Scheduling Services and Security Ticket Token Services in iLab Interactive Services

by

Tingting Mao

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

Author .....
Department of Civil and Environmental Engineering
August 10, 2007

Certified by .....
Steven R. Lerman
Class of 1922 Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by .....
Daniele Veneziano
Chairman, Department Committee on Graduate Students

Scheduling Services and Security Ticket Token Services in iLab Interactive Services

by

Tingting Mao

Submitted to the Department of Civil and Environmental Engineering
on August 10, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science in Civil and Environmental Engineering

Abstract

The iLab architecture allows students to execute laboratory experiments remotely through internet. It supports three different kinds of experiments: batched, interactive and sensor-based. The iLab Interactive Experiments architecture includes the following servers and services: the Interactive Service Broker (ISB), the Experiment Storage Service (ESS) and the Lab Server (LS). In addition, students execute interactive experiments by running a Lab Client (LC). In order to support interactive experiments which require scheduled access, the iLab interactive architecture envisions scheduling servers and services which enable students from different campuses to reserve time periods to execute experiments. Since the user side and lab side require different scheduling functionalities, a user-side scheduling server (USS) and a lab-side scheduling server (LSS) are introduced in the iLab Interactive Services to manage reservations. In the first part of this thesis, the philosophy of the scheduling services design and the implementation will be illustrated in detail. In dealing the security issues in the iLab interactive architecture, the complexity of the higher level authentication between iLab processes increases when one considers collaboration between domains. In second part of this thesis, I present a Security Token Service (STS) scheme for using WS-Security to optimize the cross-domain authentication in the iLab interactive architecture. The scheme uses the brokered authentication with a security token issued by the STS. The STS is trusted by the web applications and web services in the iLab interactive architecture to provide interoperable security tokens. A security token is used to convey the credential information and the proof of a relationship with the broker, which can be used by the service to verify the token. A comparison between the STS scheme and the current General Ticket scheme is summarized.

Thesis Supervisor: Steven R. Lerman
Title: Class of 1922 Professor of Civil and Environmental Engineering

Acknowledgments

The author would like to thank the myriad persons who provided support and encouragement throughout the pursuit of this project. In particular: Professor Steven R. Lerman for his insightful advise and comments; Jud Harward, the principal research scientist at CECI, for providing invaluable advice, as well as excellent instruction through out all the stages of this project; Philip Bailey, Kirky DeLong,Imad Jabbour and Rabih Zbib for being wonderful teamates and providing invaluable support and help; Chris Felknor for being a technical "sounding board" and good friend.

No acknowledgement could be wholesome without extending my utmost gratitude to my parents and sister. You love and encouragement remain the excess of a impulse behind all that I am and all that I do.

Contents

List of Figures

Chapter 1

Introduction

1.1 iLab project and interactive architecture

The MIT iLab Project is developing a distributed software toolkit and service infrastructure to support internet accessible laboratories and promote their sharing among schools and universities. The project starts with the assumption that the faculty teaching with online labs and the researchers that provide those labs are acting in two roles with different goals and concerns. The iLab architecture focuses on fast platform-independent lab development, scalable access for students, and efficient management for lab providers while preserving the autonomy of the faculty actually teaching the students. In the 28 months of iLab's operation using a standardized software architecture, MIT experiment servers have executed approximately 38000 experiments in behalf of over 2600 MIT students and authorized guests as well as over 400 anonymous Internet users. The ultimate goal of the project is to establish an economy of shareable labs to enhance science and engineering education. In iLab project, three categories of experiments are envisioned to be supported by the iLab architecture: batched, interactive and sensor-based.

1.1.1 The Batched Experiment

In a batched experiment, the student specifies all parameters that govern the execution of the experiment before the experiment starts. The lab session consists of submitting an experiment protocol, executing the experiment, and then retrieving and analyzing the results. Typically, batched experiments run quickly so that scheduling is rarely necessary. The MIT Microelectronics WebLab (weblab.mit.edu) provides an excellent example.

1.1.2 Interactive Experiment

In an interactive experiment, the student typically sets a series of parameters, initiates the experiment, and then monitors the experiment's course, changing control parameters as necessary. Conceptually, an interactive experiment can be thought of as a sequence of alternating control and monitoring intervals. In general, the control intervals have many of the characteristics of a batched experiment, and the monitoring intervals resemble sensor experiments. The record of an experiment session typically includes both time-stamped control and sensor data as well as other forms of documentation that may include images or video. The Internet accessible heat exchanger at MIT (heatex.mit.edu) provides a good example of this type of experiment.

1.1.3 The Sensor Experiment

In a sensor experiment, the student usually can not specify any parameters although she may be able to select the particular sensor data that she wishes to receive. Running the experiment consists of subscribing to real time sensor data, usually presented in a graphical user interface such as a virtual strip chart. The system may provide options to filter the data or to transform it as well as to access archival data. Long running sensor subscriptions may benefit from implementing trigger or alarm mechanisms. Imagine an online seismometer that notifies a student of a seismic event through email or instant messaging. The detection of a seismic event that passed a specified threshold might trigger more frequent sampling or complementary data

presentations. Sensor experiments frequently have very asymmetric data flows. It takes few bits to subscribe to a sensor, but the resulting data stream from the sensor to the student's client may require a great deal of bandwidth. Some sensors may only provide best efforts to deliver continuous data with no guarantee that all samples will arrive. Other systems may provide archival quality data but perhaps with a variable lag time. This category of experiment was represented at MIT by a flagpole instrumented with accelerometers that stream continuous data (flagpole.mit.edu).

1.2 Web services and Web application

Web services and web application are the two techniques adopted to implement the iLab project. This was decided based on the design requirement in the iLab project. The students on one campus must be able to use labs housed on different campus. The lab side services may need to run on different hardware and software platforms. Moreover, the lab-side campus may enforce different networking policies (e.g., firewalls, directory and email services). The transparency of web services makes this technology an obvious choice to integrate our distributed application framework..

Also, the loose coupling of web services makes it easy to reuse the preexisting code to manage the lab equipment or to display results and control the lab equipment from one or more client machines.

Web Services WSDL and UDDI provide a framework in which to help schools and colleges to discover what online labs in their area of interest are available and to verify whether those labs are compatible with the pedagogic goals of their courses. To carry this one step further, one can imagine WSDL-based negotiation that will match an Internet accessible lab with proprietary high end visualization and data analysis tools that are licensed by the client-side campus.

Since we can expect tens of thousands users will be world wide distributed, the ability to update and maintain Web applications without distributing and installing software on potentially thousands of client computers is the key reason the web application technology is one the foundations of the iLab architecture.

Because we based our shared software infrastructure for Internet accessible labs on web services and web application, we give a brief introduction about web services and web application techniques.

1.2.1 Web services

The W3C defines a Web service [1] as a software system designed to support interoperable machine to machine interaction over a network. Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate using XML messages that follow the SOAP-standard [13]. Common in both the field and the terminology is the assumption that there is also a machine readable description of the operations supported by the server, a description in the WSDL. The latter is not a requirement of a SOAP endpoint, but it is a prerequisite for automated client-side code generation in the mainstream Java and .NET SOAP frameworks. Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a Web service.

The specifications that define Web services are intentionally modular, and as a result there is no one document that contains them all. Additionally, there is neither a single, nor a stable set of specifications. There are a few "core" specifications that are supplemented by others as the circumstances and choice of technology dictate, including:

SOAP- An XML-based, extensible message envelope format, with "bindings" to underlying protocols. The primary protocols are HTTP and HTTPS, although bindings for others, including SMTP and XMPP, have been written.

WSDL- An XML format that allows service interfaces to be described, along with the details of their bindings to specific protocols. Typically used to generate server and client code, and for configuration.

[1] many sources also capitalize the second word, as in Web Services

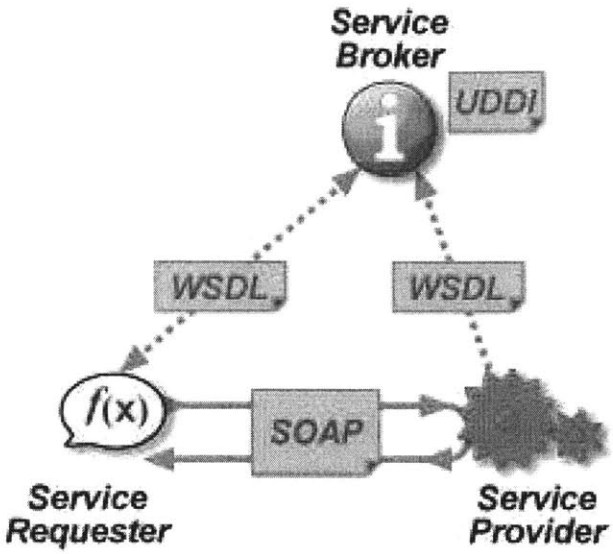

Figure 1-1: Web services

UDDI- A protocol for publishing and discovering metadata about Web services, to enable applications to find Web services, either at design time or runtime.

Most of these core specifications have come from W3C, including XML, SOAP, and WSDL; UDDI comes from OASIS.[14, 6]

1.2.2 Web Application

In software engineering, a Web application, or "webapp", is an application that is accessed with a Web browser over a network such as the Internet or an intranet. Web applications are popular due to the ubiquity of the browser as a client, sometimes called a thin client. The ability to update and maintain Web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity. Web applications are used to implement Webmail, online retail sales, online auctions, wikis, discussion boards, Weblogs, Massive multi-player online role-playing game (MMORPG) and many other functions. In earlier types of client-server computing, each application had its own client program which served as its user interface and had to be separately installed on each user's personal computer. An upgrade to the server part of the application would typically require an

upgrade to the clients installed on each user's workstation, adding to the support cost. In contrast, Web applications dynamically generate a series of Web documents in a standard format such as HTML/XHTML supported by common browsers. Client-side scripting in a standard language such as JavaScript is commonly included to add dynamic elements to the user interface. Generally, each individual Web page is delivered to the client as a static document, but the sequence of pages can provide an interactive experience, as user input is returned through Web form elements embedded in the page markup. During the session, the Web browser interprets and displays the pages, and acts as the universal client for any Web application. A significant advantage of building Web applications to support standard browser features is that they should perform as specified regardless of the operating system or OS version installed on a given client. Rather than creating clients for MS Windows^{TM}, Mac OS X^{TM}, GNU/Linux^{TM}, and other operating systems, the application can be written once and deployed almost anywhere. However, inconsistent implementations of the HTML, CSS, DOM and other browser specifications can cause problems in web application development and support. Additionally, the ability of users to customize many of the display settings of their browser (such as selecting different font sizes, colors, and typefaces, or disabling scripting support) can interfere with consistent implementation of a Web application. Another (less common) approach is to use Macromedia Flash^{TM} or Java^{TM} applets to provide some or all of the user interface. Since most Web browsers include support for these technologies (usually through plug-ins), Flash- or Java-based applications can be implemented with much of the same ease of deployment. Because they allow the programmer greater control over the interface, they bypass many browser-configuration issues, although incompatibilities between Java or Flash implementations on the client can introduce different complications. Because of their architectural similarities to traditional client-server applications, with a somewhat "thick" client, there is some dispute over whether to call systems of this sort "Web applications"; an alternative term is "Rich Internet Application".[9] Though many variations are possible, a Web application is commonly structured as a three-tiered application. In its most common form, a Web

browser is the first tier, an engine using some dynamic Web content technology (such as ASP, ASP.NET, CGI, ColdFusion, JSP/Java, PHP, Python, or Ruby On Rails) is the middle tier, and a database is the third tier. The Web browser sends requests to the middle tier, which services them by making queries and updates against the database and generating a user interface.

1.3 Challenges in interactive architecture

According to the different features of the three categories of experiments mentioned in 1.1, the iLab project envisions three different architectures to support those experiments, which are iLab Batched Architecture, iLab Interactive Architecture and iLab Sensor Architecture. Each of the architecture has its own challenges. Since this thesis only deals with the interactive architecture, the scheduling and security challenges, the main challenges in the iLab Interactive Architecture will be discussed in this chapter.

1.3.1 Scheduling challenge in iLab

The interactive architecture permits students to observe the progress of the experiment and to interact with the experiment in ways that can change the experiment's course. Such labs typically require more time to execute than batched experiments because they proceed in human not machine time. A typical interactive experiment requires 20 minutes to several hours to execute. Because users control the lab equipment, they usually require exclusive access to it. Asking users to queue for their turn to use the lab wastes their time.

Hence most interactive experiments require a scheduling application that allows the users to sign up in advance for time on a particular piece of lab equipment. Access to this scheduling application must be authorized by the Interactive Service Broker (ISB) since only the ISB can authenticate a user and vouch for his identity. The scheduling application must notify users if their reservation must be canceled or changed. Finally certain labs have operating requirements that require actions

either before or after the execution of an experiment. For instance, a heat exchanger experiment may require the apparatus to achieve thermal equilibrium before the start of an experiment. A chemical diffusion experiment may require that the diffusion apparatus be flushed at the end of the experiment. The scheduling application must allocate time for these actions in scheduling experiment sessions.

Scheduling can be looked at from two perspectives. From the lab provider's perspective, the scheduling application coordinates reservations to use a lab from multiple campuses. The scheduling server is also the process that holds the information required to "wake up" a lab server to perform required actions before a scheduled experiment. The lab provider may want to allocate blocks or percentages of time to different groups of users according to certain policies. For instance, Lab A may want to permit use from universities B and C between 6 pm and midnight, Monday through Friday, with university B allocated 60% of the time and university C the remainder. On the other hand, the lab provider generally does not want to be aware of the details of a user's reservation. If the lab server must be taken down for maintenance, the lab provider would simply like to notify the scheduling application of the down time and have the scheduling application take care of informing the affected users and rescheduling their work.

From a teacher's and a student's perspective, the scheduling application must act as their agent in scheduling time on lab servers. The application must accept authorizations to schedule from the users' ISB and must record reservations in a way that can be associated with individual users. If a reservation must be cancelled, the scheduling application must take the responsibility for informing the user. Teachers may want to stipulate policies that govern how their students may make reservations. For instance, a teacher may decide that students can only sign up for two hours of lab access per week with no single reservation lasting more than one hour. Different teachers using the same lab may want to set different policies for their students.

Given the different requirements from the lab-side and the student-side perspectives, where should the scheduling application be located?

What is more, different teachers want to represent different policies. For example,

some teachers want to set the maximum time their students can reserve for executing the experiment, while some teachers want to set the minimum time their students have to execute the experiment. When some lab manager assign time slots to the particular groups, they prefer to assign whole continuous time block the those groups, while for other groups, they prefer to assign recurrent time blocks such as from Monday to Friday, 8:00am to 9:00pm. How to design user interface so that the teacher or the lab manager can assign policy and time slots flexibly is another challenge.

The iLab architecture provides an infrastructure to allow world wide distributed clients visit the remote online labs. It will always happen that a client wants to make reservation for executing experiments on the labs in a different time zone. To the user's convenience, the time represented to the user should always be the user's local time. During the design of the scheduling algorithm, how to deal with the delta of time input from different time zones is a major consideration.

1.3.2 Security challenge in iLab

To begin either an administrative or an experimental session with the iLab interactive architecture, the user must authenticate himself to the ISB. The reference implementation supplies a simple user name and password scheme carried out using a standard browser-based web application.

Because iLab interactive architecture is a multi-domain, multi-server environment, in addition to user authentication on the Service Broker, the security challenges lie in the other three sessions:

- Single sign on (SSO) for multiple web applications sessions: For example, after authentication, the user may indicate that he wishes to schedule a future lab session and chooses one of the labs to which he has access. The ISB would then redirect him to the web application of the User-side Scheduling Server (USS) that handles the reservations for that lab. The redirection must be accompanied by credentials sufficient to identify the user and to convince the USS to allow him to schedule a future experiment session.

- Client to Web services security session: For example, when the time has come for the student to execute the experiment, the ISB must launch the client with credentials that the lab server will recognize. The client will usually try to contact the lab server directly, and the lab server should only accept the connection if the client can provide authentication information to the lab server.

- Web service to web service session: For example,when the lab server needs to store experiment data, it must contact the user's ESS and present the forwarded credentials that will allow the Experiment Storage Server (ESS) to recognize who owns the data that is being stored and to decide whether the data should be accepted.

Chapter 2

Scheduling server design

2.1 Scheduling server design goal

From the functionality view, in the interactive iLab architecture, the scheduling server should provide the services to the lab server providers, the teaching faculties and the students so that:

- Lab server providers are able to assign permissions to particular groups to execute certain kinds of experiment during defined time blocks. Providers are able to define the time limit properties for the particular experiments which can be executed in the lab server, such as the minimum executing time or recover time. They are also able to set certain scheduling policies for different groups. They can also revoke previously scheduled lab sessions for unexpected maintenance. They can monitor the usage of their equipments by different resources.

- Teaching faculties are able to monitor the signups of their students and set policies under which students can reserve lab sessions.

- Students are able to make reservations for future lab sessions, check on their previously made reservations, and change or cancel these reservations. They should also be reminded how much time earlier or how much time left for their time slots executing the experiments.

From the system efficiency view, we expect that the lab server provider, the teaching faculties and the students will access the scheduling server with low latency.

From the system flexibility view, we expect that the one group can chose multiple scheduling servers for different experiments. Also the intelligence housed in the scheduling server can allow the teaching faculties and lab server providers to set more complex scheduling policies. For example, instead of adopting first come, first served policy, the lab server provider can set priorities to certain groups so that the reservation from those groups will always be accepted first.

2.2 Interactive architecture and topology

The need to coordinate reservations from multiple campuses for a single lab server argues that there should be a single scheduling application located close in network terms to the lab server. But the requirement to accommodate the different policies of individual teachers suggests the need for multiple scheduling applications, at least one on each student campus like batched service brokers. We have decided that the two perspectives require two related scheduling applications, a Lab-Side Scheduling Server (LSS) and a User-Side Scheduling Server (USS). By polling the LSS, the USS presents the available time slots to certain user.

As the result, the iLab Interactive Experiments architecture includes following servers and services: the Interactive Service Broker (ISB) which authenticates users and handles authorization and administrative issues, the Experiment Storage Service (ESS) which provides storage for binary and text records associated with experiments, and the Lab Server (LS) which actually executes experiments and typically stores the results by invoking web service methods on the ESS. In addition, students execute interactive experiments by running a Lab Client (LC). In order to support interactive experiments which require scheduled access, iLab interactive architecture envisions scheduling servers and services which enable students from different campus to reserve executing experiments. Since the user side and lab side require different scheduling functionalities, user sides scheduling server (USS) and lab side scheduling server (LSS)

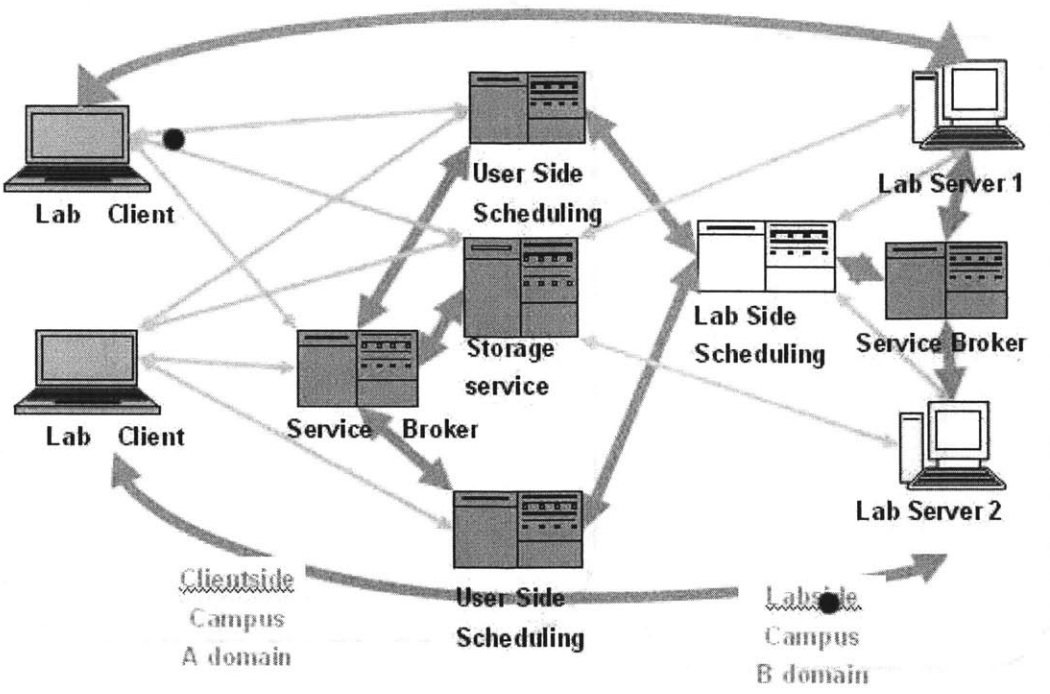

Figure 2-1: Ilab topology

are introduced in the iLab Interactive Services to mutually manage reservations from different students to execute different experiments.

The topology in the interactive architecture is as follows:

1. The relation between Service Brokers and USSs is many to many. In this way, a group is able to use multiple USS to manage their reservation on different experiment.

2. The relation between USSs and LSSs is many to many. This relationship allows great flexibility for students to chose the labs in different campus.

3. The relation between LSS and lab servers is one to many. This relationship is dictated by the unrepeatable nature of time. The time slots of one lab server only can be managed by one LSS.

2.3 Scheduling concepts and Rules

Some scheduling concepts should be clarified before the further discussion.

Experiment- The combination of a lab client and a lab server. The lab client is uniquely defined by lab client name and lab client version

Credential set- The combination of a Service Broker and an effective group accessed through a USS.

Time block- A period of time during which particular group can execute particular experiment.

Permitted experiment- Since multiple experiments can be executed in one lab server, the experiments which are permitted to execute in a certain time blocks are called permitted experiments.

Quantum- The minimum time unit for particular experiment.

Time slot- After dividing the time block by the quantum, we get number of time slots.

Early arrival time- Students are allowed to access the lab server if they arrive in a certain amount time earlier than his reservation. The allowed longest early arrival time is defined as the early arrival time.

Prepare time- Before the execution of the experiment, the warm up time the equipment needs is defined as the prepare time.

Recover time- After the execution of the experiment, the time the equipment needs to return to the normal state is defined as the recover time.

Minimum time- The minimum time the experiment needs to be executed.

Recurrence- The time block assigned to particular group can recur in certain style. Here we have four recurrence styles, which are daily, weekly, monthly and no recurrence. The aggregation of the time blocks in a certain time period with

the same recurrence style is called a recurrence. For example, a recurrence can include all the time blocks from 9am to 5pm every week day from June 1st 2007 to September 1st 2007.

Policy- The scheduling rule users need to comply with. Here we have user side policy and lab side policy. The user side policy is set by the teaching faculty of the particular group. The lab side policy is set by the lab server provider.

During the design of the scheduling servers, two rules are strictly complied with. The first rule is that the LSS policy should not involve any decisions based on user identity. All user-based policy should be implemented by USS policy and supported by the USS data model. The web service API between the LSS and USS should not have to communicate policy, only reservation information. In this way, the boundary of USS and LSS is clearly defined. The second rule is that any scheduling user interface should present times in the time of the current user. All times should be recorded in UTC. In this way, the time information can be shared globally.

2.4 Scenario

During the design of the scheduling server, there are seven scenarios we considered.

2.4.1 Scenario 1

The Student makes a reservation to execute particular experiment:

1. The student log in the ISB in his campus. He or she selects the group that has the permission to execute the experiment.

2. After the student click the Schedule/Redeem Session button, he is redirected to the Make Reservation Page on the USS which manages the reservations from the group the student selected from step one.

3. Through the calendar on the Make Reservation Page, the student selects the date on which he plan to do the experiment. The available time blocks to

execute the experiment on the selected day are retrieved from the LSS which manages the time slots of the lab server which provides the experiment. The student selects the time block he is interested in. All the time slots in the selected time block are retrieved from the LSS. At the same time the reservation policy defined by the TA of the group is represented on the page. The student selects the free time slots by clicking the check box next to them. After clicking the Make Reservation button, the USS checks whether the reservation policy is satisfied. If the reservation policy is satisfied, then the LSS checks whether the selected time slots are still free. If the reservation passes the two checks, the reservation will be represented in the Reservation window. Otherwise, the warning will be rejected.

2.4.2 Scenario 2

The student executes the experiment:

1. The student log in the ISB in his campus.He or she selects the group that has the permission to execute the experiment.

2. After the student click the Schedule/ Redeem Session button, he is redirected to the Make Reservation Page on the USS which manages the reservations from the group the student selected from step one.

3. After clicking the reservation showed in the reservation window to highlight the reservation the student is going to redeem, the student clicks on the Redeem Reservation button. If the current time is earlier than the start time of the reservation, the USS will remind the student how long until the reservation is valid. If the reservation is valid now, the student can launch the Lab Client of the experiment.

2.4.3 Scenario 3

The teaching faculty set the scheduling policy for his group:

1. After signing in the ISB as the manager of his group, the Lab Server provider is redirected to the policy management page of the USS which manages the reservation for his group.

2. After selecting the experiment whose scheduling policy is going to be set, the teaching faculty fills in the related text box to set the policy for his group.

2.4.4 Scenario 4

The Lab Server provider sets the experiment scheduling property and assigns the time blocks to particular group:

1. After signing in the ISB as the manager of the Lab Server, the Lab Server provider is redirected to the Manage page of the Lab Sever on the LSS.

2. In the experiment Information Management page, the Lab Server provider is able to set up the scheduling properties such as quantum, prepare time and recover time for particular experiment which is executable on his lab server.

3. In the TimeBlock Management page, the Lab Sever provider selects the group he is going to assign time blocks to. Then the Lab Server provider can assign the time blocks to the group by setting the recurrence style, the start time and end time, the start date and end date.

2.4.5 Scenario 5

The Lab Server provider revokes the time blocks for the unexpected maintenance:

1. After signing in the ISB as the manager of the Lab Server, the Lab Server provider is redirected to the Manage page of the Lab Sever on the LSS.

2. In the Revoke Reservation page, the Lab Server provider sets the start time and end time of the time period during which the lab server is going to be down for maintenance. All the reserved lab sessions during the time period are deleted and related USSes are notified.

2.4.6 Scenario 6

The USS Superuser checks the USS server records:

1. After signing in the ISB as the Superuser of the USS, the Superuser of the USS is redirected to the USS.

2. He can check information of the registered LSSes and registered Experiments on his server.

2.4.7 Scenario 7

The LSS Superuser checks the LSS server records

1. After signing in the ISB as the Superuser of the LSS, the Superuser of the LSS is redirected to the LSS.

2. He can check information of the registered USSes and registered groups on his server.

2.5 Functionality

According to the scenarios described above, we define the functionality for the USS and the LSS from the web application aspect and the web service aspect.

2.5.1 USS functionality provided by web application

- A user must be able to make reservations as well as to view, cancel or modify previously scheduled reservations.

- The super user of the USS must be able to check a Lab-side scheduling server (LSS) and the lab client on an USS.

- A teaching faculty should be able to review reservations made from his group. He or she, with required permission, should be able to make, modify, or cancel reservations for members of the group over which the staff member has authority.

- A teaching faculty should be able to specify a rule set that governs whether a reservation request to execute an experiment at a certain time will be accepted from a student with a particular credential set.

2.5.2 USS functionality provided by web services

- Allow an LSS to revoke a block of reservation (e.g., due to maintenance) and assume the responsibility for informing the affected staff and users.
- Redeem the reservation, that is, check whether the reservation from a user for a particular experiment is ready for execution.
- Allow the ISB in its domain to register the Groups and the Experiments.

2.5.3 LSS functionality provided by web application

- A lab administrator must be able to register the experiments provided by the Lab Server on an LSS.
- A lab administrator must be able to offer a block of time for reservations to one or more groups. A lab administrator must be able to specify rule sets (policies) to determine whether a reservation from a particular USS for a particular time should be accepted or not.
- A lab administrator should be able to check and revoke previously confirmed reservations to accommodate maintenance or other unexpected requirements of the lab server team.
- A super user of the LSS should be able to check the USSs and the Groups registered on his server.

2.5.4 LSS functionality provided by web services

The LSS must be able to provide a listing of available reservation time blocks for a given experiment and credential set within a particular USS specified interval.

The LSS must be able to provide a listing of time periods which are valid for a given experiment and credential set within a particular USS specified time block.

The LSS must be able to confirm or deny a particular reservation request from a USS; if the request is denied, the LSS should provide a brief explanation string.

The LSS must be able to invoke an alert method built into the interactive lab server web service. The purpose of this method is to wake up a lab server that needs to prepare for an upcoming scheduled reservation.

The LSS must be able to allow the USS to revoke the reservations.

The LSS must be able to register the USSs and Groups by the request of the ISB in its domain.

2.6 Work flow

In this section, we illustrate what work flows happens when a student from one campus schedules a reservation or executes an experiment on a different campus. Here, we call the student's campus domain A, and the campus where the lab service is located domain B.

2.6.1 Scheduling reservation work flow

1. The student logs in to the service broker through the student is web browser.
2. After the Interactive Service Broker in domain A (ISA) authenticates the student, the ISA redirects the client with coupon in the http request to the USS.
3. The student enters the time period in which he wants to make reservation.
4. The USS retrieves from the LSS available time periods with coupon ID, an identifier of the tickets collection, in the header.
5. The LSS called the the Interactive Service Broker in domain B (ISB) to retrieve scheduling ticket by showing the coupon ID.

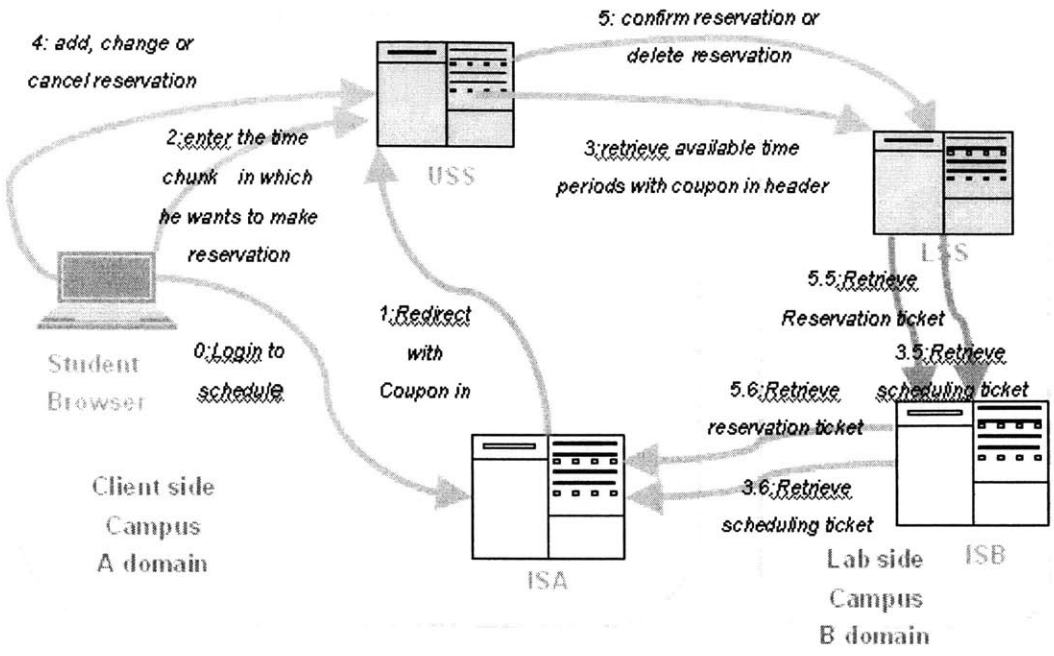

Figure 2-2: Scheduling reservation work flow

6. The ISB in domain B called the ISB in domain A to retrieve the scheduling ticket by showing the coupon ID

7. Once the ticket is received, the LSS sends the available time periods back to the USS. The USS returns the time slots available to the student.

8. The student adds or changes the reservation from the USS.

9. The USS confirms the reservation on the LSS to check whether the time slot is still available.

10. The LSS calls the ISB in domain B to retrieve the reservation ticket by showing the coupon ID.

11. The ISB in domain B calls the ISB in domain A to retrieve the reservation ticket by showing the coupon ID.

12. Once the ticket is received, the LSS adds or changes the reservation. The USS adds or changes the reservation as well.

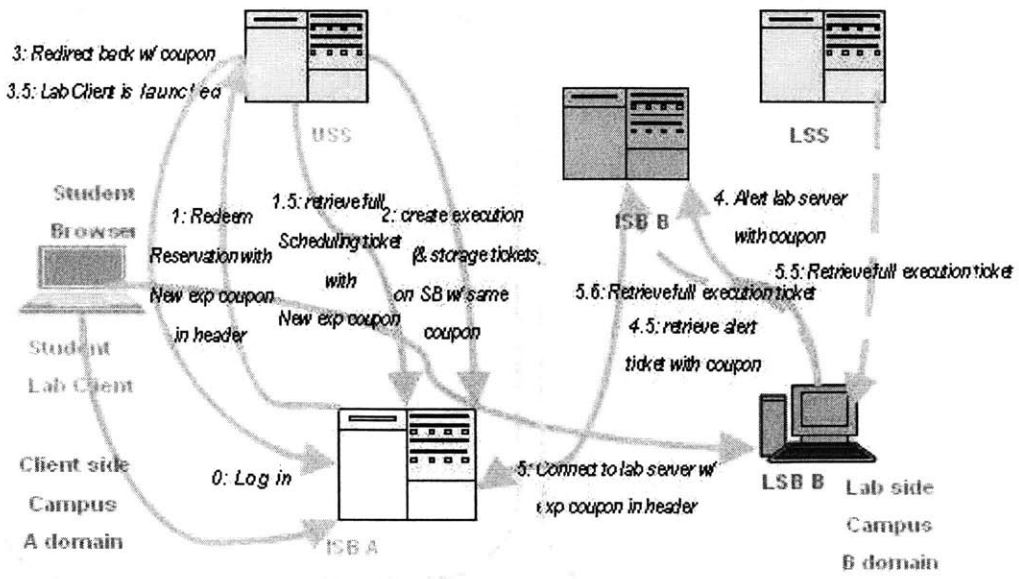

Figure 2-3: Scheduling execution work flow

2.6.2 Scheduling execution work flow

1. The student logins to the service broker through the student lab client.

2. After ISB A authenticates the student, the ISB A redeems the reservation on the USS for the student with the coupon by making an the http request to the USS. The USS retrieved the scheduling ticket from ISB in the domain A to check whether the student is authenticated to redeem the reservation. The lab client is launched.

3. Once the student is authenticated, and the USS finds the reservation for the student to execute the requested experiment now. The USS calls the ISB in domain A to create an execution ticket and storage ticket with the same coupon ID.

4. The Lab Client connects to the Lab Server with the coupon in the header of the SOAP.

5. The Lab Server calls the ISB in domain B to retrieve scheduling ticket by

showing the coupon ID.

6. The ISB in domain B calls the ISB in domain A to retrieve the execution ticket by showing the coupon ID.

7. Once the execution ticket is received, the lab client gets connected to the lab server.

8. At the same time, the LSS in domain B alerts the Lab Server and wake the Lab Server up if the LSS knows there is a reservation is to be fulfilled.

2.7 Data model

Five entities can be extracted from the scenario related with the USS. They are Reservation, Credential Set, Experiment Information, USS Policy and LSS Information. The relations among them are as follows:

- The Reservation has a foreign key, Credential Set ID, which is the Credential Set's primary key. This relation shows which group the reservation comes from.

- The Reservation has a foreign key, Experiment Info ID, which is the Experiment Info's primary key. This relation shows which experiment the reservation is made for.

- The Experiment Info has a foreign key, LSS Info ID, which is the LSS Info's primary key. This relation shows which LSS manages this experiment's scheduling information.

- The USS Policy has a foreign key, Credential Set ID, which is the Credential Set's primary key. This relation shows which group the USS Policy applies to.

- USS Policy has a foreign key, Experiment Information ID, which is the Experiment Information's primary key. This relation shows which experiment the USS Policy applies to.

The data model for the USS entities is shown in Figure 2-4.

Eight entities can be extracted from the scenario related with the LSS. They are Reservation Information, USS Information, Credential Set, LSS Policy, Time Block, Recurrence, Experiment Information and Permitted Experiment. The relations among them are as follows:

- The Reservation Info has a foreign key, Credential Set ID, which is the Credential Set's primary key. This relation shows which group the reservation comes from.

- The Reservation Info has a foreign key, Experiment Information ID, which is Experiment Information's primary key. This relation shows which experiment the reservation is made for.

- The Credential Set has a foreign key, USS Information ID, which is the USS Information's primary key. This relation shows which USS manages the reservation information from the group with this credential set.

- The LSS Policy has a foreign key, Experiment Information ID, which is the Experiment Information's primary key. This relation shows which experiment the LSS Policy applies to.

- The LSS Policy has a foreign key, Credential Set ID, which is the Credential Set's primary key. This relation show which group the LSS Policy applies to.

- The Time Block has a foreign key, Credential Set ID, which is the Credential Set's primary key. This relation shows which group the time block is assigned to.

- The Time Block has a foreign key, Recurrence ID, which is the Recurrence's primary key. This relation shows which Recurrence the time block belongs to.

- The Permitted Experiment has a foreign key, Experiment Information ID, which is the Experiment Information's primary key. The Permitted Experiment has a foreign key, Recurrence ID, which is the Recurrence's primary key. These

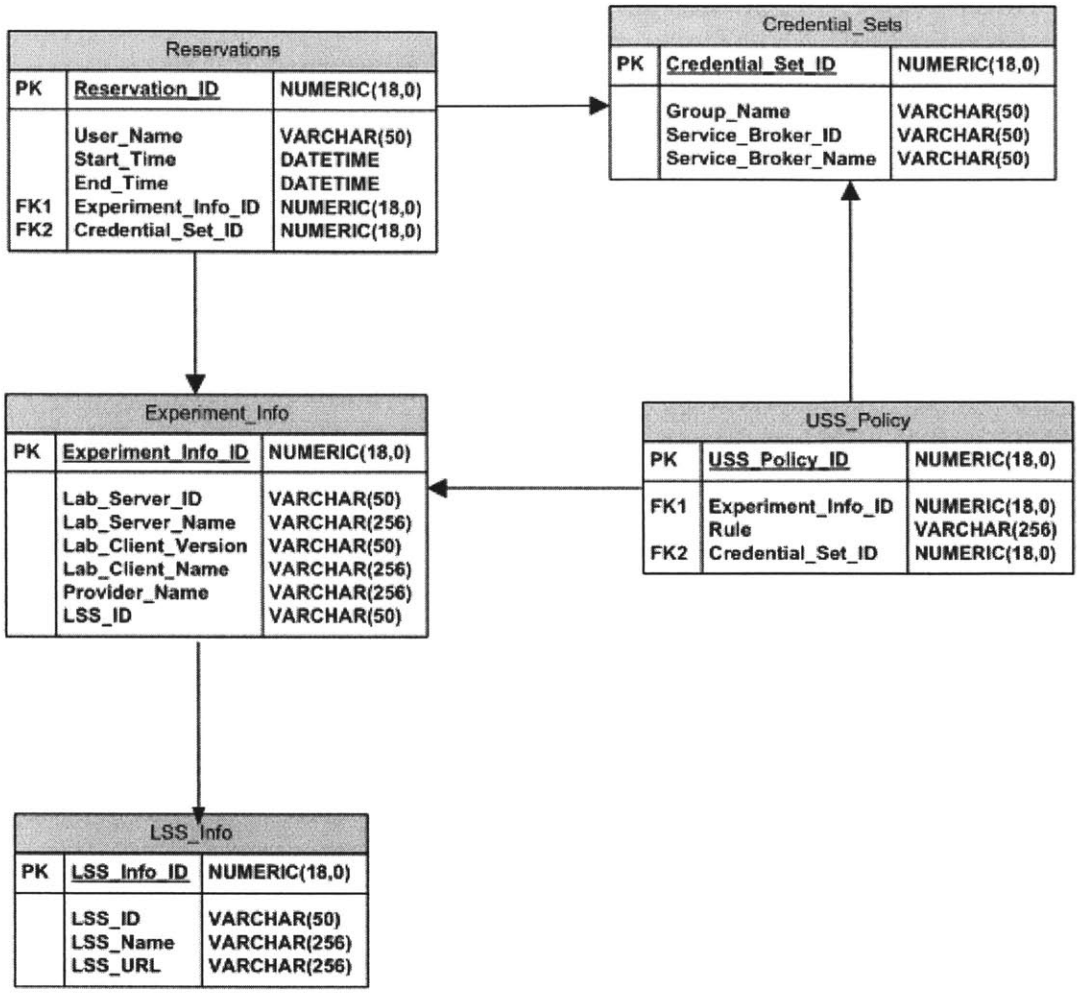

Figure 2-4: USS Data Model

two relations show the maps between multiple Recurrences and multiple experiments. That means multiple experiments can be permitted to be executed in the certain Recurrence, and multiple Recurrences can be assigned to execute a certain experiment.

Given all the relation among all the entities in the USS and the LSS, we build up the database according to the data model shown in Figure 2-5.

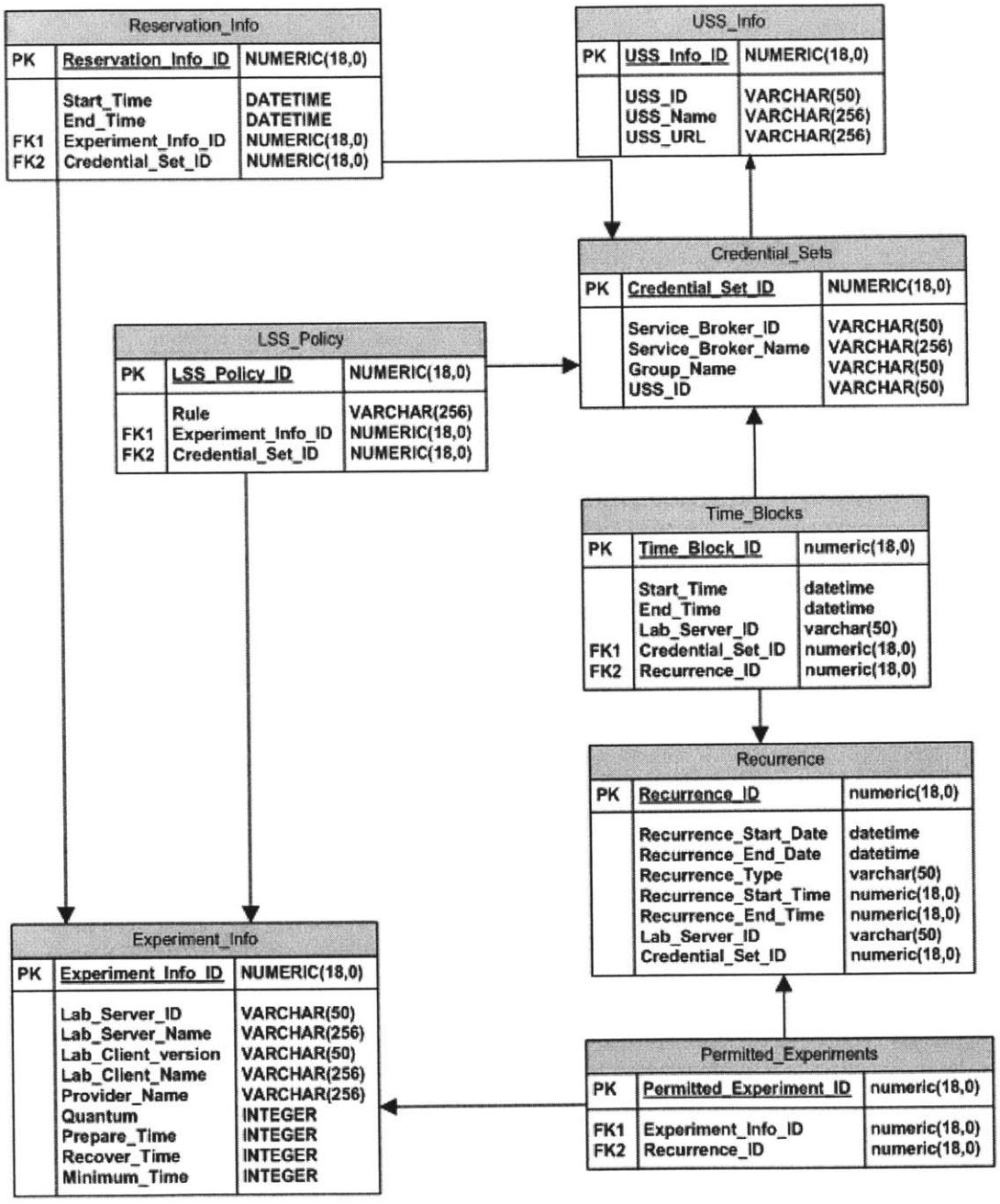

Figure 2-5: LSS Data Model

Chapter 3

Scheduling server implementation

3.1 Scheduling algorithm

In this section, the Retrieve Available Time Periods algorithm as the main scheduling algorithm will be introduced.

The goal of the Retrieve Available Time Periods algorithm is to retrieve sequential available time periods (in local time of the LSS) which are the minimum available time periods during which a particular group has permission to execute a particular experiment.

During the design of this scheduling algorithm, we should keep three time axes in mind, as the figure showed above, the first time axis (TA1) shows the time blocks during which the user has the permission to execute the particular experiment. The second time axis (TA2) shows the time chunk which is defined by the input parameters of the start time and end time. The third time axis(TA3) shows the distribution of the unavailable time which includes the reservations already made plus the cool down and warm up times needed for those reservations. The basic idea is to first intersect TA1 with TA2 to get the time blocks during which the user has the permission to execute the experiment in the time chunk. We then calculate the free time during the time chunk, which is the time chunk minus the unavailable time. We then intersect the time blocks retrieved from the first step with the free time retrieved from the second step. After filtering the time periods which are shorter than the minimum

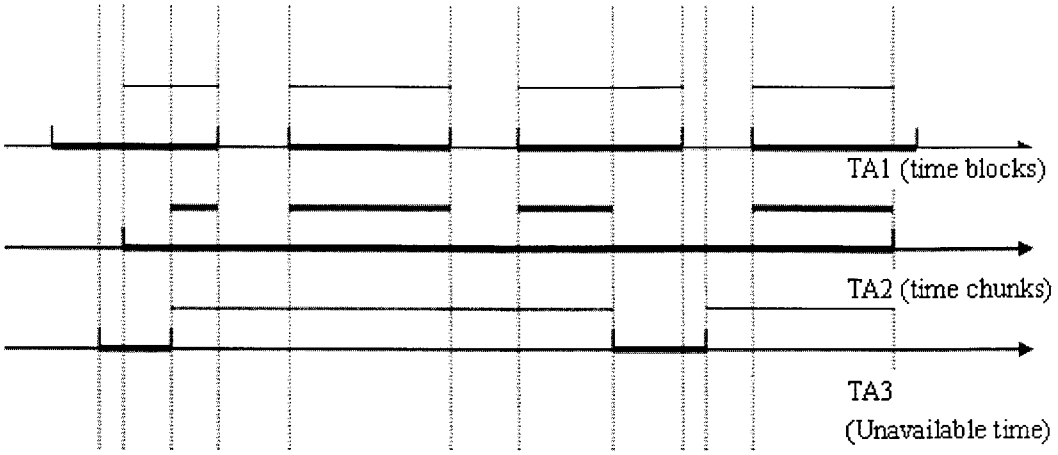

Figure 3-1: Time Intersection

execution time for the experiment, we get the available time periods on which the user can make reservation.

The algorithm in detail is as follows:

1. Get the experiment configuration.

2. Get the time blocks which are the minimum time blocks set covering the time chunk for the lab server where the experiment is executed.

3. Get the IDs time blocks in the time chunk for which the user's group has permission.

4. According to the IDs of the time blocks retrieved in the step 3, get the time blocks which are the minimum time blocks covering the time chunk, and for which the user's group has permission.

5. Get the reservations made for the lab server whose experiment time, including the warm up and cool down time, overlaps with the time chunk.

6. Get the unavailable time periods (in UTC) which overlap with the time chunk defined by the start time and end time

7. Get the free time periods (in UTC) during the time chunk defined by the start time and end time.

8. Get the available time periods (in UTC) For each available time block, get the free time periods which are overlapping with each available time block, intersect the time block with the corresponding free time periods, and convert the result into local time.

9. Select the time periods from the available time periods from the previous step which are longer than the minimum time the experiment needed.

10. Sort the time periods sequentially.

3.2 Data structure

During the implementation of the Scheduling Server, we defined following data structures:

3.2.1 User-side Scheduling Server

```
/* This structure describes the attributes of a group's credentials. */
public struct CredentialSet
        {
                int credentialSetID;
                String serviceBrokerID;
                String serviceBrokerName;
                String groupName;
        }
/* This structure describes the attributes of a reservation.*/
public struct Reservation
        {
                int reservationID;
```

```
            String userName;
            int credentialSetID;
            DateTime startTime;
            DateTime endTime;
            int experimentInfoID;
        }
/* This structure describes the attributes of a policy for user-side scheduling.*/
public struct USSPolicy
        {
            int ussPolicyID;
            int experimentInfoID;
            String rule;
            int credentialSetID;
        }
/* This structure describes the attributes of an experiment.*/
public struct ExperimentInfo
        {
            int experimentInfoID;
            String labServerID;
            String labServerName;
            String labClientVersion;
            String labClieIntName;
            String providerName;
            String lssID;
        }
/* This structure describes the attributes of a Lab-side Scheduling Server.*/
public struct LSSInfo
        {
            int lssInfoID;
            String lssID;
```

```
        String lssName;
        String lssURL;
    }
```

3.2.2 Lab-side Scheduling Server

```
/* This structure describes the attributes of a group's credential.*/
public structCredentialSet
    {
        int credentialSetID;
        String serviceBrokerID;
        String serviceBrokerName;
        String groupName;
        String ussID;
    }
/* This structure describes the attributes of a reservation.*/
public struct ReservationInfo
    {
        int reservationInfoID;
        int credentialSetID;
        DateTime startTime;
        DateTime endTime;
        int experimentInfoID;
    }
/* This structure describes the attributes of a time block during which a particular
group has the permission to execute the experiment on a particular lab server. */
public struct TimeBlock
    {
        int timeBlockID;
```

```
        int credentialSetID;
        DateTime startTime;
        DateTime endTime;
        String labServerID;
        int recurrenceID;
    }
/*This structure describes the attributes of a experiment.*/
public struct ExperimentInfo
    {
        int experimentInfoID;
        String labServerID;
        String labServerName;
        String labClientVersion;
        String labClielntName;
        String providerName;
        int quantum;
        int prepareTime;
        int recoverTime;
        int minimumTime;
    }
/* This structure describes the attributes of a Lab-side scheduling policy.*/
public struct LSSPolicy
    {
        int lssPolicyID;
        int credentialSetID;
        string rule;
        int experimentInfoID;
    }
/* This structure describes the attributes of a permission which a particular experiment
has to be executed in the recurrent time periods. */
```

```
public struct PermittedExperiment
    {
        int permittedExperimentID;
        int experimentInfoID;
        int recurrenceID;
    }
/* This structure describes the attributes of a set of time periods which are in the same
recurrent pattern. */
public struct Recurrence
    {
        int recurrenceID;
        DateTime recurrenceStartDate;
        DateTime recurrenceEndDate;
        DateTime recurrenceStartTime;
        DateTime recurrenceEndTme;
        String recurrenceStyle;
        int labServerID;
        int credentialSetID;
    }
/* This structure describes the attributes of a User-side Scheduling Server. */
public struct USSInfo
    {
        int ussInfoID;
        String ussID;
        String ussName;
        String ussURL;
    }
```

3.3 Web Service interface

3.3.1 User-side Scheduling Server

RedeemReservation

Purpose:

/*Returns a Boolean indicating whether the reservation identified by reservationID is ready for execution */

Arguments:

int reservationID

/* The ID identifying the reservations which need to be redeemed. */

Returns:

bool redeemed

/*True if reservation was redeemed. In order for this happen, the current time needs to be covered by the time period defined by the startTime and endTime of the reservation; false otherwise.*/

RevokeReservation

Purpose:

/*Remove all the reservation for certain lab server being covered by the revocation time and send emails to the affected staff and users */

Arguments:

string labServerID

/* The ID identifying the lab server whose time is being revoked */

DateTime startTime

/* The start time of the revocation period. */

DateTime endTime

/* The end time of the revocation period. */

Returns:

void none

3.3.2 Lab-side Scheduling Server

ConfirmReservation

Purpose:

/* Returns a Boolean indicating whether a particular reservation
from a USS is confirmed and successfully added to the database in LSS .
If it fails, an exception will be throw out indicating the reason for
rejection.*/

Arguments:

string serviceBrokerID
/* The Global Unified Identity (GUID) identifying the service broker which is in the same domain of the user */
string groupName
/* The name of the group whose member made the reservation requested*/
string ussID
/* The GUID identifying the user side scheduling server which the reservation is requested from*/
string labClientName
/* The name of the client of the experiment whose time is requested to be reserved*/
string labClientVersion
/* The version of the client of the experiment whose time is requested to be reserved*/
DateTime startTime
/* The startTime of the reservation requested. Note that startTime is the time in UTC */
DateTime endTime
/* The endTime of the reservation requested. Note that endTime is the time in UTC */

Returns:

bool confirmed

/* If validated and successfully added to the database in LSS, true; otherwise, false. Before the reservation is added to the database of LSS, LSS needs to judge whether the reservation information is confirmed. To validate the confirmation, all the following must be satisfied.

1. The reservation is in the currently available time periods for the group that the reservation comes from.

2. All the corresponding lab server side policies which the reservation comes from should be satisfied.

3. All the scheduling properties for the experiment which the reservation is made to should be satisfied. */

RemoveReservationInfo

Purpose:

/* Remove reservation information. */

Arguments:

string serviceBrokerID

/* The GUID identifying the user side scheduling server which the reservation is requested from*/

string groupName

/* The name of the group whose member made the reservation removed*/

string ussID

/* The GUID identifying the user side scheduling server which the reservation removed is requested from */

string labClientName

/* The name of the client of the experiment whose time is reserved*/

string labClientVersion

/* The version of the client of the experiment whose time is reserved*/

DateTime startTime

/* The startTime of the reservation removed. note that startTime is the time

in UTC */

DateTime endTime

/* The endTime of the reservation removed. note that endTime is the time in UTC */

Returns:

bool removed

/* True if the reservation is successfully removed; false otherwise */

RetrieveAvailableTimePeriods

Purpose:

/* Retrieve all the available time for a given experiment and credential set within a particular USS specified interval */

Arguments:

string serviceBrokerID

/* The GUID identifying the service broker which is one of the properties of the credential set*/

string groupName

/* The name of the group which is one of the properties of the credential set */

string ussID

/* The GUID identifying the user side scheduling server which is one of the properties of the credential set */

string labClientName

/* The name of the client of the experiment whose available time is requested*/

string labClientVersion

/* The version of the client of the experiment whose available time is requested*/

DateTime startTime

/* The start Time of time the particular USS specified interval */

DateTime endTime

/* The end Time of time the particular USS specified interval */

Returns:

ArrayList availableTimePeriods

/* ArrayList containing available time periods for a given experiment and credential set within a particular USS-specified interval */

3.4 Scheduling user interface

The Cascading Style Sheets (CSS) was adopted to unify the style of all the web pages in the scheduling server.

CSS is a stylesheet language used to describe the presentation of a document written in a markup language. CSS is used by both the authors and readers of web pages to define colors, fonts, layout, and other aspects of document presentation. It is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation (written in CSS). This separation can improve content accessibility, provide more flexibility and control in the specification of presentational characteristics, and reduce complexity and repetition in the structural content. CSS can also allow the same markup page to be presented in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on braille-based, tactile devices. Similarly, identical HTML or XML markup can be displayed in a variety of styles or color schemes by using different CSS. CSS specifies a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called cascade, priorities or weights are calculated and assigned to rules, so that the results are predictable.

Advantages of using CSS include:

- Presentation information for an entire website or collection of pages can be held in one CSS file, allowing sweeping changes to be propagated with quick changes to this one file.

- Different users can have different style sheets: for example a large text alternative for visually-impaired users, or a layout optimized for small displays for mobile phones.

- The document code is reduced in size and complexity, since it does not need to contain any presentational markup.

Also we use the user control technique in ASP.Net[8] to make customized Banner and Footer for each pages in the scheduling server. So when different campus install the Scheduling Server, they can customized their banner according to their preferred feeling and look easily. Web user controls define controls easily as desired for the application, using the same programming techniques as used to write Web Forms pages. Web Form pages can be converted into a Web user control with a few modifications. To make sure that a user control cannot be run as a stand alone Web Forms page, user controls are identified by the file name extension. ascx. A Web user control is similar to a complete Web Forms page, with both a user interface page and a code-behind file. The user interface page differs from an .aspx file in these ways:

- The extension must be .ascx.

- The user control does not have <HTML>, <BODY>, and <FORM> elements in it (these elements must be in the hosting page).

In every other way, a user control is like a Web Forms page. Similar HTML elements and Web controls can be used on a user control as done on a standard Web Forms page. For example, if you are creating a user control to be used as a toolbar, you can put a series of Button Web server controls onto the control and create event handlers for the buttons.

Chapter 4

Security in the iLab Interactive architecture

4.1 Introduction to security system

A typical implementation of the protection model is layered as shown in Figure 4-1. The bottom layer contains cryptographic transformations: ciphers, pseudo-random number generators, and cryptographic hashes. The transformations can be used to protect against attacks on a message. These transformations can be used to implement security primitives, which are the next layer up. Operations such as sign generate the signature for a message and Verify, using the signature, checks that the message has not been modified. Encrypt transforms the message so that it can not be read by attackers, and Decrypt untransforms the message so that the recipient can read it.

Access control lists and capabilities can be used to implement authorization. Cryptographic protocols, the top layer, combine the security primitives to implement secure client/server applications such as setting a secure communication path from a web browser to a web server. In this section, how to realize authentication, authorization and confidentiality will be elaborated.

application	Cryptographic protocols		
Function	authentication	authorization	confidentiality
Security primitive	Sign and verify	Access control lists and capabilities	Encrypt and decrypt
cryptography	Ciphers, pseudo-random number generators, and cryptographic hashes		

Figure 4-1: Security System

4.1.1 Authentication

As computer systems have increased in complexity, the challenge of authenticating users has also increased. As a result, there are a variety of models for authentication. There are two models for web service authentication, which are referred to as direct authentication and brokered authentication. Both the Direct Authentication pattern and the Brokered Authentication pattern focus on the relationships that exist between a client and service participating in a Web service interaction. When both the client and service participate in a trust relationship that allows them to exchange and validate credentials including passwords, direct authentication can be performed. In a situation where the client and service do not share a direct trust relationship, broker authentication can be performed. In a multi-server environment, the authentication broker can manage trust centrally. As for the brokered authentication, there are three brokered authentication design patterns that illustrate authentication using the Kerberos protocol [10], X.509 [8], and a Security Token Service (STS) respectively.

As Figure 4-2 shows, there are two ways to implement the patterns described above using Microsoft technologies. Transport layer security represents an approach where the underlying operating system or application servers are used to handle security features. For data confidentiality, Secure Sockets Layer (SSL) [5] is a common transport layer approach that is used to provide encryption. Message layer security represents an approach where all the information related to security is encapsulated

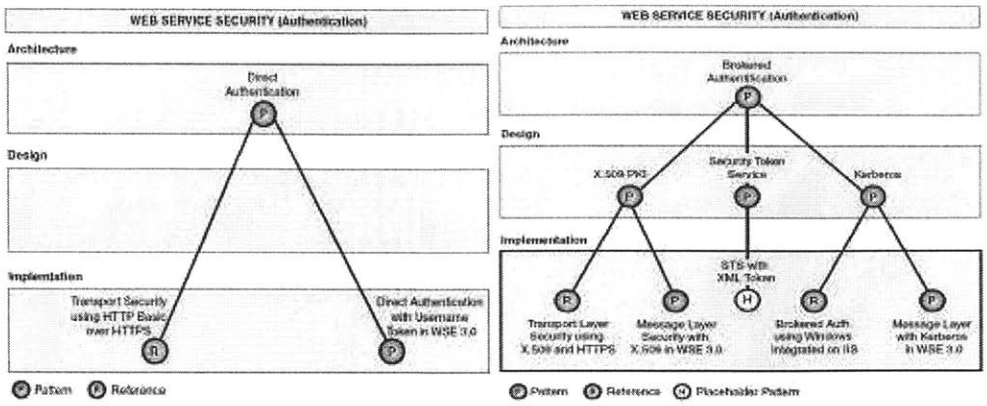

Figure 4-2: The direct authentication and the brokered authentication patterns

in the message [1].

4.1.2 Authorization

Authorization answers the question: is the principal authorized to request the specified operation? There are three primary operations in authorization systems: authorization, mediation and revocation. Authorization is the operation granting a principal permission to perform an operation on an object. Mediation is the operation checking if a principal has permission to perform an operation on a particular object. Revocation removes a previously granted permission from a principal. The authority can increase or decrease the set of principals that have access to a particular object by authorizing or revoking respectively their permissions.

There are three different models to keep track of who is authorized and who is not. They are the simple guard model, the caretaker model and the flow control model. In the simple guard model, the service conceptually surrounds each object with a wall that has only one door providing access to the object. The service posts a guard at the door who decides whether a principal has access or not. The principal and the guard both have a token. If the token presented by the principle is the same with the one the guard has, the principal is granted access. In the caretaker model, objects are only accessible through the caretaker. There is only a mail slot for communicating

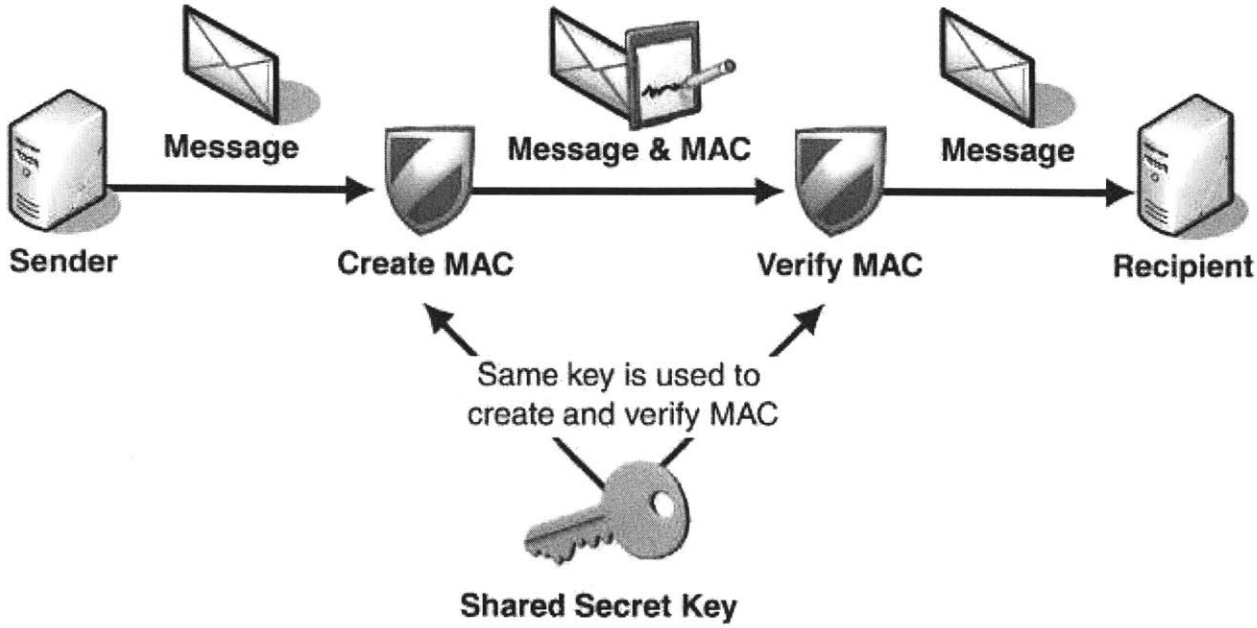

Figure 4-3: Shared key system

with the caretaker. The caretaker is intelligent. It can enforce arbitrary constraints on access, and it may even interpret the data stored in the object to decide what to do with a given request. In flow control model can provide both nondiscretionary and discretionary control. It allows untrusted programs to work with sensitive data, but confines all program output to prevent unauthorized disclosure.

4.1.3 Confidentiality

Using the Encrypt and Decrypt primitives, the Sender and Recipient can ensure the communication between them is confidential. The Encrypt and Decrypt primitive can be implemented using cryptographic transformations. Encrypt and Decrypt can be either shared-secret systems or public key systems. In a shared-secret system, the Sender and the Recipient share a key that only they know. In a public key system, the Recipient has a key pair (public key, private key). The Recipient gives his public key through an existing channel. Given the Recipient's public key, the Sender can encrypt the message with the Recipient's public key and send the encrypted message over an insecure network. Only the Recipient can decrypt the message[8].

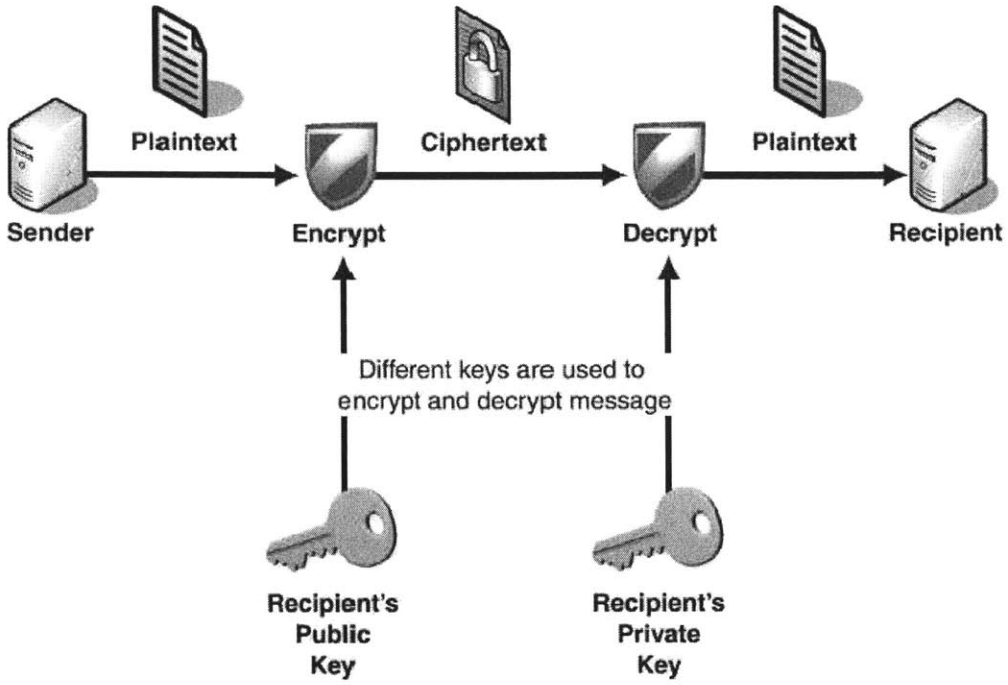

Figure 4-4: Public key system

4.2 Web service security

Getting Web Services off the ground means keeping them simple; however, providing security is seldom simple. Microsoft and IBM, among others, are working together to address this issue. Their efforts have resulted in a group of specifications for providing Web Services security, which includes WS-Security, WS-Trust, WS-SecureConversation and Web Service Security Profile for XML-based Tokens. Taken as a group, these specifications lay the foundation for a usable, interoperable, and quite complete approach to providing security for Web Services.[3] WS-Security is the foundation for all the other specifications.

WS-Security defines no new security technology. Instead, it focuses on applying existing effective mechanisms for distributed security, including Kerberos, public key technologies, and others to SOAP [13] messages. The Web Service Enhancements (WSE) is the Microsoft .NET implementation of WS-Security.

4.3 Threat analysis in the iLab Interactive architecture

iLab interactive architecture is an architecture to support interactive experiments that are remotely accessed in a multi-domain, multi-server environment. The current iLab interactive architecture envisions the following servers and services: the Interactive Service Broker (ISB) which authenticates users and handles authorization and administrative issues, the Experiment Storage Service (ESS) which provides storage for binary and text records associated with experiments, the Lab-Side Scheduling Server (LSS) and the User-Side Scheduling Server (USS) which mutually work to enable students to reserve time for executing experiments during the time blocks registered by the lab administrators for their lab server, and the Lab Server (LS) which actually executes experiments and typically stores the results by invoking web service methods on the ESS. In addition, students execute interactive experiments by running a Lab Client (LC).[11] These services host both web applications and web services. The security threats in iLab system can be classified into three categories:

Unauthenticated information release- This threat comes from malicious students reading and taking advantage of information stored in the iLab system or being transmitted over networks, such as eavesdropping on other students experiments results.

Unauthenticated information modification This threat comes from unauthorized students making changes in the stored information or modifying messages that cross a network, such as changing the experiment results in the Experiment Storage Server, or modifying the grades given by the TA.

Unauthenticated denial of use This threat comes from an intruder preventing an authorized students or TAs from reading or modifying the information. Causing iLab servers "crash", flooding a lab service by replaying messages are examples of denial of use.

Authentications play very important role in interactive iLab system. According to the principle of the iLab architecture, authentications in the interactive iLab architecture can be classified into direct authentication and indirect authentication. The direct authentication is the credential management for the user to access the Service Broker web application. The iLab interactive architecture assumes that the user's identity is confirmed by the Interactive Service Broker (ISB) at the start of every session. The ISB may accomplish this using a simple user name and password scheme. The indirect authentication is necessary to bridge to web applications and web services hosted on other servers after the user authenticates himself on the Service Broker so that once the user has authenticated himself, he should not have to do so again to perform any allowed action or to access any allowed resource within his iLab environment.

The complexity of the higher level authentication between iLab processes only increases when one considers collaboration between domains, e.g., authenticating a user in one domain and making a reservation through her own USS to execute an experiment hosted on a lab server in another domain. Incorporating a security token service (STS) to the ISB is my proposed solution to this problem. The STS can issue security tokens for the client on behalf of the ISB, which manages all the credential information of the service providers and users in its domain. These security tokens can be used by the target services, LSS, USS, ESS or LS to authenticate the client. The security token is always verified, but the service does not need to interact with the ISB to perform the verification. This is because the token itself contains proof of a relationship with the Service Broker, which can be used by the service to verify the token.

4.4 Current solution

The General Ticket scheme is used currently to deal with authentication cross multiple domains. In the General Ticket scheme, tickets are small XML documents to convey users' credentials. There are three parties in the General Ticket scheme which are the

ticket holder, the ticket redeemer and the ticket issuer. The process that is making the web service request or page access is known as the ticket holder. The process to which the holder directs its request is known as the ticket redeemer. The server that creates the credentials to authorize the operation is known as the ticket issuer. The ticket holder can be one of the distributed interactive services or web applications such ac the User-side Scheduling Server (USS) or the user's lab client. The ticket redeemer in an operation is always that part of the distributed iLab architecture that controls or manages the resource that the user is trying to access. A lab server acts as a ticket redeemer during the execution of an experiment, and the User-side Scheduling Server (USS) acts as a ticket redeemer when confirming a user's reservation to perform an experiment. The ticket issuer is always an Interactive Service Broker (ISB). The ticket issuer creates and keeps the definitive copy of all tickets so that all ticket redeemers must trust it. It must supply ticket holders with the ticket coupons they need to authorize their access to resources. When a service provider, which is always a ticket redeemer, receives a web service call from a client (which is usually the ticket holder), the client has to show the ticket coupon ID which is provided by the ISB (the ticket issuer) to the ticket redeemer. The ticket redeemer uses the coupon ID to retrieve the whole ticket from the ISB. If the ticket can be retrieved from the ISB, the action is authenticated by the ISB. Since ISB is the central manager for all the service providers in the local domain, it is reasonable for the ticket redeemer to trust the tickets issued by the ISB in its own domain. To deal with the cross domain authentication issue, the current solution makes the ISBs from the two domain the trust bridge between the clients from one domain and the services from the other domain.

Consider the Scheduling scenario as an example. Once the USS which is in domain A wants to call the web method "confirm reservation" in the LSS in domain B, it needs the Interactive Service Broker in domain A (ISB A) to create a "REQUEST RESERVATION" ticket after being authenticated by ISB A. The coupon ID identifying the scheduling tickets is attached to the SOAP request as part of the SOAP header. When the LSS receives the web method call from the USS, it has to verify

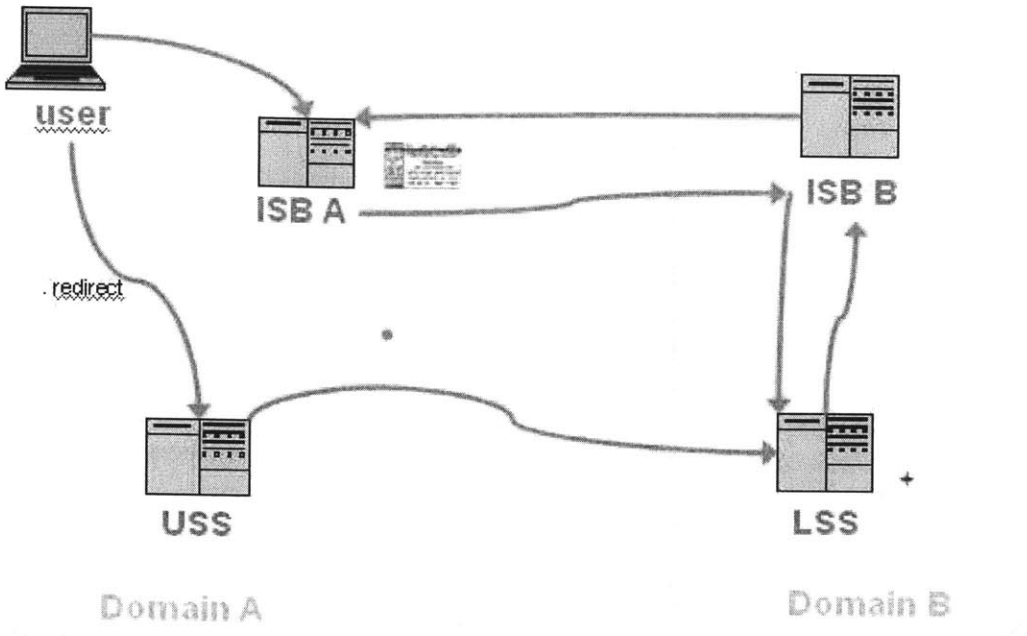

Figure 4-5: The General Ticket Scheme

the ticket by retrieving the full ticket from the Interactive Service Broker in domain B (ISB B). ISB B must go back to ISB A to retrieve the full ticket and then forward it to the LSS. This sequence of messages is illustrated in Figure 4-5.

Chapter 5

Security Ticket Token Services (STS) design

5.1 Design goal

In the interactive architecture we have designed an integrated approach to system management based on the following three principles:

- All administrators should authenticate on the ISB, and if they must then be redirected to another server, they should not have to authenticate again.
- All administrative functions that relate to iLab functionality should be carried out as far as possible by reusable iLab modules.
- The administrative interfaces of the iLab processes should be integrated in such a way that administrators should never have to enter the same data twice. [7]

Based on the principles of the iLab architecture, once the user has authenticated himself, he should not have to do so again to perform any allowed action or to access any allowed resource within his iLab environment. This is possible because the ISB will forward the user's credentials whenever the user wants to use an iLab service or application. The following design requirements beyond this core functionality also need to be satisfied.

- The design must bridge web applications and web services by providing coordinated authentication for both technologies. For example, it should enable an authentication performed by a login to a web application to authenticate subsequent web service calls.

- The design must support authentications and authorizations that span internet domains and academic communities.

- The design must be able to express transient, user specific rights, e.g., this student can use your lab server for the next 60 minutes.

- The design can be implemented by WS-Security

Based on the principles and requirements of interactive iLab architecture, the brokered design patterns that illustrate authentication using the Security Token Service carried out by Message layer security was adopted as the solution to the cross domain authentications of web service call. The reasons for this design decision are :

1. Securing the message using message layer security was adopted instead of using transport layer security because message layer security has several advantages such as increased flexibility, including support for auditing and supporting for multiple protocols. In addition, in the iLab interactive architecture, a message needs to go through multiple points to reach its destination. In transport layer security, each intermediate point must forward the message over a new computationally expensive SSL connection, and the original message from the client is not cryptographically protected on each intermediary.

2. The brokered authentication pattern is used because the interactive iLab architecture is a multi-domain, multi-server environment, which requires centralized authentication management in each domain.

3. X.509 requires support for a Public Key infrastructure (PKI), which can be expensive to set up and maintain [8]. Kerberos Token Security requires an identity provider that supports the Kerberos protocol, such as Active Directory. The

Kerberos protocol is used to authenticate clients within a domain. Cross-domain trusts can be established but are typically limited with in an organization[10]. The iLab project is a world wide distributed project, so Kerberos token is not a solution. A custom Security Token Service (STS) can provide authentication across organization boundaries easily. Considering these reasons, the custom security token service was chosen instead of X.509 and Kerberos Token.

5.2 Design of STS

The iLab architecture is designed to encourage the sharing of labs across institutions and campuses, but it is also a principle of the architecture that the administration of users should be as local as possible. All organizations should ideally run their own ISB, and each such ISB defines its own domain and authentication realm. In the General Ticket scheme used currently to deal with authentication cross multiple domains, the authentication information has to be redeemed "backward" from the ISB by the service provider.

In contrast, the authentication information is sent "forward" to the service provider in the STS scheme. Here, I use the same scenario as the example as in Section 4.4. Because the LSS only trusts ISB B, in order to obtain a security token to be authenticated by a LSS, the USS must first get a security token for the STS in domain B (STS B) from the STS in domain A (STS A) after being authenticated by SB A. Then, with the security token for STS B, the USS can get a security token for the LSS from STS B. With the verifiable security token for the LSS, the web method call, "confirm reservation", from the USS in domain A can be authenticated by the LSS in domain B.

The security token is the encryption of the symmetric key signed by the STS using the public key of the target service. The symmetric key generated by the STS is for the client and service to encrypt subsequent messages between them. The copy of the symmetric key is returned to the client together with the security token. The target service decrypts the security token with its private key and validates the token by

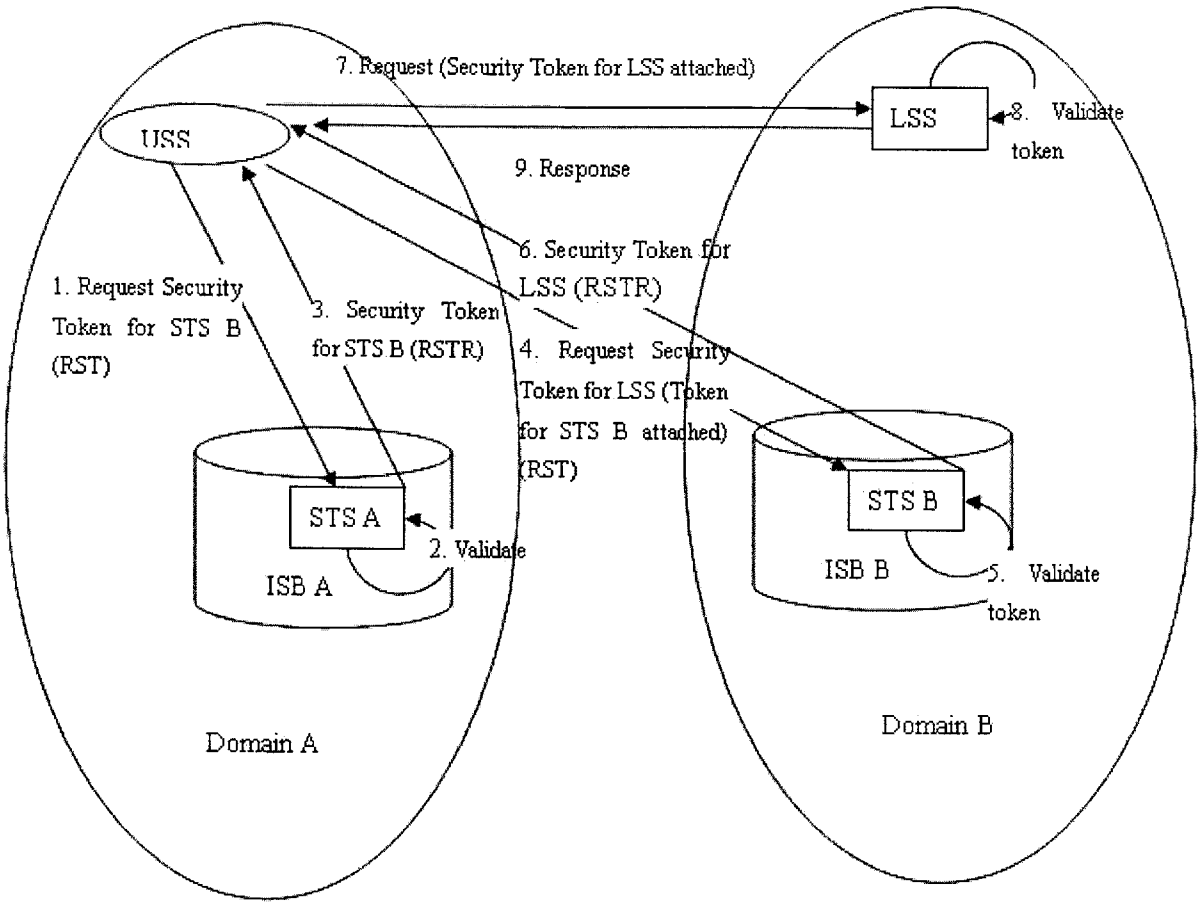

Figure 5-1: STS work flow

verifying the STS's signature. In this way, the target service can be sure that the call from the client is authenticated by the trusted broker.

Here, the STS scheme is demonstrated in detail through the scheduling scenario. Now let us describe the workflow in the scheduling scenario.

1. The user in domain A logs on the ISB A and requests to go to the USS to make reservation for executing the experiment on the Lab Server in domain B.

   During this procedure, the ISB A authenticates the user by verifying the user's credential. After the user is authenticated by the ISB A, the ISB A starts a scheduling session and redirects the user to the USS in the domain A. The security during the redirection from the web application of ISB A to the web

application of USS is ensured by the General Ticket scheme.

2. After user makes a reservation on the Lab Server in Domain B, the USS checks whether a valid (i.e. unexpired) security token for the LSS which manages the time blocks for the Lab Server is cached. If it is cached, the USS can call the web method "confirm reservation" on the LSS in domain B directly using the cached security token to authenticate itself.

3. If the USS can not find the cached security token for the LSS, the USS requests a security token from the STS A.

   During this procedure, the USS first needs to initialize a Request Security Token (RST) message for the STS:

   The global unique identity (GUID) of the USS is attached to the RST message.

   The USS sends the RST message to the STS A.

   The communication between the USS and STS A is secured by SSL.

4. The STS A processes the RST sent by the USS and issues a security token in response.

   During this procedure, the processes are:

   The STS A validates the credential of the USS, the GUID, attached to the RST.

   If the credential is valid, the STS A initializes a security token that will be returned to the USS.

   The STS A generates a symmetric key, which can be used for the confidential communication between the USS and the LSS, as a claim in the security token.

   The STS A also includes a second copy of the symmetric key in a proof token.

   The STS A signs the security token with its X.509 certificate private key to provide the data integrity for claims within the token and to provide proof to the USS that the token was issued by the STS A.

The security token is encrypted with the STS B's X.509 certificate public key, which can ensure that the security token can not be read if it is intercepted by others.

ThenSTS A returns both the security token and proof token back to the client in the Request Security Token Response (RSTR).

5. The USS initializes and sends a request for the security token for the LSS to the STS B.

   During this procedure, the processes are:

   The security token issued by the STS A is attached to the request message.

   The USS generates a signing key from the symmetric key contained in the proof token and signs the request message with it.

   The USS encrypts the sensitive part of the request message with the symmetric key contained in the proof token and sends the request message to the STS B.

6. The STS B processes the request from the USS and sends back the security token for the LSS.

   During this procedure, the processes are:

   The STS B decrypts the security token with its X.509 private key and uses the symmetric key contained in the security token to decrypt the request message. The STS B computes the derived signing key from the symmetric key in the security token and verifies the message signature. This provides data origin authentication and integrity assurance. The STS B checks whether the security token is signed by the STS A which is one of its trust token issuers. The STS B trusts STS A. If the security token is validated from STS A which means the USS has been authenticated by the STS A, the STS B will send the security token for the LSS to the USS. The procedure for generating and sending the security token for the LSS is similar to step 4.

7. After the USS receives the security token for the LSS, the USS caches the security token for the LSS.

8. The USS initializes and sends a "confirm reservation" request to the LSS. The procedure is similar to step 5.

9. The LSS authenticate the USS by validating the security token attached in the SOAP message. If the security token is validated as coming from STS B which the LSS trusts, the reservation can be confirmed. The procedure of validating the security token for the LSS is similar to step 6. Now the cross-domain web service process is completed.

As for executing experiment scenario, the only difference is that the attribute assertions will be added to the security token in order to let the lab server know the time period during which the user is authorized to execute the experiment.

5.3 Comparison between current solution and STS

The benefits of using the Brokered Authentication: Security Token Service (STS) pattern include the following:

- This pattern provides a flexible solution for exchanging one type of security token for another to accomplish a variety of goals in a Web service environment, such as authentication, authorization, and exchanging session keys.

- The solution is not dependent on any one mechanism, such as the Kerberos protocol or X.509, to secure messages. This makes it easier to enable different authentication protocols to interoperate by adding a level of abstraction on top of existing protocols.[3]

In addition, compared with the current iLab General Ticketing scheme, the advantages of STS scheme include:

- The security token can be cached by the STS. In this way, the overhead of creating the security tokens can be avoided when talking to the same service more than once. The latency of the cross-domain web service call is mainly caused by the cross-domain communication due to the authentication in the interactive iLab architecture. The STS scheme can reduce the latency greatly because the number of the cross-domain communication is only half of that in the General Ticket scheme.

- Confidential communication between client and service can be ensured by the symmetric key assigned to the both sides by the STS.

- The service does not need to interact with the service broker to perform the verification.

- In the General Ticket scheme, the communication between several points is done via SSL. The STS scheme can provide end-to-end at the message layer security which can avoid the unacceptable application response times due to the SSL and the possibility that the message being tampered with on each intermediary.

However, the STS can not totally take the place of General Ticketing in the interactive iLab architecture because the STS is implemented based on WS-Security, which can only ensure the Client to Web services security. It does not provide a solution for a single sign on (SSO) for multiple web applications, which is very common scenario in interactive iLab architecture. So we only can use WS-Security to optimize part of the General Ticketing scheme, such as the messages between the USS and LSS. Currently, ADFS in Windows Server 2003 R2 uses SAML 1.1 tokens and the WS-Federation passive client profile specification to enable SSO scenarios with web applications.[4, 12, 2]Further support for active client scenarios (such as SSO support for web services) is under development.

Chapter 6

Conclusion

In this thesis, a complete solution is provided for solving the reservation issue in the interactive iLab project. The scheduling servers and services are implemented to enable students from different campuses to reserve time on experimental apparatus. Since the user side and lab side require different scheduling functionalities, user sides scheduling server (USS) and lab side scheduling server (LSS) are introduced in the iLab Interactive Services to mutually manage reservations from different students to execute different experiments. In the first part of this thesis, the philosophy of the scheduling services design and the implementation are illustrated. It is proved that the two layer scheduling servers can satisfy the flexibility and scalability requirement of the interactive iLab architecture.

I also present a Security Token Service (STS) scheme for using WS-Security to optimize the cross-domain authentication in the iLab interactive architecture. The scheme uses the brokered authentication with a security token issued by the STS. The STS is trusted by the web applications and web services in the iLab interactive architecture to provide interoperable security tokens. How to introduce the STS scheme to the interactive iLab workflow is represented in detail. By comparing the STS scheme and the current General Ticketing scheme, it can be concluded that the STS scheme can optimize the General Ticketing scheme used currently in the iLab architecture in term of the efficiency and security of the cross-domain authentication.

Bibliography

[1] *Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Service Enhancements(WSE 3.0).* March 2006.

[2] Siddharth Bajaj, Giovanni Della-Libera, and Brendan Dixon. *Web Services Federation Language (WS-Federation)*, July 2003.

[3] David Chappell. *WS-SECURITY: New Technologies Help You Make Your Web Services More Secure.*

[4] Microsoft Corporation. *Active Directory Federation Services: A Path to Federated Identity and Access Management*, September 2004.

[5] T. Dierks, Certicom, C. Allen, and Certicom. The tls protocol, version 1.0. jan 1999.

[6] W3C Working Group. *Web Services Architecture*, February 2004.

[7] Jud Harward and Jedidiah Northridge. ilab internactive ticketing and integrated management-overview. Technical report, MIT, April 2005.

[8] Housley. Internet x.509 public key infrastructure certificate and crl profilem. Technical report, RFC 2459, jan 1999.

[9] http://en.wikipedia.org/wiki/Web application. Web-application.

[10] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Journal*, 39(9):33–38, July 1994.

[11] Jedidiah Northridge and Jud Harward. General ticketing. Technical report, MIT, 2002.

[12] OASIS. *SAML Executive Overview*, April 2005.

[13] W3C Recommendation. Soap version 1.2 part 0: Primer 24. Technical report, W3C, 2003.

[14] W3C Recommendation. Web services description language (wsdl) version 2.0 part 0: Primer. Technical report, W3C, June 2007.