

UTILIZING SECURE MECHANISMS FOR REMOTE ACCESS TO SCIENTIFIC INSTRUMENTS

Mohamed Ahmed Mohamed.Ahmed@nrc-cnrc.gc.ca
Andre Charbonneau
Ratilal Haria
Roger Impey
Gabriel Mateescu
National Research Council, Canada

Darcy Quesnel darcy.quesnel@canarie.ca
CANARIE, Canada

Abstract: Grid computing supports sharing of distributed resources across boundaries and authorization domains, and can help scientists by providing secure, transparent and easy access to computing resources. We have designed and developed grid-based services for accessing, visualizing, and reliably manipulating remote spectrometry instruments. We harness grid-computing technologies to allow scientists to concentrate on performing experiments and data analysis, without having to become experts in the software technologies enabling these activities. One major design issue was the security aspect of our system. We will present our solution to this problem given the ease of use required and remote access by the scientists. This solution provides single-sign on, message integrity, confidentiality and transparent access to remote graphical scientific software and remote file transfer.

Keywords: Grid computing, Mutual authentication, Digital certificates, Single sign-on, Web services.

1. INTRODUCTION

High performance and grid computing [5] are making it possible for scientists to solve more complex problems and to securely carry out remote collaboration and resource sharing. Advances in processing power, network

bandwidth, and parallel and distributed computing enable scientists to apply innovative ideas and approaches to their work. Recent standards include security, job submission, resource management, and information services. Some of the available middleware solutions include the Globus Toolkit [1] (a set of basic services and APIs for computational and data grids), Avaki [4] (a solution for data grid technology), and Unicore [8] (a system for uniform access to remote resources). However, these tools usually require a good understanding of the underlining protocols, standards, commands, and APIs. This does not promote their adoption by end users that are not computer professionals, such as biologists, chemists, and physicists. End users want easy to use, reliable, and intuitive tools.

Currently, scientists do not have an easy way to access Nuclear Magnetic Resonance (NMR) resources, either within their institution or remotely. The scientists require secure and easy to use software for accessing existing NMR software for data acquisition and analysis. Their requirements include remote utilization, security, one-time authentication (single sign-on), secure file transfer, scheduling, heterogeneous clients, graphic interface, and visualization. Existing NMR applications are not specifically designed to be exposed via web services [2] or for resource sharing. In this paper, we will present the design and implementation of a secure system that provides remote access service to those NMR instruments or most of similar scientific instruments in that matter. The system streamlines access to multiple resources by providing single sign-on, resource visualization, and supporting cross-platform clients. The user can select from available resources in order to monitor, manage, schedule, and ultimately use them. We make use of open standards such as the SSH protocol, Grid Security Infrastructure (GSI), X11, Java [5] and Simple Object Access Protocol (SOAP).

The remainder of this paper is organized as follows. In section 2, we will discuss the system requirements and design. We present the implementation details in section 3. In section 4, we will present another secure alternative using grid security infrastructure mechanism and conclude and mention our future direction in section 5.

2. SYSTEM DESIGN

Budget and location considerations have made the use of remote resources (often encapsulated into services) an increasingly attractive alternative to in-house resource provisioning. From collaborative research in physics to magnetic resonance imagery, the paradigm of scientific applications is shifting toward the sharing of resources across multiple

organizations. This paradigm shift is called Grid computing and involves the creation and use of hosted services. Users of a hosted service may belong to other organizations than the providers of the service. Security and web services technologies support this paradigm shift. A security infrastructure helps define reliable identity authentication mechanisms for users belonging to different organizations. Web services technologies, which employ XML-based standards such as SOAP and the Web Services Description Language (WSDL), allow applications developed in different languages and executing on different platforms to discover services available over the Internet and to invoke those services (using SOAP) via interfaces that are specified in a portable manner (using WSDL).

Our contribution is the definition and implementation of a software architecture that combines several technologies in order to meet functional requirements, as well as usability and extensibility requirements. The secure system provides several services:

1. Resource allocation management, such as defining NMR projects, NMR instrument allocations, and user and projects quotas.
2. Experiment scheduling.
3. Experiment launching and real-time monitoring.
4. File transfer of experiment data.

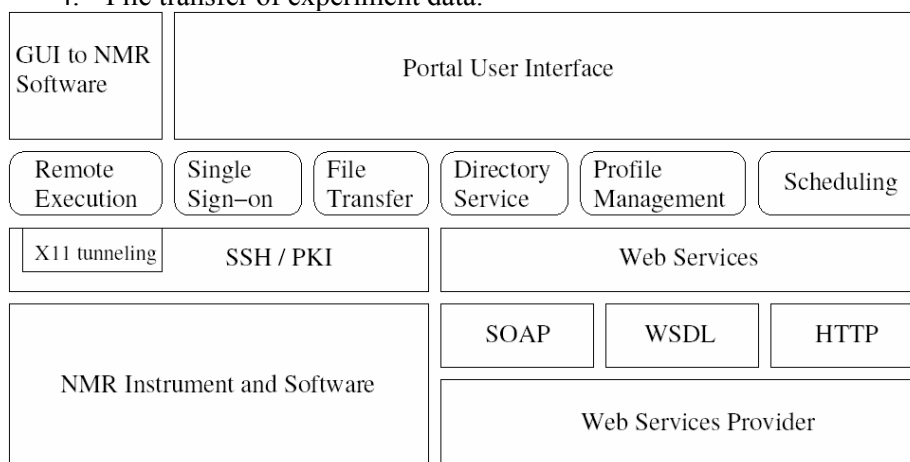


Figure 2.1. System Architecture.

The design follows a three-tier approach:

1. The first tier is the client providing the user interface for connecting to and accessing the remote resources and managing user preferences such as communication settings that define the data compression and encryption.

2. The second tier, referred to as the *NMRConnect server*, contains the control logic and the data source. The logic provides functionality such as user authentication and authorization, work scheduling, and project management, while the data source provides persistence. The data source is implemented as a relational database and can be extended to support automatically updated dynamic information (for example the status of a machine that controls an instrument) using as information source the Grid Information Services. This tier also includes the NMR Gateway, which provides SSH connectivity to all the NMR systems.
3. The third tier is the executor of the NMR software. We will refer to the third tier as the *NMR system*.

The architecture of the system is illustrated in figure 2.1, where we distinguish several components. The first is the user interface, which consists of an HTML client (as shown in figure 2.2) and a Java client program (as shown in figure 2.3). The former is for experiment scheduling and the latter for launching the execution of the remote application, which controls the instruments. They expose the user interface of the controlling software, which communicates with the remote application via the X11 protocol, tunnelled through SSH (explained further in Section 3). The second component is the set of functional modules that provide the desired services, including single sign-on and profile management (for user authentication and authorization), directory service (for locating the machines and instruments), scheduling (for reserving time for the experiments), remote execution of the control software and transfer of the files generated by the experiments). The third component is the protocol suite employed, which include SSH (for authentication and encryption), HTTP (for transport), and SOAP (for communication between the Java client and the remote service). The fourth component is the NMR Gateway which provides secure, authenticated connectivity and data transfer to all the NMR systems through the use of an SSH forwarding agent. The final component is the set of web services that implement policy management, resource scheduling, and user management.

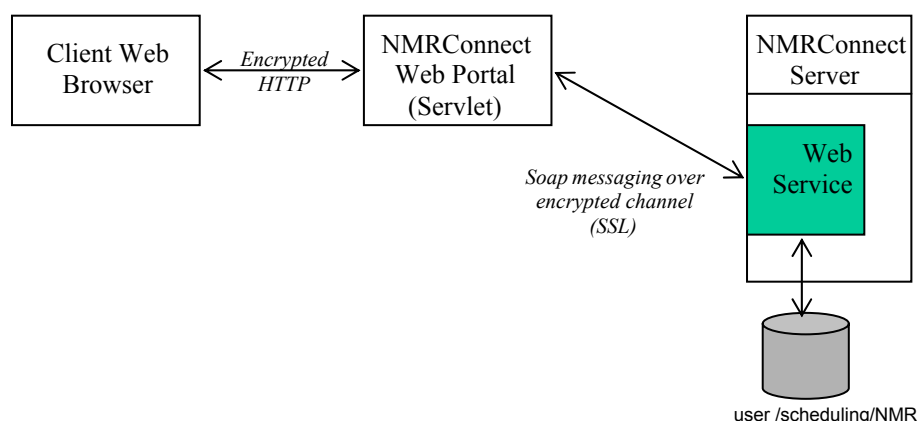


Figure 2.2. Web Client Diagram

While different machines can provide these components, for simplicity we will assume that they are provided by a single machine. The second tier is structured following a service-based architecture, which uses the Model-View-Controller pattern as follows:

1. The *model* consists of a set of web services for querying and updating information such as the available NMR systems, user profiles and resource allocation. The state is stored in a relational database, which is accessed by the web services via JDBC. A part of the NMRConnect WSDL interface definition is depicted in figure 2.4.
2. The *view* consists of two types of components:
 - a) Java Server Pages (JSPs) and servlets that generate HTML content to be rendered in a web browser and a
 - b) Java client application that scientists use to access the remote NMR system.
3. The *controller* includes a session management servlet, which receives the client requests, decodes the requests and dispatches them to the appropriate servlet that translates the requests into operations on the model.

The web services are invoked by the clients via client side stubs that encapsulate the web service invocation and hide from the client code details such as the location and the binding of the web service. The value object design pattern is used to reduce the number of web service invocations: all attributes related to an entity (such as scientific instrument) are gathered into a single class, and a web service that returns arrays of value objects is

invoked. Currently, single sign-on is implemented by using SSH clients in the GUI client and NMRConnect server tiers, SSH servers in the NMRConnect server and NMR system tiers, and an SSH agent in the GUI client tier (see also Section 3.2). Note that the single-sign-on concept refers to accessing multiple NMR systems, while access to the web portal is controlled as described in Section 3.1.

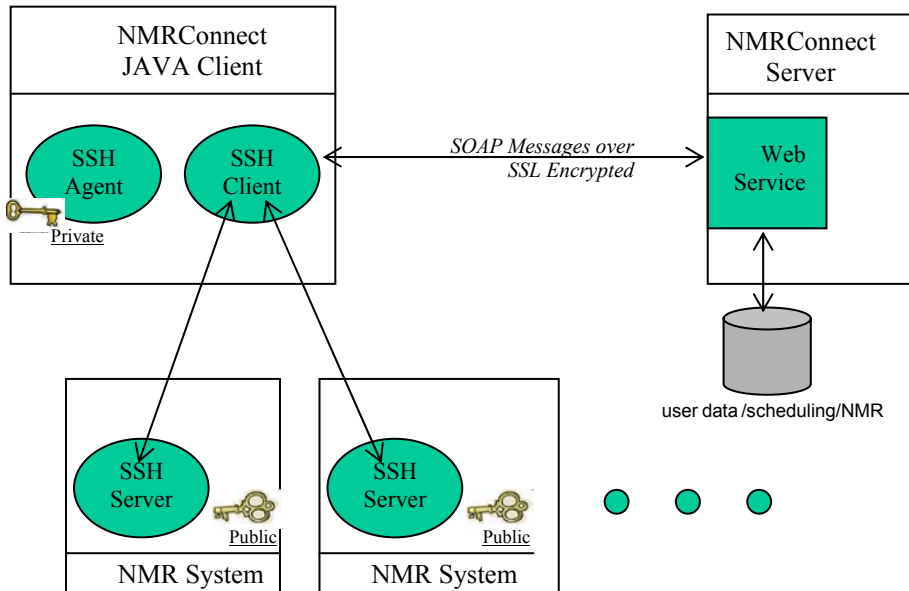


Figure 2.3 Java GUI Client Diagram.

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:NMRConnectServer"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="urn:NMRConnectServer" xmlns:intf="urn:NMRConnectServer"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:message name="getNMRSystemsWithAccountForUserResponse">
    <wsdl:part name="getNMRSystemsWithAccountForUserReturn"
type="impl:ArrayOf_xsd_string"/>
  </wsdl:message>

  <wsdl:message name="getReservationsForNMRSystemResponse">
    <wsdl:part name="getReservationsForNMRSystemReturn"
type="impl:ArrayOf_xsd_string"/>
  </wsdl:message>

  <wsdl:message name="getInactiveAccountsForNmrResponse">
    <wsdl:part name="getInactiveAccountsForNmrReturn"
type="impl:ArrayOf_xsd_string"/>
  </wsdl:message>

  <wsdl:message name="removePublicKeyRequest">
    <wsdl:part name="in0" type="xsd:string"/>
    <wsdl:part name="in1" type="xsd:string"/>
  </wsdl:message>

  <wsdl:operation name="removePublicKey">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="removePublicKeyRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:NMRConnectServer"
use="encoded"/>
    </wsdl:input>

  <wsdl:service name="NMRConnectServerService">
    <wsdl:port binding="impl:NMRConnectServerSoapBinding"
name="NMRConnectServer">
      <wsdlsoap:address
location="https://arsenic.sao.nrc.ca:8443/axis/services/NMRConnectServer"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```

Figure 2.4. Part of NMRConnect Server Web Services Interface Definition.

3. IMPLEMENTATION AND DISCUSSION

The implementation follows a three-tier model as described in the previous section. Java has been chosen as the implementation language because of portability requirements and the abundance of web services tools.

3.1. Tier Implementation Details

The client is a Java Swing-based graphical interface that allows the user, typically a scientist who wants to analyze a sample already put in the NMR machine, to select from available NMR systems and launch the data acquisition and analysis software for that system. The user does not have to worry about terminal sessions, setting up an X server on the desktop or X11 tunnelling. Instead, the user simply selects the NMR system they want to use and the remotely running NMR software's graphical user interface is displayed on the user's desktop. In addition, the Java GUI client enables users to create new SSH key pairs as needed. When a new key pair has been generated, the public part of the key is automatically sent to the NMRConnect server and every NMR administrator is notified of the new key. NMR administrators then install the public key onto the systems that the user has an account on. Before installing the key, the administrator might do further user authentication, by doing a phone call to the user for example. Single sign-on objective is achieved using an SSH agent. The SSH agent is automatically started when the NMRConnect Java client GUI is launched. The user is then prompted to enter a password (it's the password that was used to encrypt their private key). The agent will then take care of sending the user credentials to every SSH connection that is using that SSH key pair. When the Java application exits, the SSH agent is automatically shut down. The choice of SSH is motivated by its protection against threats such as eavesdropping and masquerading.

In addition to the Java client software, the first tier is also composed of an interface to the scheduling and administrative functions presented via a web browser. The web pages composing this interface are HTML forms that are built dynamically at runtime by Java servlets, which rely on web services provided by the NMRConnect server. The NMRConnect server is implemented in Java as a web service hosted by a servlet engine. We use the open source Apache Tomcat as a servlet container and the Apache Axis as the SOAP implementation.

A relational database is used to support authentication and authorization to the NMRConnect Web Portal. The database, containing user accounting and authorization information such as user roles as well as all

NMR systems and the settings needed to connect to them, is accessed via JDBC (which makes it easy to support various data sources at deployment time). An example of the user roles table and their possible values is illustrated in table 3.1. Each user has an account on the NMRConnect server. The role mapping is stored in a database (MySQL in this case) on the NMRConnect server and Tomcat is configured to use the database to map users to their roles. User passwords are stored in the NMRConnect database in a 'one-way' encrypted form (using Secure Hash Algorithm, SHA). This assures that original passwords cannot be derived from the database in the case that it is accessed by a person without proper authorization. NMRConnect web services are implemented in Java and hosted by Apache Axis. The Java GUI client makes calls directly to the Axis servlet. The web portal (which contains the scheduling and various administrative functions) is rendered by a layer of servlets, which will make NMRConnect web service calls to the Axis servlet as needed. User authentication is done both at the web portal servlet level and at the Axis web services level. Both levels are using the NMRConnect database to authenticate users using password and user role tables. Jakarta Tomcat implements the authentication engine and is configured to use the NMRConnect database as a source of authentication data. When a user accesses the NMRConnect web portal, they will authenticate (using basic web portal authentication methods) to the NMRConnect web portal servlet. When that servlet needs to make a web service call to the NMRConnect web service, the user credentials are automatically extracted (by the NMRConnect web portal servlet) from the HTTP request object and passed along with the web service call. The web service will then authenticate the user again, but this is transparent to the user since its done automatically. User authorization is implemented in the NMRConnect web services themselves.

As shown in table 3.1, the NMRConnect database contains a table that maps user roles to web service methods. This table specifies which web service is allowed to be executed by this user. In this example, the users belonging to the `nmr_nmadmin` role can call the `addNmr` web service method to add a new NMR system to the database. Only project administrators can add a project and NMR users are allowed to add a new public key but only to themselves, not to other users (hence the '1' in the last column). Implementing the authorization in this way allows us to use any authorization logic that might be necessary. The NMRConnect database, which is accessed by the NMRConnect web service methods, is password protected and is always accessed under the same username. The password associated with that username is stored on the NMRConnect server in a directory that is only readable by the owner of the NMRConnect web service.

This assures that the database password is not hard coded in the Java code of the web services. User public keys are also stored in the NMRConnect database. That way, when a new NMR account is needed for a user, the existing public keys for that user are extracted from the database and installed in this new account. User private keys are never sent to the NMRConnect server, therefore it is the user's responsibility to manage and secure their private keys. When client requests information about the NMR systems, the server will filter the information appropriately so that only the NMR systems that a user is authorized to use are presented.

The NMR system hosts the software that manages the NMR instrument directly. The NMR software handles exclusive access to the resource, and presents the user with data acquisition and analysis tools. For the NMR machines hosted at the National Research Council of Canada, the NMR software is a graphical application running on a Unix system and using X11 protocol, thereby allowing any client that has an X server to remotely interact with it. To enforce proper authorization and resource allocation, when the NMR software on the NMR system is started, it communicates with the NMRConnect server to determine if the user has reserved a time-slot for that NMR for that particular point in time. If the user has not reserved or is not authorized to use that NMR system (for example the user exceeded the allocated quota), the software on the NMRConnect server will refuse to execute and an error message will be displayed to the user.

Table 3.1. Table of user roles and authorizations.

```
mysql> select * from user_authorization order by method;
```

<i>role_name</i>	<i>method</i>	<i>user_restriction</i>
nmr_nmradmin	addNmr	0
nmr_projadmin	addProject	0
nmr_user	addPublicKey	1

OpenSSH, the open source implementation of the SSH protocol, is used to securely connect from the client through the NMR Gateway (which is part of the NMRConnect server) and to the NMR system. The SSH functionality is accessed by executing external commands (such as ssh, ssh-keygen, and ssh-agent). The X11 protocol spoken by the application on the NMR system to the client (with an X server which is automatically started by the client software when required) is tunnelled through this SSH connection. Compression on the SSH connection is turned on and offers a noticeable increase in performance. On our systems, no significant difference was perceived between different encryption ciphers. For users that do not need encryption, our model also supports direct connection between the X server

(running on the user's desktop system) and the X client (running on the NMR system host) instead of tunnelling through SSH. Figure 3.1 shows the user interface that the client uses to configure the session's settings.

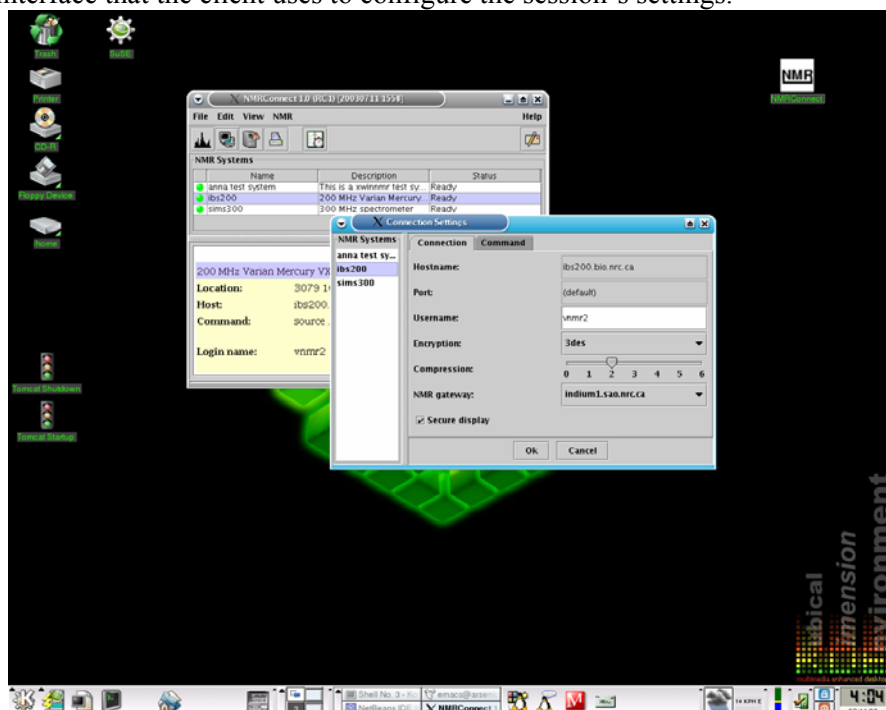


Figure 3.1. The User Configures the Session Settings.

3.2. Connecting to The Remote Resource

As discussed earlier, using the Java client program, researchers and scientists can easily locate available NMR resources and run the NMR data acquisition software remotely. Although starting the NMR software seems like a single step process to the end-user, the process involves multiple steps, which are transparent to them. Figure 3.2 illustrates the basic steps performed by the client program; from the time it is started up to the time the NMR data acquisition and analysis software is remotely displayed on the user's desktop. Upon client program startup, an SSH agent is launched by the client program in the background. The agent forwarding feature of SSH agent is used in this scenario. What happens is that when the SSH client on the NMRConnect server receives a request for authentication, it will check if the forwarded SSH agent is able to handle the authentication. That way, the private key that is stored on the client side will be used for authentication purposes, therefore

it does not need to be present on the NMRConnect server, only the public key is needed on the NMRConnect server in order to establish the first SSH connection, which is between the client and the NMRConnect server and then between the NMRConnect server and the remote NMR system using an SSH forwarding agent.

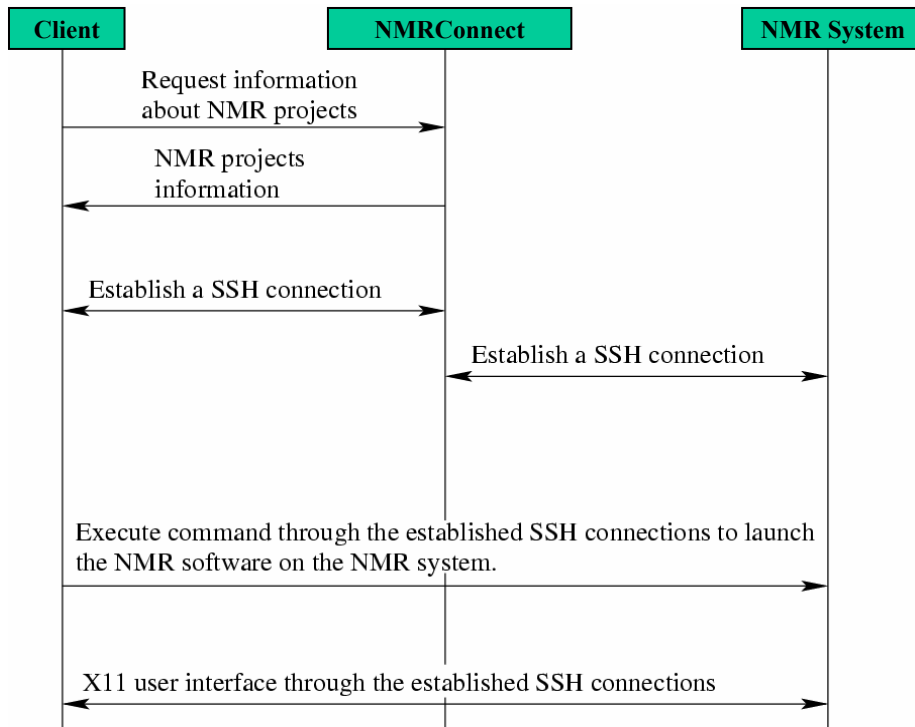


Figure 3.2. Interaction Sequence Diagram.

The SSH agent maintains user credentials, so that the user needs to authenticate only once with this agent. Subsequent user authentication for SSH connections will be handled by this agent. Resource discovery is the next step of the process. Upon startup, the client application will automatically query the server for a list of available NMR systems. After consulting its database, the server will return the information to the client program, which will then display a list of available NMR systems to the user. Figure 3.3 shows a snapshot of the Java client program's graphical user interface. After being presented with the information shown in figure 3.3, the user will select an NMR system from the list of systems to use for an experiment, then click on a button to launch the NMR software.

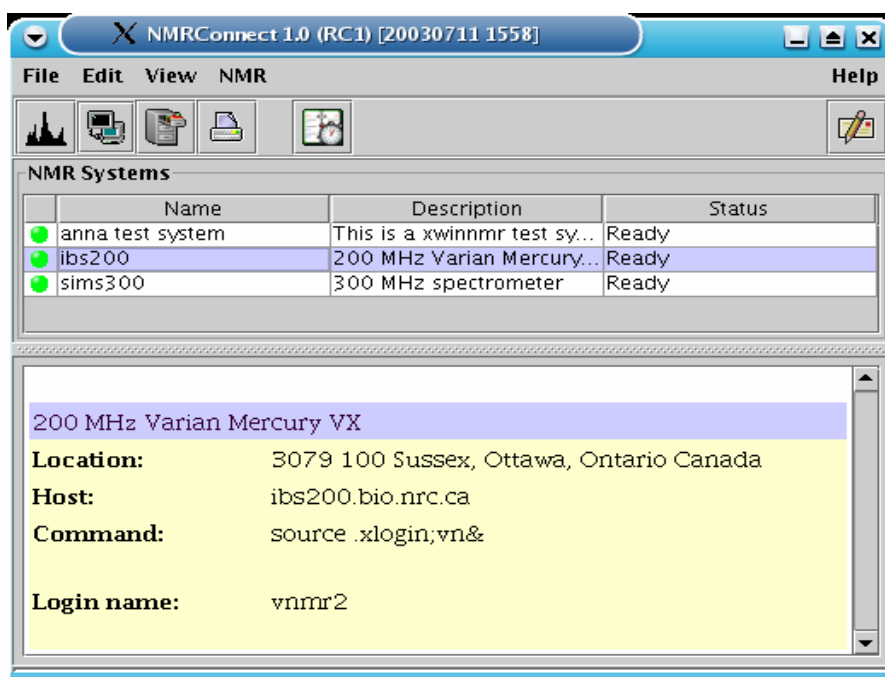


Figure 3.3. The Graphical User Interface of the Java Client Program.

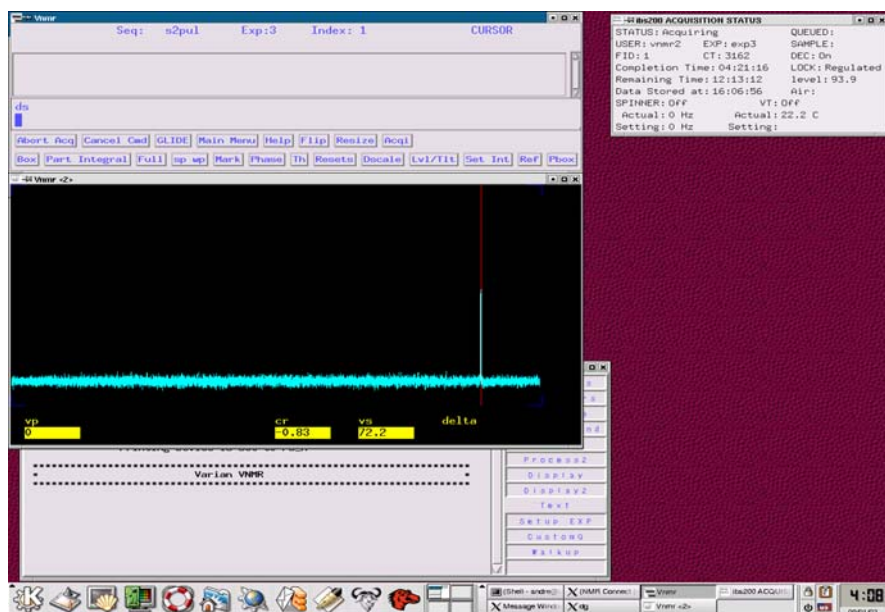


Figure 3.4. The NMR software user interface on the client desktop.

In the next step, the client program establishes an SSH connection between the user's system and the NMRConnect server. Then, the SSH agent handles the passing of user credentials between the client program and the server. Once the first SSH connection is established between the client program and the NMRConnect server, a second SSH connection is immediately created between the NMRConnect server and the NMR system that was selected by the user. This new connection is initiated from the client program by using the first connection to execute an SSH command on the NMRConnect server. These two consecutive SSH connections create a communication channel between the client program and the NMR system, passing through the NMRConnect server. From this point on, the client program will keep this connection open until either it is explicitly closed by the user or the client program is terminated. In addition, the client program also supports direct SSH connections between the clients and NMR systems, bypassing the NMRConnect server. Direct connections can be used when the NMR systems do not have security restrictions on incoming connections. The final step involves starting the software on the NMR system. To achieve this, the client program sends, through the communication channel, the command specifying path name of the NMR software that will launch the software on the NMR system. Using SSH X11 tunnelling, the application's graphical user interface will be forwarded to, and displayed on, the client desktop through the communication channel without the user intervention. This is shown in figure 3.4. Note that, the NMR software will only be run after a launcher program first contacts the user profile and scheduling services of the server to determine if the user is allowed to run the software. Transfer of experiment results between the NMR systems and the user's local system is achieved using a secure copy via SSH. This assures that the data being transfers is encrypted and securely copied in both directions.

4. AUTHENTICATION USING GRID SECURITY INFRASTRUCTURE

In this section two common scenarios will be presented to show how strategic hacking could be used to electronically terrorize important governmental and commercial entities over the internet. The first scenario is concerned with terrorizing governmental agency on the internet by taking over its network. The second scenario will attack the business of a commercial company by a simple but effective attack that is denial of service.

An alternative way of implementing secure, remote access is to use the Grid Security Infrastructure (GSI) [6] instead of regular SSH and SSH

agent. GSI is a set of protocols, tools, and software libraries that are used in the Globus Toolkit to allow users and software applications to securely access services and resources. It is based on public key infrastructure (PKI), extending SSL/TLS authentication, message protection, and integrity functionalities to support single sign-on and delegation features. In GSI, each user and resource has a set of credentials that are used to prove their identities on the grid. This set consists of an X509-based certificate and a private key. The certificate includes:

- ◆ Subject name,
- ◆ Public key belongs to that name,
- ◆ Subject name of certificate authority (CA) that has signed the certificate, and
- ◆ Digital signature of the named CA

The private key should never be transferred over the wire. The idea of GSI certificate is to bind a certain user or service name to certain public key. This binding is authorized at a trusted third party. This party is usually called a certificate authority (CA). This is a main advantage of GSI over regular SSH to ensure that this user is the one who they claim to be. A CA is an entity on the grid that exists to sign user certificate requests. The user can then sign a proxy certificate on for use in job submission, for example.

A related project has implemented the Grid Canada Certificate Authority, which we use here. This CA is responsible for providing the users of the NMR project with a trusted certificate that can be used for any form of authentication either local or remote. Figure 4.1 shows the steps of applying for a Grid Canada user or host certificate (after reading and accepting the Grid Canada Certificate Policy and Certification Practice Statement). As seen in the figure, the user first downloads the public key of the Grid Canada CA. Then the user generates a private key and certificate request using a GSI client command (`grid-cert-request`). They keep this private key locally and sign the request containing their public key and any needed information using their private key (the CA confirms that the signature is by the private key associated with the certificate request's public key). The user then emails the resulting certificate request to the CA from an email domain name that corresponds to the request. After verification, the CA signs the certificate request and sends it back to the user. This certificate is valid for one year, by default. The user can subsequently send a renewal request when the certificate is close to expiring.

After the user receives their certificate from the CA, they can use it for various activities on the grid. This could range from GRAM (Globus Resource Allocation Manager) job submission to GridFTP transfers. Figure 4.2 presents the main steps performed for GSI mutual authentication, where the client verifies the server's credentials and vice versa. This will succeed if they both have valid and trusted certificates, where the CA's that issued the respective certificates are explicitly trusted by one another.

As shown in figure 4.2, mutual authentication starts when the user on host A generates a proxy certificate. This proxy certificate is generated by signing the user's certificate with the private key (which is protected locally by a pass phrase). This proxy certificate, and not the original user certificate, will be used for all further requests by the grid user tools to all grid resource, thus fulfilling the single sign-on requirement. The proxy certificate has a time limit that guarantees that the proxy cannot be used after certain amount of time even if it is compromised. The proxy can then be delegated to a grid service so that it can work on behalf of the user, authenticating with other grid services with the user's delegated credentials. To proceed with the authentication, client tools on host A sends the public part of the proxy credentials to the server on host B. Using the Grid Canada CA public key, host B can confirm that the user on host A is trusted and certified by this third party CA. To confirm that this user submitted this request, the service on host B issues a challenge to host A. Host B generates a random number and send it to host A asking to be encrypted by the user's private key. Host A encrypts the random message and sends it back to host B. Host B decrypts the encrypted random message using the user's public key it grabbed from the proxy certificate. If the result match the original random message, then host B confirms that the owner of this proxy certificate sends this request. To continue the mutual authentication process, the same steps take place but in reverse order and this time using host B certificate.

However, mutual authentication, if successful, doesn't guarantee access to the resource for the user. The user should be granted authorization for this resource by the resource owner, who maps the user's unique subject name (or distinguished name that resides in their certificate) to a user name local to the resource itself. (This is usually done using a file called gridmap-file in a Globus deployment.)

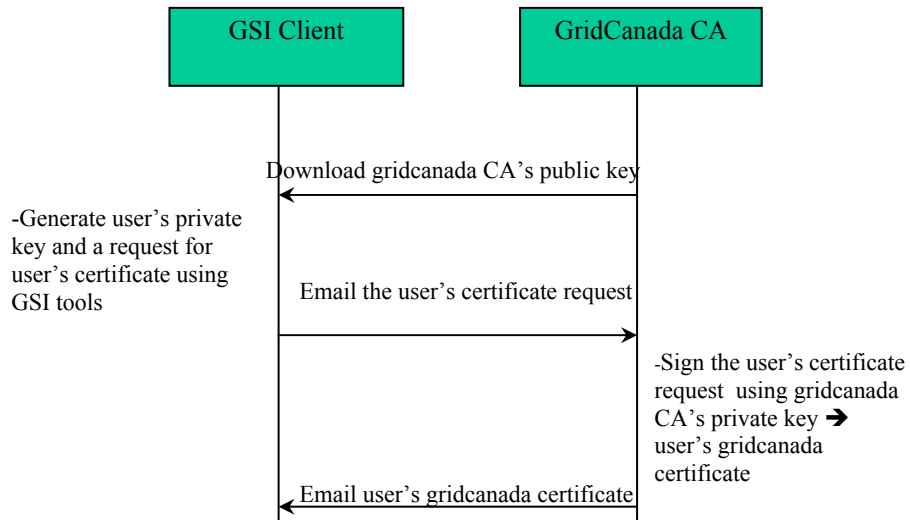


Figure 4.1. Steps of a user application for a Grid Canada certificate.

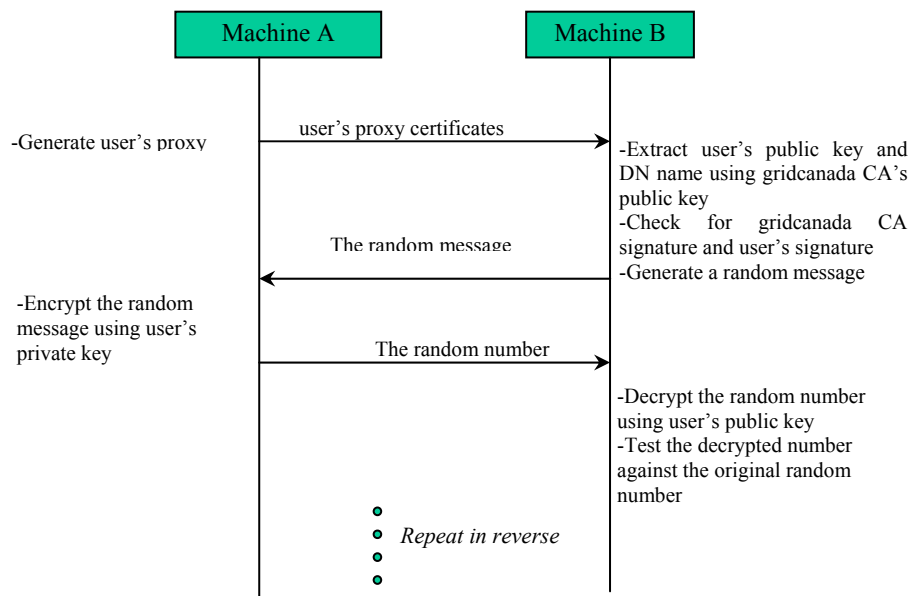


Figure 4.2 Steps of GSI mutual authentication.

A major step in being able to use the user certificate remotely, when user is not at the host with their private key, is that our system should include a MyProxy [8] server. A major requirement in the grid is that the user keeps their private key on a secure host, not necessarily on their laptop or PDA, while they are roaming or travelling. The MyProxy server is a credential repository that contains the user proxy certificates that can be retrieved by a grid web portal to perform the GSI authentication and then job submission, file transfer, or resource and service discovery on behalf of the user. Figure 4.3 illustrates the steps needed to support the grid authentication while the user is remote or roaming. First, the user makes a secure remote connection from their remote client machine to their own host using SSH Java signed applet client. They run the `myproxy-init` client command to generate proxy certificate that is valid for certain period of time and also provides a passphrase to retrieve this proxy. Then the MyProxy client process mutually authenticates against the Grid Canada MyProxy server using GSI protocol to store a limited-delegation proxy. The user provides the username/password that grants access to the MyProxy server to the grid portal. Then the portal will mutually authenticate to the MyProxy server to retrieve the user proxy, which it will then use to authenticate on behalf of the user against any authorized grid resource. This way, the user can use their credentials without compromising their certificate or private key while they are roaming.

Thus, the advantages of using GSI certificate-based security mechanism in future over regular SSH mechanisms, even though both support single sign-on feature, are: GSI uses a time-limited proxy to reduce the effect of theft or hacking and man-in-the-middle attacks; in addition, GSI provides traceable links to the trusted third party CA. GSI, by default, implements mutual authentication between the entities as well as providing limited or full delegation capabilities. Also, GSI implements authorization through mapping certificate subject names to local host user accounts with local policy associated with it. However, we see one of the problems with GSI is the headache of maintaining and using the certificates for multiple resources in different virtual organizations when the user expects to access these grid resources from more than one point of entry and use certificates from multiple certificate authorities. We can use a GSI-supported MyProxy repository as discussed earlier with time-limited proxies to help solve this problem. Another disadvantage is that GSI-enabled SSH clients need to be supplied for each remote NMR system.

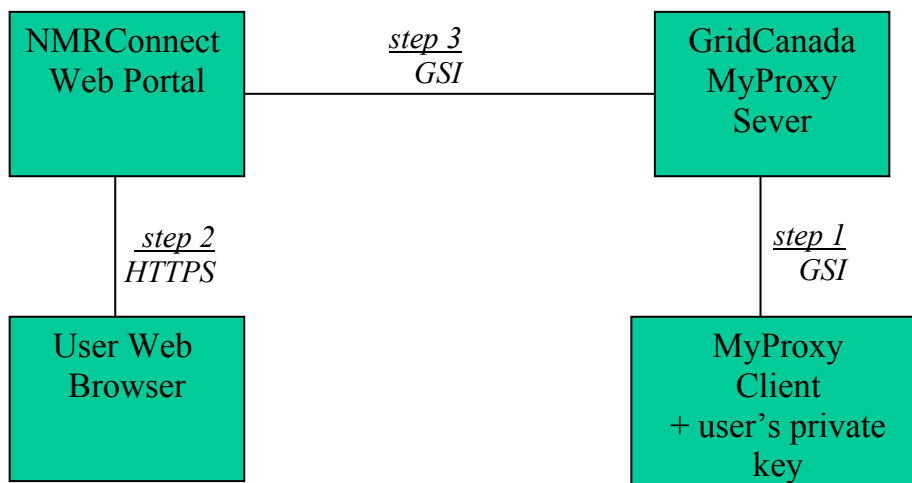


Figure 4.3. Authentication using Grid Canada MyProxy Server

5. CONCLUSIONS

In this paper, the system for accessing NMR instruments and software remotely has been introduced. This approach is superior to previous NMR system in several respects, including single sign-on, ease-of-use, seamless data transfer and flexibility. Our system incorporates some web services components that are hosted by Apache Axis and Jakarta Tomcat servers. We presented two major secure alternatives to accessing remote scientific instruments. We discussed the detailed implementation issues of each solution in the context of our project objectives. Consideration will be given in near future to incorporating emerging technologies and standards provided by Open Grid Services Architecture (OGSA), message-level security, XML digital signature, and XML encryption into our software design and implementation.

6. REFERENCES

- [1] Foster, I. and Kesselman, C. (Summer 1997), "Globus: A metacomputing infrastructure toolkit", The International Journal of Supercomputer Applications and High Performance Computing, 11(2), pp. 115–128.
- [2] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (June 2002), "Grid services for distributed system integration", Computer, 35(6), pp. 37–46.

- [3] Foster, I., Kesselman, C., and Tuecke, S. (Fall 2001), "The anatomy of the Grid: enabling scalable virtual organizations", International Journal of High Performance Computing Applications, 15(3), pp. 200–222.
- [4] Grimshaw, A., Wulf, W., and the Legion team (Jan 1997), "The legion vision of a worldwide virtual computer", Communications of the ACM, 40(1), pp. 39–45.
- [5] Joy, B., Steele, G., Gosling, J., and Bracha, G. (2000), "The Java language specification", Addison-Wesley, second edition.
- [6] Grid Security Infrastructure, <http://www.globus.org/security/>.
- [7] Novotny, J., Tuecke, S. and Welch, V. (August 2001), "An Online Credential Repository for the Grid: MyProxy", 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco CA.
- [8] The UNICORE project, <http://www.unicore.org/>