iLab Interactive Ticketing and Integrated Management — Overview

Author: *Jud Harward, Jedidiah Northridge, Rabih Zbib, Loai Naamani, Imad Jabbour, Ting Ting Mao, Philip Bailey*

Last Revision: ~~*11 May 2006*~~*14 January 2010*

Revised by: ~~*Jud Harward*~~*Philip Bailey*

Introduction

This document falls into two parts. The first explains in an introductory way the distributed authorization scheme that guarantees that individual users can perform the actions appropriate to their role in the iLab environment after they sign on. The second part discusses the integrated management of the multiple applications and services that make up the interactive architecture. Clearly, the second depends heavily on the mechanisms of the first. Both general readers and developers will find this document useful, but developers will then need to read the more detailed documentation listed below. This document assumes that readers are already familiar with the paper *iLab*~~*b Interactive*~~ *Services — Overview* and the following web technologies: SOAP, web services, and SSL.

Related Documents:

There are three series of iLab ~~interactive~~ architecture documents. The *Overview* series provides a high-level introductory view of the architecture. It is aimed at the general technical reader, but it also provides a starting point for developers encountering the architecture for the first time. This document is the second of the Overview series. The *Operation* series describes individual web services and applications within the interactive architecture at a level of detail that developers will require. The *Specification* series presents the application programming interfaces and data type descriptions for each of

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

the interactive web services. It is intended to serve as a programming reference for developers and is derived from comments embedded in the source code.

1. *iLab ~~Interactive~~ Services — Overview*
2. *iLab Interactive Ticketing and Integrated Management — Operation~~Operation~~*
3. *iLab Interactive Ticketing and Integrated Management — Specification*
4. *iLab Experiment Storage Service —Operation ~~Operation~~*
5. *iLab Experiment Storage Service — Specification*
6. *iLab Interactive Scheduling — Operation~~Operation~~*
7. *iLab Interactive Scheduling Services — Specification*

Ticketing

The Problem

The overview document *iLab ~~Interactive~~ Services — Overview* has a section titled *Authentication and Authorization*, which introduces the concept of the iLab credentials known as *tickets*. The iLab ~~interactive~~ architecture assumes that the user's identity is confirmed by the ~~Interactive~~ iLab Service Broker (ISB) at the start of every session. The ISB may accomplish this using a simple user name and password scheme or may employ more sophisticated mechanisms such as user certificates. It is a principle of the iLab architecture, however, 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. This is possible because the ISB will forward the user's credentials whenever the user wants to use an iLab service or application. The purpose of tickets is thus to allow centralized authentication and centrally managed authorization within the iLab environment.

Several other design requirements beyond this core functionality have shaped the design of iLab tickets:

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

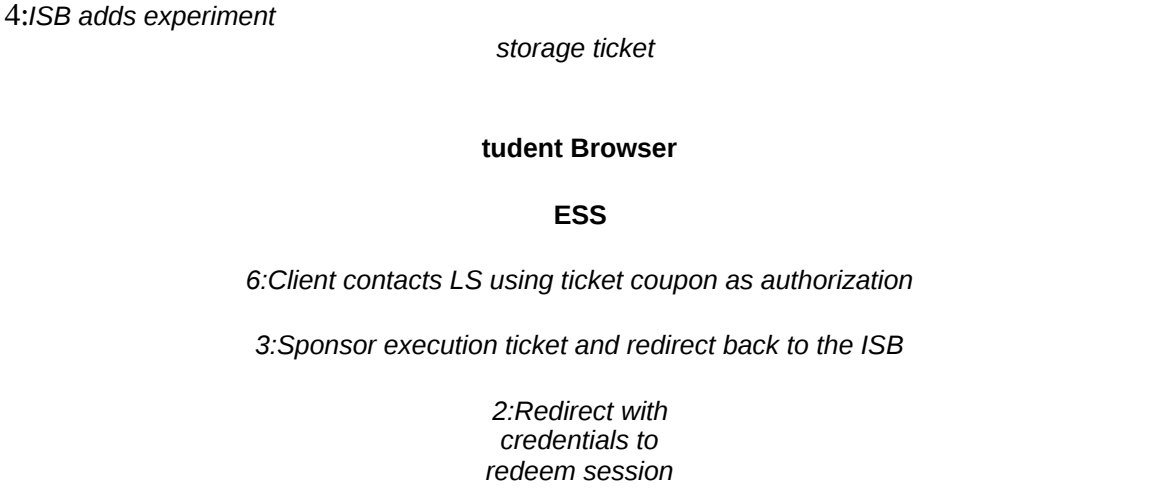
2:Redirect with credentials to redeem session

- The design must bridge web applications and web services by providing coordinated authorization for both technologies. For example, it should enable an authentication performed by a login to a web application to inform authorizations for subsequent web service calls.
- The design must support authentications and authorizations that span internet domains and academic communities.
- The ticket mechanism must be secure, efficient, and both OS and vendor independent.
- 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 must be modular and similar enough in structure and function to WS-Security so that it can be eventually replaced by the latter evolving technology with minimal disturbance to the remaining code of the interactive architecture.

The Strategy

The strategy for ticketing starts by considering how the different design requirements shape the concept of credentials. Some form of the credentials must accompany privileged web service calls and web application page accesses. In the case of a web service call, the obvious place to put the credentials is in the header of the SOAP request, following the strategy of WS-Security. During access to a privileged web page, a similar set of credentials can be embedded in the query portion of the page's URL if that part of the HTTP request is not otherwise used. Since the credentials must be presented on every privileged web service or page access, the credentials should be as efficient as possible. Small and easy to parse is the main criterion of efficiency.

If the credentials themselves accompany the web service call or web page request, then what is to prevent the presenter who is requesting access from forging the credentials? In WS-Security, the solution is to have a trusted server sign the credentials. In iLab ticketing we have taken a different approach, at least initially.



A little terminology will help here. 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 ticket holder can be one of the distributed interactive services or web applications like 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 server that creates the credentials to authorize the operation is known as the *ticket issuer*. In iLab ticketing, the ticket issuer is always an ~~Interactive~~ iLab Service Broker (ISB). Often tickets are created in response to an administrative action taken on an ISB, but on other occasions, any iLab service (ProcessAgent) may ~~process agent~~ request~~s~~ that the ISB create the ticket on behalf of itself or another process. We call the process agent that requests the creation of the ticket the *ticket sponsor*. If the ticket is created in response to an internal action on the ISB, the ISB is registered as the ticket sponsor.

Now let us return to the problem of insuring that a ticket holder can not forge the credentials he presents. In the iLab ticketing scheme, the ticket holder does not present his real credentials (tickets) to the redeemer, but rather a receipt that the redeemer can use to retrieve the true tickets from the ticket issuer. We call the receipt a *ticket coupon*. It is a small XML-based data structure, whose format we discuss below. The redeemer must retrieve the corresponding *ticket* to evaluate the ticket holder's level of authorization. The ticket is another XML-based data structure that contains a number of standard fields, for example, the identity of the ticket redeemer and the duration of the ticket. The ticket also contains a special XML-encoded field, the *payload*, whose structure varies according to the type of authorization that the ticket expresses. In the case of a ticket authorizing a Lab Client to execute an experiment on a Lab Server, the payload contains XML attributes specifying the group to which the user executing the experiment belongs, the period reserved for the experiment's execution, and other context-related information.

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

Note that although we call the process that initiates the privileged request, “the ticket holder”, the process never actually *holds* a copy of the ticket. The ticket holder just holds the ticket coupon, and the only process that can use the coupon to retrieve the associated tickets is the ticket redeemer.

The ticket issuer clearly plays a central role. It 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. Remember that ticket issuers are currently ~~Interactive~~ iLab Service Brokers. This makes it easier for them to associate tickets with authenticated users. It also leads to a natural scoping of tickets and processes. ISBs issue tickets and ticket coupons for processes and resources that they manage. The web services and web applications that control these resources are known as process agents.

Thus, the processes that form the iLab interactive environment fall into three categories. There are the interactive service brokers (ISBs), the process agents that use authorizations provided by a particular ISB to contribute to tasks, and the transitory client processes that users employ to execute experiments. Process agents are always long running servers hosting web services or web applications. The following all belong to the category of process agents (for more information, please consult the document *iLab ~~Interactive~~ Services — Overview*):

- User-side Scheduling Server (USS);
- Lab-side Scheduling Server (LSS);
- a lab server (LS);
- Experiment Storage Service (ESS)

Each ISB forms an iLab domain that includes the service broker and all those process agents that rely on the ISB’s tickets for authorization. A process agent can only belong to one domain, and each domain contains exactly one ISB. Clients, however, can access lab

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

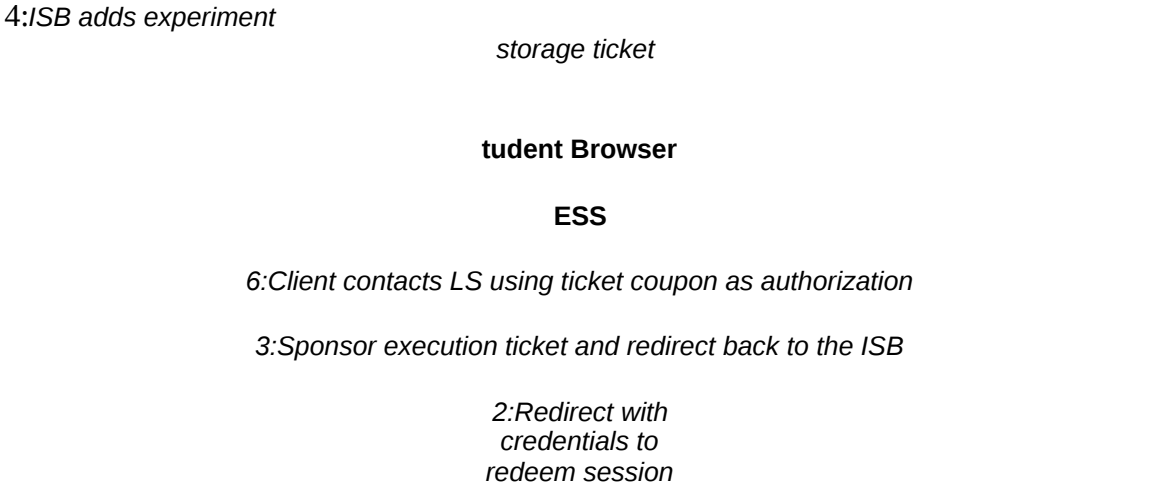
servers (a type of process agent) in multiple domains. We will see how this is done later in this paper.

The Mechanism

Tickets and Ticket Collections

Performing an experiment is a multistage operation that usually involves multiple authorizations to several process agents communicating among each other. As an example, let's describe one possible workflow without trying to be slavishly complete.

1. The user authenticates through the ISB and then indicates that she wants to confirm a previously made reservation and proceed to execute the experiment.
2. The ISB redirects her to the USS with authorization to look at and confirm her reservation.
3. The USS allows her to redeem a previously made reservation. The USS sponsors the creation of an allow experiment~~ecution~~ ticket on the ISB for her to use the reserved lab server (LS) for her scheduled time and then redirects her back to the ISB.
4. The user requests the launch of the experiment.
5. The ISB creates a new ticket collection. ~~The ISB~~ adds ~~a~~ tickets authorizing the storage of experiment data on the local ESS, and creates an execute experiment ticket.~~:~~
6. ISB launches the client with a coupon that corresponds to ~~both these authorizations~~the execution collection..
7. The client makes a web services connection to the lab server using the supplied ticket coupon in the SOAP header of the request. The lab server uses the coupon to retrieve the execution ticket that specifies the user's reservation and level of access. If it accepts the authorization, the lab server will now open a control channel to permit the client and lab server to communicate directly.
8. The lab server asks to store *experiment records* on the ESS on the user's behalf. Each storage request is accompanied by the same ticket coupon that the client originally presented to the lab server. The ESS uses the coupon to retrieve the



storage ticket created for this experiment session by the ISB once, and then caches the retrieved ticket.

The first thing to notice in this walkthrough is that executing an experiment is of little use if you can not store the results. So the ISB creates a storage ticket to allow the lab server to store records on the user's local storage service (ESS). This ticket is associated with the execution ticket when the ISB places it in the same *ticket collection* as the original execution ticket. Ticket coupons actually identify *ticket collections* rather than individual *tickets.*

When a process agent wants to retrieve a ticket from a ticket collection to check an authorization (using the **`RedeemTicket`** web service call), it supplies two arguments:

1. the **`Coupon`**, which is a compound object, and
2. the requested *ticket type*, which is a **`String`**.

The ticket type varies according to the type of operation that the user is trying to perform. The ISB uses the ticket type to distinguish between different tickets in the same collection. Although it is not a formal argument, the ISB can also determine the identity of the process agent trying to retrieve the ticket. It will only honor the request to retrieve the ticket if the requesting process agent was specified as the ticket redeemer when the ticket was created and inserted into the ticket collection. This implies that a ticket collection can contain only one ticket of a given type for a particular ticket redeemer.

The redeemer knows the type of ticket to request from the type of operation invoked or resource accessed by the web service call. The same logic applies if the ticket authorizes an action through a web application like confirming a reservation on the USS. Thus if a user is redirected to the experiment scheduling page of the USS, the USS will contact the ISB to retrieve a **`Schedule_Session`** ticket. It will use the coupon that was included as the query portion of the redirect URL to identify the ticket collection that the ISB should search for the scheduling ticket. As another example, once the experiment is underway, the lab server may contact the Experiment Storage Service (ESS) with a web service request to add a record to the experiment log. The coupon in the SOAP header of

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

this request must lead to a ticket collection that contains a **`Store_Records`** ticket with the ESS specified as the ticket redeemer. (See Appendix 1 for examples of ticket coupons in requests.)

This walkthrough also illustrates that two processes can use the same ticket coupon to authorize an operation. In this example, both the lab client and the lab server act as ticket holders of the same ticket collection. The lab client is supplied the ticket coupon when it is launched. It then passes the ticket coupon on to the lab server when it makes initial contact. In certain experiments the lab client may want to contribute to the experiment record as well as the lab server. Both processes can ask to store data associated with the experiment by presenting the same coupon to the ESS. The storage ticket in the collection specifies the ESS as the ticket redeemer, but it does not specify which processes are allowed to present the coupon for service. In fact, a ticket does not specify a ticket holder. This means that mere possession of a ticket coupon gives authorization, and therefore, the ticket coupon must be kept secure. By default, all web service calls and web page accesses that use coupons should use SSL to protect the coupon as it is transmitted over the network.

If a ticket redeemer has already seen a particular ticket coupon and retrieved the corresponding ticket to authorize the user's request, it may choose to cache the ticket so that it doesn't need to retrieve the ticket again. This decision is left up to the ticket redeemer. Tickets have expiration times, which a ticket redeemer should respect. In addition, a ticket sponsor can request the ISB to cancel a ticket. In that case, the ISB will make a **`CancelTicket`** web service call to the ticket redeemer to inform it that the ticket is no longer valid. Any attempt to retrieve a cancelled ticket will fail.

We have been treating the ticket coupon as a black box, but we can make the concept more concrete by taking a look at the ticket coupon structure:

Figure :

`Coupon`

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

Members:

`long couponId`

An ID unique in the domain of the issuer that identifies the associated ticket collection

~~`long`~~ `string` `passkey`

A security passkey that must always accompany the couponId to retrieve tickets; note it is not guaranteed to be unique, just hard to guess

`string issuerGuid`

A GUID in **`string`** *form that identifies the Service Broker that issued the associated ticket collection*

The **`issuerGuid`** identifies the ISB that issued the ticket, and, therefore, the ultimate target of any **`RedeemTicket`** calls for this coupon. We haven't explained how a process agent knows a given ISB's contact information yet, e.g., web service address. We will discuss that in a later section. The **`couponId`** identifies the coupon and the associated ticket collection on the issuer so it must be unique on the issuer. The **`passkey`** is a random ~~**`long`**~~ **`string`** that is added to the coupon as a security precaution since the **`couponId`** may be easy to guess. (The simplest implementation of **`couponId`** assigns successive integers to coupons as they are created in the backend database.) Appendix 1 shows ticket coupons in action in two contexts, first in the SOAP header of a web service request and second as the query portion of a URL authorizing a web application page access.

Here for concreteness is the corresponding structure for the **`Ticket`** itself:

`Ticket`

Members:

`long ticketId`

An ID unique in the domain of the issuer that identifies this ticket

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

`string type`
A string drawn from a fixed set that specifies the type of operation authorized by this ticket

`string sponsorGuid`
A GUID in **`string`** *form that identifies the sponsor that created this ticket*

`string redeemerGuid`
A GUID in **`string`** *form that identifies the service provider for which this ticket authorizes an operation*

`long creationTime`
Specifies when the sponsor created this ticket

`long duration`
Specifies when the authorization specified by this ticket expires; the value **`-1`** *indicates that the ticket has no expiration and will remain valid until it is cancelled, duration is in seconds.*

`string payload`
An XML-encoded string that specifies the details of the authorization carried by the ticket

`boolean isCancelled`
`true` *if the ticket has been cancelled*

Two details are worth noting. First, the **`payload`** field carries the content that is specific to each type of ticket. The ticket redeemer must be able to parse this XML-encoded field to determine what the ticket authorizes. Second, tickets that authorize particular users to perform an action like executing an experiment almost always have a limited duration. Other tickets, however, that authorize how one process agent can access another tend to be valid until revoked. We call these *permanent tickets*.

Creating Tickets

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

In the experiment execution walkthrough in the previous section, we described how tickets to execute an experiment and store the results are created and assembled. We glossed over, however, how the USS was authorized to add an execution ticket to the user's ticket collection.

Process agents are registered in their interactive service broker domain via a procedure known as *installing domain credentials*. This will be discussed in the next section, but the end result of the procedure is that the ISB and the process agent each acquire a special ticket coupon that they can use to authorize subsequent web service calls that concern ticketing. (See Appendix 1, Example 3.) These reciprocal ticket coupons allow the process agent to call the **`TicketIssuer`** interface on the ISB and the ISB to call the **`Process–Agent`** interface on the process agent. These tickets are obviously permanent, that is, valid until revoked.

Since the ISB holds the persistent versions of tickets and their enclosing collections, it is easy for an ISB to add tickets using an internal API. If other process agents acting as ticket sponsors want to contribute tickets to a ticket collection, however, they must request to do so using two web service methods in the ISB's **`TicketIssuer`** interface. **`AddTicket`** requests the ISB to write a new ticket with a payload and a redeemer specified by the caller and then adds it to a preexisting ticket collection. **`CreateTicket`** is similar, but it assumes the created ticket is the first of a new collection.

Creating a ticket amounts to giving a user or a process agent authorization to perform an action or access a resource so a ticket sponsor's request to create a ticket (possibly for itself!) must itself be authorized. The ISB checks all such requests against an internal table which lists the rights of process agents to create particular types of tickets to be redeemed on particular service providers. These rights are entered into the table during the execution of the business logic behind the ISB's administrative pages. These pages

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

can only be accessed by privileged users. We will return to this topic when we discuss *integrated management.*

Startup

Our discussion of ticketing has assumed that the ISB and its domain are already configured. Now let's turn to examine how an ISB and its associated servers establish trust and communication. There are two parts to the problem. The first concerns discovery. How do the process agents and ISB discover each other, learn each other's roles, and exchange the information necessary to communicate, e.g., the URL of a process agent's web service. The second part deals with the establishment of trust and secure communication. To become a part of a domain, a process agent must learn to accept tickets from the domain's ISB, and the ISB must learn to recognize the new process agent. This must happen securely so that no rogue system can masquerade either as an ISB or a process agent seeking membership in the domain. This section discusses the second problem. We delay discussion of the first to the later section on integrated management.

Once trust has been established, communication between process agents is authenticated by the ticketing scheme described above. Communication between a process agent and the ISB is authenticated by a special pair of ticket coupons called *domain credentials* that allow the ISB and the process agent to recognize each other. For example, a process agent uses its half of the domain credential pair in the SOAP header of a **`RedeemTicket`** call when it wants to check the authorization for a request made by another process agent (see Appendix 1, Example 3). Thus, domain credentials must be established before a process agent can accept any normal tickets to authorize a request.

The ISB accomplishes the installation of domain credentials through a single web service call, named appropriately **`InstallDomainCredentials`**:

`InstallDomainCredentials`

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

Purpose: Used by an Interactive Service Broker to install the initial tickets coupons (Service Broker and Service Provider ticket collection identifiers) on a service provider to make it a member of a domain. The call is authenticated by an initial passkey communicated out of band to the Service Broker administrator by the service provider administrator. This passkey is sent in the SOAP header of this method call, and is never used again.

Arguments:

```
ProcessAgent broker  /* ISB's info */
Coupon inIdentCoupon
Coupon outIdentCoupon
```

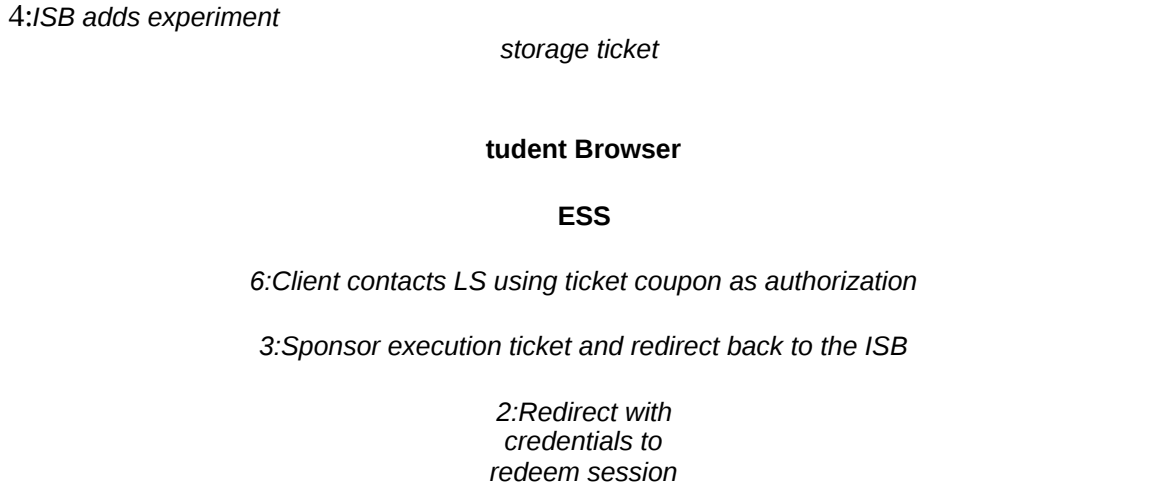
Returns:

```
ProcessAgent provider   /* provider's info */
```

The above summary is taken from the API documentation, but we will expand on it here. The call requires two pieces of information usually entered on one of the ISB's administration pages by the process agent's administrator:

1. The process agent's web service address.
2. A **`string`** passkey that the service provider will recognize.

The ISB then makes the **`InstallDomainCredentials`** web service call on the process agent. The SOAP header includes a simplified **`Coupon`** (see Fig. 1). The only part of the coupon that is required is the passkey field, which contains the passkey previously supplied by the process agent administrator. Its role is to assure the process agent that the call comes from the ISB whose domain the process agent wishes to enter. The passkey is used only once. The web service call contains three arguments. The first is a data structure, **`ProcessAgent`**, that contains the ISB's basic contact information. It is detailed in Appendix 2. The second argument, **`inIdentCoupon`**, is a **`Coupon`** that the ISB will use to identify itself in the SOAP header of all future web service calls to the target process agent (the process agent seeking to enter the domain). Functionally, it



replaces the initial passkey. The third argument, **`outIdentCoupon`**, is the corresponding **`Coupon`** that the process agent will use in the SOAP header of its own calls back to the ISB. These two ticket coupons are special in that they will not normally be used to retrieve authorization tickets. The combination of the two fields, **`couponId`** and **`passkey`**, are considered sufficient to authenticate the sender. The third field, **`issuerGuid`**, is included for completeness and identifies the domain, but since a process agent never belongs to more than one domain, it is not logically necessary.

The recipient of the call returns its own **`ProcessAgent`** data structure, which gives the ISB all the contact information it will need to integrate the process agent into the domain.

Cross-Domain Ticketing

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 ticket realm. So if a student at MIT authenticates through his on-campus ISB and wants to use a lab hosted at a university in Singapore that has its own ISB, how are the student's credentials checked when his lab client presents a ticket coupon while requesting to execute an experiment?

The lab server in Singapore will call **`RedeemTicket`** on its own ISB as usual, but the third part of the **`Coupon`** that is supplied as an argument to the call is the ID (GUID) of the issuing ISB, MIT's ISB. Singapore's ISB will recognize that it was not the issuer of the **`Coupon`** and, therefore, that it can not immediately return the ticket. Instead it forwards the request to the issuing ISB. There is a procedure almost identical to that by which an ISB incorporates a process agent into its domain that allows an ISB to recognize a foreign ISB and its domain as a *partner domain*. This relationship is established when one ISB calls **`InstallDomainCredentials`** on a second ISB. As a result the two ISBs will recognize each other's identifying GUIDs when they appear as the third part of a **`Coupon`**, and the ISBs will possess the proper credentials to exchange

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

ticket information. Note that this implies that ISBs are a special type of process agent and they support all the methods that process agents support.

A ticket written in one domain to access a process agent in a second domain will still need to specify the redeemer in the partner domain. This means that the ticket sponsor in the originating domain must know the particulars of the ticket redeemer in the partner domain. It also leads to the one exception to the rule that an ISB will only retrieve a ticket for the redeemer of the ticket. In the example above the lab server must apply to retrieve the execution ticket from its own ISB. But the local ISB did not issue the ticket so it must issue a second **`RedeemTicket`** call directed to the issuing ISB. This ISB will honor the request even though the partner ISB is not the redeemer. In effect, all ISBs trust each other once they have exchanged domain credentials and will honor a ticket retrieval request without further checking. This is not as glaring a security hole as might first appear, because ticket retrieval requests must also be accompanied by the coupon ID and passkey. The requirement for these two arguments makes it impossible to “fish” for tickets as long as ticket coupons are protected by SSL during transmission.

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with
credentials to
redeem session

Integrated Management

The Problem

As we have observed a fully developed iLab domain that supports users and publishes at least one lab will include at least five server processes not counting backend database services:

1. The domain ~~Interactive~~ Interactive Service Broker (ISB)
2. a User-side Scheduling Server (USS);
3. a Lab-side Scheduling Server (LSS);
4. a lab server (LS);
5. an Experiment Storage Service (ESS)

Note that these processes don't need to run on dedicated servers. Typically lab servers will be interfaced to special purpose hardware and thus tend to run standalone, but this set of processes will often be configured to run on three or four servers instead of five.

The complexity of the interaction between iLab processes only increases when one considers collaboration between domains, e.g., a user in one domain making a reservation through her own USS to execute an experiment hosted on a lab server in another domain. Our experience in developing the far simpler batched architecture has taught us that if each iLab process has its own independent administrative interface with its own authentication system, creation and management of the iLab environment as a whole becomes cumbersome and fragile. Instead, in the interactive architecture we have designed an integrated approach to system management based on the following three principles:

1. All administrators should authenticate on the ISB, and if they must then be redirected to another server, they should not have to authenticate again. Instead the same ticketing scheme described above should provide the credentials and authorization to perform the required tasks on any iLab server.

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

2. All administrative functions that relate to iLab functionality should be carried out as far as possible by reusable iLab modules. Reference implementations are provided for the ProcessAgent web services, ISB, ESS, LSS, and USS. The case of lab servers is more complicated because each lab server provides custom functionality for the lab and experiments it supports. The lab server may need to be implemented in a particular operating system due to restrictions imposed by the custom lab equipment to which it is interfaced. The iLab interactive architecture does provide, however, standard lab server modules that interface to the iLab services as well as a complete sample interactive lab server.
3. 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 and that complicated textual data such as GUIDs never need to be hand copied.

The ESS is an example of an iLab service that really does not require an independent, administrative interface except for debugging purposes. The information about particular experiments is split between an administrative data model housed in the ISB's backend database and the records that document the course of the experiment run itself, which are stored in the ESS. For example, the identity of the user who executed the experiment and the group under whose authorization the experiment ran are stored as part of the administrative data model on the ISB so that the administrative fields can be used as indices for faster lookup. On the other hand the temperature, say, of the main reactor vessel in a chemistry experiment would be stored in time-stamped records as part of the experiment log in the ESS. The user's lab client can review the experiment log. The ISB can use web service calls to administer the log, e.g., to delete an experiment or to export the log to a different format.

The USS and LSS, on the other hand, will always maintain their own administrative interfaces governing experiment scheduling. Their operation is vastly simplified, however, if they do not have to determine which labs an individual may access or administer. In the integrated administration of the iLab environment, this information is

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

incorporated into the ticket that the user or administrator presents to gain access to the scheduling page.

Mechanism

The earlier parts of this paper described the iLab ticketing mechanism in some detail. The current section will describe the internal ISB mechanisms that support both ticketing and integrated management. The next section will then present walkthroughs of single and cross domain use that illustrate integrated management.

Agents

The ISB supports the concept of *groups* of users. In fact, each user must select one of her groups as the current *effective group* at the start of each session. Authorizations are usually tied to groups of users, and a user's effective group defines her authorizations for the session. This is both a safety feature and a means by which a user can view the system from a number of roles, one for each group. If a user is a member of the superuser group as well as other groups, it would be dangerous to always have superuser permissions available. By forcing the choice of an effective group, the ISB requires an administrator to make a conscious decision to turn on superUser authorization, and while it is in effect, the user can act with corresponding care. The conscious use of roles can be useful when an administrator wants to check the effect of a particular system configuration. If a faculty member has used a privileged group to establish scheduling policy for the class he is teaching, he may want to assume the role of the student (choose the student group as the effective group for a new session) in order to check that scheduling is in fact behaving the way that he intended. Assuming the student role also allows the faculty member to demonstrate the use of iLab from the student's point of view during lecture.

Groups can contain other groups known as *subgroups*. This allows groups to be arranged in a hierarchy. For instance, the teacher of a course, Chem100, might want to define a group Chem100Students for the students taking her course. The teacher would also want

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

to be a member of this group so that she would have all the authorizations that her students had by default. She might also want to establish a subgroup of the original group called Chem100Staff for the teaching assistants for the course. Since Chem100Staff is a subgroup of Chem100Students, members of Chem100Staff are automatically considered to be members of Chem100Students, and her teaching assistants then automatically have all the permissions that apply to her students. Any further permissions assigned to Chem100Staff will only apply to her teaching assistants but not to her students, because members of a group are not automatically members of the subgroup. Finally the teacher might want to define a subgroup of Chem100Staff for herself and her colleague with whom she shares teaching duties. This would allow her to define certain privileges that would apply *only* to herself and her colleague and not to the teaching assistants or regular students. The teacher may want to be an explicit member of Chem100Students so that she can choose that group as her effective group for sessions in which she wants to demo or check that experiments will execute appropriately for users with only student permissions.

We have noted that permissions are usually associated with groups, but to be more accurate permissions in iLab are associated with *agents*. An agent is a single user, and group of users (possibly with subgroups), or a process agent acting as a pseudo-user. The last case is useful when the ISB wants to record a permission that applies to the process agent. The standard example of this is the permission that states that a particular process agent can sponsor a type of ticket to be redeemed on a second process agent. The next section describes the mechanism by which the ISB specifies these permissions.

Grants

When the ISB must determine if a particular action on behalf of a user is authorized, it performs the check in three stages. It first asks if the user's effective group is the superUser group. If so, there is no reason to check further because a superUser can do anything. Second, if the action involves access to a resource like an experiment log that has been created by a particular user, then the ISB checks if the user is the creator and, therefore, the owner, of the resource. If that is so, then the action is authorized. All other

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

cases are evaluated against authorizations (permissions) stored as instances of a special class called a *grant*.

In the batched architecture a grant consists of a triple:

1. an *agent* representing the user or group to which the authorization applied, e.g., group Chem100Students;
2. a *function* supplied as a string that specifies the operation or access that the grant authorizes, e.g., “UseLabServer”;
3. a *qualifier* that acts as a reference to the resource to which the agent is allowed to apply the function, e.g., a reference to a particular lab server, titrationlab.mit.edu.

The interactive architecture requires this triple to be extended by including an optional fourth element,

4. a *modifier* that specifies or restricts the application of the function to the target of the qualifier.

The main application for this fourth element is the grant that permits a process agent to sponsor a particular type of ticket to be redeemed on a second process agent. In this case, the agent is the ticket sponsor, the function is “SponsorTicket”, the qualifier refers to the ticket redeemer, and the modifier is the type of ticket that the agent is allowed to sponsor. For example, the grant to record that USSA can write ~~execution~~ allow experiment tickets for titrationlab.mit.edu would resemble:

{USSA_ID, “SponsorTicket”, qualifier for titrationlab.mit.edu,
“Allow~~Execute~~Experiment”}

As we observed in the previous section, if a grant applies to a group, it applies to all the members of that group and any subgroup. This introduces a type of “inheritance” into the iLab permission structure. There is a second inheritance dimension because qualifiers can also have a parent-child relationship. In fact, a qualifier can be the child of multiple

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

parent qualifiers. Any grant that applies to a qualifier also applies to all children of that qualifier.

One instance in which this is useful involves giving course staff permission to view the work of students taking a particular class. By default, only the creator of an experiment log can view it. One could, of course, create a grant to give course staff read permission for each experiment log created by students in a particular class, but this is inefficient and very expensive in terms of the administrative time to create the grants for each experiment. Instead, the standard iLab approach is to create a special qualifier (*collection qualifier*) that is the parent of all the qualifiers representing experiments executed by members of the class group. Then one can give the course staff read access to all the experiments performed by all class members using a single grant that uses the collection qualifier. Since all the qualifiers representing individual experiment logs are children of this collection qualifier, they all inherit the grant.

Resource Mapping

The ISB uses grants to check whether a particular operation or resource access is authorized, but it must often answer a related question before it can check authorizations. That is, in order to perform a composite operation like executing an experiment, what resources are required? The ISB uses a mechanism called *resource mapping* to answer this more fundamental question. The mechanism identifies such complex operations with a resource mapping ID. It then links the resource mapping ID with keys that identify types of resources and values that specify the sources of the typed resources.

For example, assume the execution of CircuitLab by students in EE204 requires the students to schedule their experiment time on USS2 and also requires that the experiment logs be stored on a particular ESS4. The combination of the class group, “EE204”, and CircuitLab is first assigned a resource mapping ID. Then a key/value pair is associated with this ID, in which the string “USS” specifies the key type and a GUID specifies USS2 as the value. A similar entry is created for the ESS that the lab server must use to store the experiment results.

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

Registration

Groups, grants, and resource mappings are all internal ISB mechanisms, but the registration mechanism that we will now discuss is the means by which the ISB informs one process agent about another process agent with which it will need to interact. All process agents support a **`Register`** call that takes a single argument, an array of **`Registration`** objects. The Register call will only be accepted from a process agent's own ISB. The **`Registration`** class members follow:

`Registration`

Members:

`string serviceProviderInfo`

An XML-encoded string that describes the services provided by the service provider described by this registration entry

`Coupon coupon`

An optional ticket coupon that can be used to redeem the described services.

`string address`

An optional ID or name that can be used by a non-local Service Broker to route the entry to the process agent that will redeem the described service.

Like the **`payload`** field in a **`Ticket`**, a **`Registration`** object's **`service-ProviderInfo`** field carries custom XML information that must be parsed by the target of the call. It describes the location and details of a web service provided by a second process agent that the target will require. The **`coupon`** field provides a **`Coupon`** that the target can use to call the service. Finally, the **`address`** can be thought of as routing information. It should never be required in intra-domain calls.

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

To understand the **`address`** field, consider the case of a USS and LSS in different domains. There is always exactly one LSS for lab servers that require scheduling so that the LSS can coordinate reservations. Best practices suggest that each university supply a USS for their own faculty and students. This allows the development of custom scheduling policies that are adjusted to the teacher's class schedule and pedagogical goals. Thus, the USS and LSS may easily be parts of separate domains. The lab side ISB will use a **`Register`** call to inform the USS of the LSS it should contact to confirm reservations. But the lab side ISB doesn't have the contact information for the USS so it can not make the **`Register`** call directly. Instead the lab side ISB calls **`Register`** on the student side ISB and uses the **`address`** field to tell the student side ISB to forward the call to its USS.

Management Examples

The examples that follow are more descriptive than proscriptive since they discuss implementation details of integrated management. We expect the implementation will evolve as we gain a better sense of best practices and the efficient use of workflow.

Single Domain Privileged Access Example

To demonstrate the workflow of integrated management in action, we will first describe the process by which a faculty member can adjust the rules by which her students sign up for lab time. It is a good example of the Privileged Access pattern in which a privileged user authenticates at the ISB, which then creates a ticket collection that gives the user special privileges on a domain process agent. It assumes that the domain servers have already been registered with the ISB.

Mary, a lecturer, starts her session by logging in to her ISB. Once she has authenticated herself, the ISB looks up all the user groups to which she belongs. To change signup policy for the students in her course EECS121 (group "EECS121"), Mary chooses the group "EECS121staff". A search through the grants available to Mary as a member of this group triggers the business logic of the ISB to offer the option of managing user

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

scheduling policy for EECS121. Mary chooses this option, and the ISB creates a new ticket collection with a MANAGE_USS_GROUP ticket. The ISB must lookup which USS (a domain may potentially possess more than one USS) is responsible for scheduling the users from group “EECS121” in the ISB’s resource mapping table. This USS becomes the ticket redeemer for the MANAGE_USS_GROUP ticket. The payload of this ticket specifies that the holder may set scheduling policy for members of group “EECS121”. The ISB then redirects Mary to the Group Policy page for group “EECS121” on the USS.

The USS business logic for the Group Policy page parses the ticket coupon from the query portion of the URL that directed Mary to the page. It retrieves the MANAGE_USS_GROUP ticket from the ISB and then puts it in a temporary ticket store in case Mary leaves the page and then returns. Once it has parsed the payload of the ticket and confirmed the group for which Mary can set scheduling policy, it displays the Group Policy page initialized with the labs accessible to the “EECS121” group. Mary then uses the forms and components on the page to adjust scheduling policy for the group. The changes become persistent in the USS’s backend database.

Single Domain Process Agent Registration Example

To demonstrate how process agents are registered in a domain, we will look at the steps by which a new lab server is registered in its home domain. This example illustrates the Install Domain Credentials pattern.

Bob works as the system manager for a new lab server at MIT. His supervisor, Dr. Zeal, originally developed this online lab with its own set of user accounts for his graduate students. He now wants to share the lab much more widely using his campus’s iLab infrastructure without increasing his own administrative burden. Bob starts by contacting Sarah, who administers the campus’s ISB, MISB. They settle on the name, SignalLab, for the lab server. Bob registers for an account on MISB. Sarah creates a new group, SignalLabAdmin, on MISB, and makes Bob a member of that. She also creates a grant that gives the SignalLabAdmin group the “REGISTER_LS” function on the target

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

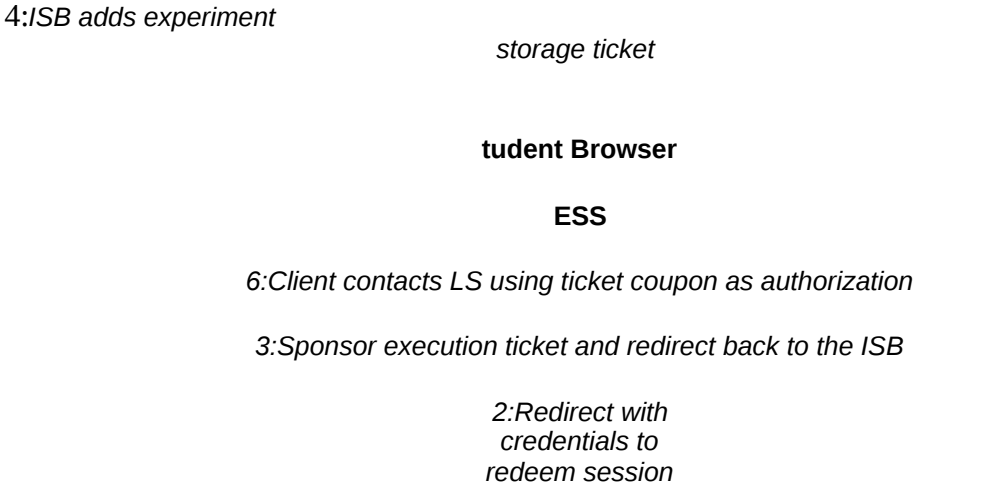
SignalLab. This includes creating a minimal process agent entry with process agent name and temporary GUID as well as a grant target for the SignalLab. The true GUID for MISB will be supplied later by the lab server itself (in the return value from the future **`InstallDomainCredentials`** call).

Bob logs into MISB and selects the SignalLabAdmin group to be his effective group. A search through the grants available to Bob as a member of this group triggers the business logic of the ISB to offer the option of administering the SignalLab. When Bob chooses this option, he is directed to the process agent administration page for SignalLab, but the page logic detects that the lab has not been completely registered. It, therefore, displays a form that asks Bob for an initial passkey and the web service address for the SignalLab. Once these two fields are entered, the option to install domain credentials becomes active. When Bob clicks it, MISB makes an **`InstallDomainCredentials`** call to SignalLab using the passkey just entered. SignalLab returns a full **`ProcessAgent`** instance to complete the registration of the server. The fields now become visible on the MISB process agent administration page for SignalLab.

This completes the first phase of the registration of the new lab server, but no users can execute experiments until Bob has also registered the lab server with the domain's USS and LSS. He does this by triggering a **`Register`** call from his privileged account on the ISB. We defer detailed discussion of this more elaborate procedure to the second document in this series: *iLab Interactive Ticketing and Integrated Management — Operation.*

Multiple Domain Process Agent Registration Example

Now let's work through the multiple domain equivalent of the previous example. In this case, Bob and the SignalLab are located at another university, the University of Queensland (UQ), running its own service broker, QISB. The goal is to allow students at MIT to use SignalLab by authenticating through the MIT service broker, MISB. We assume that Bob has already gone through the local registration procedure described in



the previous section. Thus, there is a group SignalLabAdmin on QISB, and Bob is a member of it. QISB has installed domain credentials on SignalLab.

The next phase of the registration involves the exchange of domain credentials between MIT's MISB and UQ's QISB. Sarah contacts her counterpart, Ann, at UQ, and then registers for an account on the UQ service broker. Ann creates a new group, MISBAdmin, on QISB, and makes Sarah a member of the group. She also creates a grant that gives the MISBAdmin group the "REGISTER_ISB" function on the target MISB. As before, this includes creating a minimal process agent entry with process agent name, MISB, and temporary GUID for it as well as a grant target for MISB. Remember that ISBs are actually a specialized type of process agent.

Sarah logs into QISB and selects the MISBAdmin group to be her effective group. Just as in the previous example, the UQ ISB recognizes that MISB is not properly registered, and it walks Sarah through a process that triggers QISB to invoke **`InstallDomain-Credentials`** on MISB. The end result is that MISB and QISB can now exchange ticketing information.

Now Ann as administrator of QISB must give MIT permission to use UQ resources. She starts by creating two grants on QISB. The first allows MISB, standing as proxy for the whole MIT domain, EXECUTE_EXPERIMENT permission on SignalLab. This will allow MISB to grant the MIT USS permission to write execution tickets for the SignalLab, but that is the MISB view. Ann and QISB treat the MIT domain as a black box with MISB the gateway. The second grant that Ann creates permits MISB, once again as proxy, REQUEST_RESERVATION permission on the UQ LSS. MIT's USS will need this permission to schedule reservations for MIT students, but once again UQ doesn't need to know about the MIT USS. UQ views the whole MIT domain through MISB.

The final step of this stage is for QISB to make a **`Register`** call to MISB with an array of two **`Registration`** objects. The first corresponds to the first grant, and it supplies

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

a template for the execution tickets that MIT's USS will sponsor on MISB for MIT's students. The "execute" **`address`** field in the **`Registration`** tells MISB to forward the registration to the MIT USS. The second **`Registration`** object actually supplies the ticket coupon that the MIT USS will need when it wants to confirm reservations with the UQ LSS. The first only contains a template because all execution tickets are tied to a particular user and lab session. The ticket that the USS uses to confirm reservations, however, is a "permanent" (good until revoked) ticket between process agents.

Now Sarah must create the hierarchy of groups for the MIT class that will use SignalLab. She must also create the resource mapping for this group and its associated resources: SignalLab, the MIT USS and ESS. She should also enter a grant on MISB that will allow the UQ (more particularly, its LSS) to contact the MIT USS to cancel reservations if SignalLab must be taken down for maintenance. This triggers a **`Register`** call from MISB to QISB that supplies a REVOKE_RESERVATION ticket. QISB forwards the **`Registration`** object to the LSS.

The document, *iLab Interactive Ticketing and Integrated Management — Operation*, discusses the workflow and mechanisms that underlie this mutual registration in more formal detail. This discussion should give the reader a sense of why integrated management is necessary given the complexity of the registration steps required.

Appendix 1: The Coupon Data Structure in Action

Example 1: A Lab Server creating a Blob Record on the ESS

The SOAP header of the Web Service call contains an Operation Authentication Header with embedded **Coupon.**

```
<?xml version="1.0" encoding="utf-8" ?>
```


4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

```
 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
   <soap:Header>
     <OperationAuthHeader xmlns="http://ilab.mit.edu/iLabs/type">
       <operationCoupon>
        <couponId>446</couponId>
        <issuerGuid>1be13120-04a3-4306-9b5c-9c184d694c10</issuerGuid>
        <passkey>986484769259805</passkey>
       </operationCoupon>
    </OperationAuthHeader>
  </soap:Header>
   <soap:Body>
     <CreateBlob xmlns="http://ilab.mit.edu">
      <experimentID>143</experimentID>
      <description>elephant image</description>
      <byteCount>38951</byteCount>
      <checksum>5D95EE7743CC1DC471EBD6350A5C619</checksum>
      <checksumAlgorithm>md5</checksumAlgorithm>
    </CreateBlob>
  </soap:Body>
</soap:Envelope>
```

The CreateBlob web method returns the ID of the BLOB that was created on the ESS

```
<?xml version="1.0" encoding="utf-8" ?>
 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
   <soap:Body>
     <CreateBlobResponse xmlns="http://ilab.mit.edu">
      <CreateBlobResult>130</CreateBlobResult>
    </CreateBlobResponse>
  </soap:Body>
</soap:Envelope>
```


4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

Example 2: The URL of the USS, after Redirection from the Service Broker to redeem a reservation with a **Coupon** embedded in the query portion of the URL. The "sb_url" parameter is the URL on the SB that initiated the redirection to the USS.The user will be redirected back to that URL after session redemption on the USS

http://ceci-wireless3.mit.edu/USS/Reservation.aspx?
coupon_id=444&issuer_guid=1be13120-04a3-4306-9b5c-9c184d694c10&
passkey=8491429299344 97&
sb_url=http://platypus.mit.edu/InteractiveSB/myClient.aspx

Example 3: USS retrieving a ticket of type "SCHEDULE SESSION" from the Service Broker

The SOAP header of the Web Service call contains an Agent Authentication coupon The `RedeemTicket` web method takes 2 arguments: a Coupon and a Ticket type.

```
<?xml version="1.0" encoding="utf-8" ?>
 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
   <soap:Header>
     <AgentAuthHeader xmlns="http://ilab.mit.edu/iLabs/type">
       <agentCoupon>
        <couponId>46</couponId>
        <issuerGuid>1be13120-04a3-4306-9b5c-9c184d694c10</issuerGuid>
        <passkey>284327138347703</passkey>
       </agentCoupon>
    </AgentAuthHeader>
  </soap:Header>
   <soap:Body>
     <RedeemTicket xmlns="http://ilab.mit.edu">
       <coupon>
        <couponId xmlns="http://ilab.mit.edu/iLabs/type">444</couponId>
        <issuerGuid xmlns="http://ilab.mit.edu/iLabs/
type">1be13120-04a3-4306-9b5c-9c184d694c10</issuerGuid>
        <passkey xmlns="http://ilab.mit.edu/iLabs/type">849142929934497</
passkey>
```


4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

```
      </coupon>
      <type>SCHEDULE SESSION</type>
    </RedeemTicket>
  </soap:Body>
</soap:Envelope>
```

The **`RedeemTicket`** web method returns the retrieved ticket.

```
<?xml version="1.0" encoding="utf-8" ?>
 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
   <soap:Body>
     <RedeemTicketResponse xmlns="http://ilab.mit.edu">
       <RedeemTicketResult>
        <ticketId xmlns="http://ilab.mit.edu/iLabs/type">908</ticketId>
        <type xmlns="http://ilab.mit.edu/iLabs/type">SCHEDULE SESSION</type>
        <couponId xmlns="http://ilab.mit.edu/iLabs/type">444</couponId>
        <issuerGuid xmlns="http://ilab.mit.edu/iLabs/
type">1be13120-04a3-4306-9b5c-9c184d694c10</issuerGuid>
        <sponsorGuid xmlns="http://ilab.mit.edu/iLabs/
type">1be13120-04a3-4306-9b5c-9c184d694c10</sponsorGuid>
        <redeemerGuid xmlns="http://ilab.mit.edu/iLabs/
type">546f6696-95b2-4306-8f5f-56a8bf5d3b93</redeemerGuid>
        <creationTime xmlns="http://ilab.mit.edu/iLabs/type">63279847447</
creationTime>
        <expirationTime xmlns="http://ilab.mit.edu/iLabs/type">36000</
expirationTime>
        <isCancelled xmlns="http://ilab.mit.edu/iLabs/type">false</isCancelled>
        <payload xmlns="http://ilab.mit.edu/iLabs/type">
          <ScheduleSessionPayload xmlns:ns="http://web.mit.edu/ilab/tickets"
ticketType="SCHEDULE SESSION">
            <userName>rabih</userName>
            <groupName>Time Of Day Group</groupName>
            <sbGuid>1be13120-04a3-4306-9b5c-9c184d694c10</sbGuid>
            <labClientName>Interactive Time of Day client</labClientName>
            <labClientVersion>1.0</labClientVersion>
          </ScheduleSessionPayload>
```


4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

```
        </payload>
      </RedeemTicketResult>
    </RedeemTicketResponse>
  </soap:Body>
</soap:Envelope>
```


4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with
credentials to
redeem session

Appendix 2: The `ProcessAgent` *Data Structure*

`ProcessAgent`

Members:

`string agentGuid`

A GUID that identifies the process agent

`string agentName`

A human readable name used in administrative interfaces to identify this process agent

`string type`

A string drawn from a fixed set that specifies the type of process agent

`string description`

A string that provides a brief description of the function of the agent in its domain

`string infoUrl`

A string containing the URL that points to a web document that provides further information about the agent

`string contactEmail`

An email address through which the administrator of the agent can be reached

`string webApplicationUrl`

A string containing the URL that points to the home page of the agent's web application; **`null`** *if the agent only presents a web service*

`string webServiceUrl`

A string containing the URL that points to the home page of the agent's web service; **`null`** *if the agent only presents a web application*

`float timezone`

4:*ISB adds experiment*

storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with credentials to redeem session

A **`float`** *that indicates the delta from GMT*

4:*ISB adds experiment*
storage ticket

tudent Browser

ESS

6:Client contacts LS using ticket coupon as authorization

3:Sponsor execution ticket and redirect back to the ISB

2:Redirect with
credentials to
redeem session