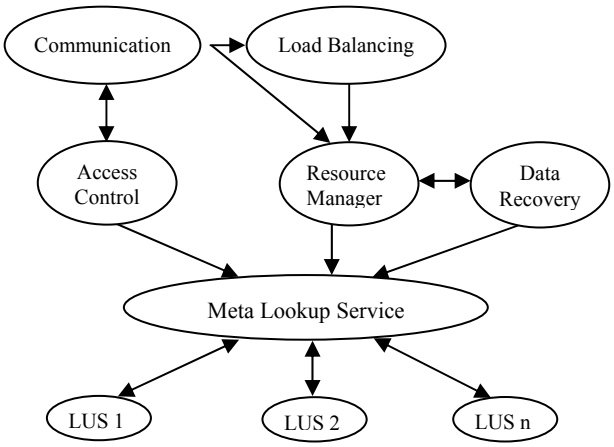

Figure 3: Components of the Broker

The *communication module* of the broker handles all communication coming into or leaving the broker. The communication module provides three main functions:

- Process join and exit requests sent by a user entity's GLA
- Process service registration requests from data producer GLAs
- Process consumer GLA requests for lookup services that support the group(s) of interest.

Upon receiving a join request, the communication module delegates authentication of the connecting GLA to the access control module. Once an entity has successfully joined the cloud, it can send other messages to the communication module. On receiving a message, the communication module verifies the sender's entry ticket to confirm that the sender is a valid cloud participant. If the sender is approved, the communication module examines the tag appended to the entry ticket, since the tag specifies the type of communication contained within the message. Based on the tag, the communication module routes the message to the appropriate broker module and, if necessary, later returns a reply to the sending GLA.

The *access control module* is used to authenticate the identity of a GLA and control access to resources in the cloud. This module is called when an entity initially wants to join the cloud or when the broker's access control list is either referenced or modified by an administrator. As mentioned previously in Section 4.3, there are four security classifications for an ABELS cloud, and the access control module implements the necessary functions of the various security protocols. In addition to the list of IP address restrictions, there is a second list containing the IP addresses of those who are granted administrative privileges for the brokering system. This module is used in two situations:

- When a user entity joins the ABELS cloud for the first time, this module will be called by the broker's communication module to authenticate the entity's GLA, using its digital certificate.
- When a request is made for an administrative modification to a lookup service or group, the module must check its access control list to determine whether the requester is authorized for administrative privileges.

Digital certificates and the secure socket layer (SSL) protocol are used to guarantee secure communication within the brokering system. Once access is approved, the broker issues the entity's GLA an entry ticket that must be presented before any subsequent communication with ABELS components or entities. After receiving an entry ticket, the GLA can communicate directly with the lookup services, for example.

The *load balancing module* is responsible for distributing the registration load among lookup services, and also ensuring that lookup services are distributed evenly across the network. Load balancing is carried out automatically to maintain a robust system that is not bogged down by excessive load concentration. Manual data distribution by a human administrator is also allowed. Automatic load balancing occurs under two scenarios: too many groups are supported by a single lookup service, or too many lookup services are running on a single computer. Currently, a lookup service exceeding the group threshold will have some of its supported groups (and the serviced proxies of the services that support these groups) moved to a lighter-loaded lookup service. To accomplish these modifications, the load balancing module sends the required changes to the resource manager module, which actually moves the groups. In the future, we will consider additional load balancing factors, including the load of the Java Virtual Machine, the number of network connections to any one machine, and the hardware capabilities of the machine.

The *resource manager module* manages the cloud data, keeping track of the lookup services and the groups they support. Thus, the resource manager maintains a list of available resources in the cloud. When a consumer GLA submits a query, the resource manager uses this list to perform a first-level lookup on the query's groups, and it returns to the consumer GLA the lookup service(s) of interest.

The resource manager also performs certain administrative tasks. As appropriate, the module takes care of adding, removing, changing, and editing lookup services. Some of these changes will occur automatically at the request of other broker modules, while others will be performed at the request of the human administrator who oversees the cloud operations.

The actual modification of lookup services requires communication between the resource manager and the *meta lookup service*, which is a lookup service of lookup services. As shown in Figure 3, the meta lookup service provides the link between the broker modules and the various lookup services in the cloud. It is required to replace multicast communication with the safer unicast communication.

As discussed in Section 4.3, by default Jini uses multicast communication to implement the discovery protocol; in this case, there are frequent announcement messages broadcast to the network describing the location and function of the message's sender. By monitoring these broadcasts, the broker could determine the locations of the available lookup services. However, the decision to use unicast communication for security purposes means that no such announcements are made, yet the broker still must know the locations of all of the

lookup services. In fact, unicast means that the broker can communicate only with lookup services whose addresses are known. The solution is to create the meta lookup service at a known location once the broker has been initialized. All lookup services in the cloud must then register themselves with the meta lookup service. Once the lookup services are registered with the meta lookup service, the broker can use the meta lookup service to access the regular lookup services.

Finally, the *data recovery module* is called as soon as a lookup service becomes unavailable. For fault tolerance, at least two copies of each lookup service are maintained, and the duplicate copies are used to recover from a lookup service failure. The required recovery data is first transferred from the backup lookup service to the broker. Next, the resource manager is called to re-build the crashed lookup service while it concurrently brings the backup online as the primary lookup service. As the data is recovered and a backup lookup service rebuilt, the broker informs the consumer GLAs of the new lookup service. Alternatively, if available system memory is limited, data recovery can occur through the use of maintained log files and Jini lookup service *snapshots* (Li 2000). Snapshots are occasional recordings of the entire system state that are saved to disk. In the case of a system crash, the snapshots are reloaded by the system and the state of the system at the time of the last snapshot is restored. Because only a single set of lookup services is maintained in this method of data recovery, the cloud requires less memory space on the network. However, two main shortcomings exist to this method: the cloud is unavailable while the system reloads the snapshots, and any changes in the system between the last snapshot and the crash are permanently lost. With either method of data recovery, new service registrations and queries may be temporarily blocked to ensure data consistency, but all the services can still be read.

5.2 Join Process

The join process is the means by which an entity initially joins the cloud. It can be divided into the following steps:

- An entity's GLA sends a join request to the communication module of the broker.
- The communication module calls the access control module to perform authentication and authorization of the GLA, depending on the cloud's security level.
- The access control module returns the result to the communication module.
- If the registration is accepted, the broker's communication module returns an entry ticket to the entity's GLA. Otherwise, an error message is returned to the GLA.

Once a GLA has received its entry ticket, it is said to have joined the cloud. It is then able to participate as a cloud member by presenting its entry ticket in all subsequent cloud communications.

5.3. Lookup Process

Once a GLA has joined the cloud, it can register services (if a producer GLA) or submit queries (if a consumer GLA) after first presenting its entry ticket which verifies it is an approved cloud participant. From the perspective of the broker, the procedures for registering services and submitting queries look very similar. As discussed in Section 4.2, a producer GLA must know the group(s) it wishes to support and a consumer GLA must determine the group(s) it would like to search for potential matches. This determination occurs before the GLA contacts the broker. Once determined, the specified set of groups is sent to the broker in order to receive all lookup services supporting these groups. This process can be divided into the following steps:

- The entity's GLA requests from the broker's communication module, a list of lookup services that support its groups of interest.
- The communication module verifies the entry ticket supplied by the GLA with its request.
- The communication module calls the resource manager module, passing the desired group(s).
- The resource manager gives the list of groups to the meta lookup service to receive the desired lookup services' addresses and ports.
- The broker returns the IP addresses and ports to the entity's GLA.

With these addresses in hand, a producer GLA can register its service with the lookup services returned. A consumer GLA will then perform a first-level lookup with the lookup services to find producers registered in its particular group(s) of interest. It will then require a second-level lookup to determine the best service for its needs. This process, described in Section 4.2, will rank all the services returned to the GLA. Following the second-level lookup, the consumer GLA can connect to the highest-ranked service's GLA to resolve the query.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the design of the secure ABELS brokering system. The brokering system manages the resources in the cloud, storing and maintaining a dynamic set of descriptions and references to the entities currently participating. The brokering system matches data producers to consumers while restricting cloud users to those that are authorized. A flexible security framework has been designed so that the system can be adapted to multiple deployment scenarios. In conjunction with the GLA, the broker provides an effective, two-level security framework for the ABELS system.

Currently, we are continuing the implementation of the design presented in this paper. We are also in the process of developing a graphical user interface (GUI) to aid the cloud administrator in creating and managing the broker. Future work will include the implementation of ABELS-specific lookup services and the development of improved load balancing algorithms. We also plan to add mechanisms to gather and utilize various system statistics, and provide email notifications for user-specified events of interest.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation KDI Grant 9873138 and U.S. Army Corps of Engineers contract DACA42-01-P-0288.

REFERENCES

Curbera, F.; M. Duftler; R. Khalaf; W. Nagy; N. Mukhi; S. Weerawarana. 2002. "Unraveling the Web Services Web: an Introduction to SOAP, WSDL, and UDDI." *IEEE Internet Computing* 6, No.2 (Mar./Apr.), 86-93.

Dahmann, J.; F. Kuhl; and R. Weatherly. 1998. "Standards for Simulation: As Simple as Possible but Not Simpler, the High Level Architecture for Simulation." *Simulation* 71, No. 6 (Dec.), 378-387.

Kumar, A.; L. F. Wilson; T. B. Stephens; and J. T. Sucharitaves. 2002. "The ABELS Brokering System". In *Proceedings of the 35th Annual Simulation Symposium* (San Diego, CA, April 14-18). IEEE, Picataway, N.J., 63-71.

Kumaran, S. I. 2002. *Jini Technology: An Overview*. Prentice-Hall, Upper Saddle River, N.J.

Li, S. 2000. *Professional Jini.* Wrox Press Ltd, Birmingham, U.K.

Mills-Tettey, G. A. and L. F. Wilson. 2003a. "Security Issues in the ABELS System for Linking Distributed Simulations". *Proceedings of the 36th Annual Simulation Symposium* (Orlando, FL, Mar. 30 – Apr. 2). IEEE, Picataway, N.J., 135-144.

Mills-Tettey, G. A. and L. F. Wilson. 2003b. "A Security Framework for the Agent-Based Environment for Linking Simulations (ABELS)." *Simulation*, to appear.

Peteet, J. O.; J. P. Murphy; and L. F. Wilson. 2003. "Matchmaking in the ABELS System for Linking Distributed Simulations". *Proceedings of the 15th European Simulation Symposium* (Delft, The Netherlands, Oct. 26 – 29), SCS, San Diego, CA, to appear.

Wilson, L. F.; D. J. Burroughs; A. Kumar; and J. Sucharitaves. 2001. "A Framework for Linking Distributed Simulations Using Software Agents." *Proceedings of the IEEE* 89, no. 2 (Feb.), 186-200.

Wilson, L. F.; B. Xie; J. M. Kimpel; G. A. Mills-Tettey; and G. Johnston. 2002. "The Design of the Distributed ABELS Brokering System". *Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT)* (Fort Worth, TX, Oct. 11-13). IEEE, Picataway, N.J., 151-158.

AUTHOR BIOGRAPHIES

LINDA F. WILSON is an associate professor at Dartmouth's Thayer School of Engineering. She received her BS degree in mathematics from Duke University in 1988 and her MSE and PhD degrees in electrical and computer engineering from the University of Texas at Austin in 1990 and 1994, respectively. Her email address is `Linda.F.Wilson@dartmouth.edu` and her web page can be found at `http://thayer.dartmouth.edu/~lwilson`.

W. RILEY LOCHRIDGE is a master's student at Dartmouth's Thayer School of Engineering. He received his AB degree in history and engineering modified with computer science from Dartmouth College in 2002. His email address is `Riley.Lochridge@dartmouth.edu`.

G. AYORKOR MILLS-TETTEY is currently working in her native country of Ghana. She received her AB degree in computer science in 2001 and her BE and MS degrees in engineering in 2003, all from Dartmouth College. Her email address is `Ayorkor.Mills-Tettey@alum.dartmouth.org`.