University of Technology, Sydney
Faculty of Engineering and Information Technology

INVESTIGATION AND IMPLEMENTATION OF A FEDERATED REMOTE LABORATORY ARCHITECTURE USING THE OPEN SOURCE SAHARA LABS SYSTEM

by
Michael Diponio

Student Number: 10306797
Project Number A11-045
Major: Software Engineering

Supervisor: Professor David Lowe

A 12 Credit Point Project submitted in partial fulfilment of the requirement for the Degree of Bachelor of Engineering

25th November 2011

STATEMENT OF ORIGINALITY

ABSTRACT

Developing a remote laboratory is an expensive investment however it has the ability to be shared. Providing this access ameliorates the cost over many uses by users that need not be resident to the remote laboratory or even the institution that provides the remote laboratory. The Labshare Institute (TLI) is an organisation that takes this premise as it's raison d'être. SAHARA Labs, sponsored by TLI, is an open source software suite that enables remote access to physical, computer controlled apparatuses (called rigs) over the Internet. It provides the software building blocks to implement a remote laboratory site (called a provider) and is currently used by multiple universities, including the University of Technology, Sydney to build their remote laboratories.

However, going to a SAHARA Labs website allows the use of rigs hosted within that provider but not rigs hosted at other providers and currently has no support for sharing. This capstone explores architectures such as centralised, distributed and peer-to-peer architectures that allow the creation of a federation of remote laboratories with the end goal of logging into a SAHARA Labs website to use rigs hosted at other providers. Using a lens of contextual (existing and legacy laboratories, inclusiveness), operational (ownership, access negotiation, user perception) and technical (performance, scalability, security) factors an architecture has been chosen which best suits the key actors in a remote laboratory federation, TLI, the providers and the end users.

This architecture is then developed, designed and implemented as a prototype system from the existing basis of SAHARA Labs. It is validated to ensure suitability of the base architecture with the future goal of a new SAHARA Labs release utilising this architecture and the capstone developed prototype.

ACKNOWLEDGEMENTS

I would like to acknowledge the contributions of the following individuals without whom this project would not have been possible:

- *Prof. David Lowe*
- *Mr. Michel de la Villefromoy*
- *The UTS Remote Laboratory team*
- *The Labshare Institute*
- *Ms. Jessica Diponio*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Word	Definition
Consumer	A member in the remote laboratory federation that consumes a rig hosted by a provider. This is the guise a site takes during consumption.
Provider	A member in the remote laboratory federation that provide a rig for use by a consumer. This is the guise a site takes during provision.
Resource	An identifier that resolves to a rig. The method of resolution is unimportant to the context of a resources use but may be a specific rig, rig type or a list of tags a rig must provide.
Rig	A specific apparatus instance. Rig time is the quantity scheduled to be assigned during sessions.
Rig Type	A grouping of rigs where each member is a fungible.
Multisite	The code name for the federation system implemented in this project.
WSDL	Web Service Definition Language file which defines the operations, messages, parameters, bindings and endpoints of SOAP web services.
XSD	XML schema file which defines XML complex types.
REST	Representation State Transfer, an architectural style which allows web services to be provided using HTTP and a response message format such as XML or JSON.
JSON	Document message format which is syntactically valid JavaScript. This can conveniently be serialised and parsed into JavaScript objects.
SOAP	Simple Object Access Protocol, a specification to develop web services.
OSGi	A Java framework that provides modularity, application life-cycle management and a service registry.
AJAX	Communication mechanism which allows a web page and web server to communicate without a full page load.
Servlet	A Java interface that is the entry point for network request processing. A HTTP Servlet processes HTTP requests.

Module	A set of Java classes which has a physical representation as a Jar file. Each module has a public interface, private implementation, lifecycle and dependency with other modules. This information is stored in the Jar manifest. The term 'module' is interchangeably used with the term 'bundle'.
Bundle	A set of Java classes which has a physical representation as a Jar file. Each bundle has a public interface, private implementation, lifecycle and dependency with other modules. This information is stored in the Jar manifest. The term 'bundle' is interchangeably used with the term 'module'.

1. INTRODUCTION

A remote laboratory is a system that allows the use of computer controlled apparatuses without the user being within a close proximity of the apparatus. A user can just as effectively use a remote laboratory if they are thousands of kilometres away or if they are in the next room. Fundamentally, a remote laboratory is a web system. It authenticates users, manages access and provides the point of access for the apparatus. Whilst these systems are typically developed organically to satisfy a local need these needs are not unique. Provided there is capacity to satisfy local needs, it seems *obvious* they should be shared. The basic premise of sharing provides benefits for those that consume and those that provide and this is what this capstone is borne from.

SAHARA Labs is an open-source software suite that enables remote access to physical, computer controlled laboratories using the Internet. It has been continually developed over the last two years initially by the Labshare project, which was an Australian Government-funded project that aims to create a national network of shared remotely accessible laboratories. The Labshare project has spun off into The Labshare Institute (TLI) who continues to do the work that was started by the Labshare project; that is to share remote laboratories.

SAHARA Labs currently allows laboratories (rigs) to be made remotely accessible which broadly includes the following: functionality, rig management to add laboratories and ensure they are fit for use, user management to add users, permission management to specify which users may access which rigs and with what constraints, scheduling and assignment to equitably share the resource across many users with queuing and reservation. It is currently used by multiple universities, called providers, with each installation as an isolated instance of a remote laboratory not allowing *direct* sharing between each laboratory. Going to a SAHARA Labs website allows the use of rigs hosted within that provider but not rigs hosted at other providers. If there is to be sharing, this creates an overhead because users and permissions must be created on a per-SAHARA Labs basis. It increases the administration required and as this requires potentially sensitive user information such as names, enrolment and contact details to be provided as a precondition of sharing, it may not be possible to conduct sharing with institutions that cannot provide this information. The user experience is also diminished because if a user has access to multiple remote laboratories hosted at different providers, they must remember each of the different SAHARA Labs website locations and the authentication details to access them.

This project aims to allow each of the SAHARA Labs systems to be interconnected in a federation which allows seamless use of the rigs hosted at separate institutions. A rig hosted at an institution should be visible to other institutions with certain constraints and when suitable permission is negotiated for access, it should be usable at the users 'home' SAHARA Labs provider. The implications of this are user and permission management is delegated not to the providing institution but to the consuming institution which considerably eases sharing. It also provides a much more seamless experience having only to use a single site for all remote laboratory access. Making sharing easier to provide a more seamless experience for users will help achieve the goal of TLI and create a 'national network of shared remotely accessible laboratories'.

The objective of this project is to develop an architectural model of a federation system and a functionally complete SAHARA Labs federation system. The scope of this project is to achieve this to a proof-of-concept level which demonstrates the process of sharing and possible operations of a sharing network of remote laboratories. This proof-of-concept should be indistinguishable from a production system but should not be expected to continuously function correctly in a production environment. The proof-of-concept will:

- Serve as a basis to validate the architecture as an appropriate federation model.
- Serve as an interactive system for discovery of further attributes that a federation model would need to satisfy (this discovery is however outside the scope of this project).
- Provide a basis for a system which may be built on to become a production ready system.

This report is composed of five chapters and starts with a literature review. The literature review primarily looks at whether any equivalent federation systems exist and if they do, what lessons those systems implementers have learnt.

The next chapter (Chapter 3), looks at SAHARA Labs as it is the basis of this projects development. First, an exposition of the existing components in SAHARA Labs and the communication between them is provided. Then the chapter looks at various architectures that extend SAHARA Labs so it allows the use of rigs in a federation. A choice is made on which architecture will be implemented. Closely followed in the next chapter (Chapter 4) is the implementation of this architecture and verification on whether it allows federation use of rigs. Chapters 3 and 4 set the basis of what the federation's management must provide to support federation use. Chapter 5

investigates the architectures that provide this management, choosing a suitable architecture. Chapter 6 implements this architecture.

Finally, the report ends in a conclusion evaluating whether the architectures developed and their implementations were successful. A list of recommendations for further development of the overall federation system is given here.

2. Literature Review

This chapter highlights notable paper and journals found when researching this projects. The central subject used in the search of literature on remote laboratories was to determine if any comparable systems to a federation remote laboratory exist.

2.1 Current Trends in Remote Laboratories

Gomes, L. Bogosyan S. 2009, 'Current Trends in Remote Laboratories', IEEE Transactions on Industrial Electronics, vol 56, no. 12, pp. 4746 – 54

This journal article looks at a range of topics about remote laboratories in engineering education. It starts with a characterisation of remote laboratories and then looks at their benefits. It provides details on what components comprise a remote laboratory and lists many different laboratories located around the world categorised by domain.

The pertinent information in this article is its summary of the current implementations of remote laboratories. It states a common feature of most laboratories is they offer stand alone solutions will limited or no capability to cooperate with other platforms. Further, most of these solutions use special and ad hoc solutions relying on different types of technologies (Gomes, Bogosyan 2009).

2.2 The iLab Shared Architecture: A Web Services Infrastructure to Build Communities of Internet Accessible Laboratories

Harward, J. Del Alamo, J. Lerman, S. Bailey, P. Carpenter, J. DeLong, K. Felknor, C. Hardison, J. Harrison, B. Jabbour, I. Long, P. Mao, T. Naamai, L. Northbridge, J. Schulz, M. Talavera, D. Varadharajan, C. Wang, S. Yehia, K. Zbib, R. Zych, D, 2008, 'The iLab Shared Architecture: A Web Services Infrastructure To Build Communities of Internet Accessible Laboratories', Proceedings of the IEEE, vol 96, no. 6, pp 931 – 50

Looking through the literature of remote laboratories, one the most prominent is iLabs developed by MIT. This journal article describes the history, goals, architecture, pedagogy and growth of the iLabs community.

The first notable point in this journal with relation to federation systems is a stated goal of the iLabs Shared Architecture (ISA) quoted below:

> ... ISA must separate the responsibility for delivering an online laboratory experience from that of managing the students who use it. Lab developers generous enough to share their laboratory equipment should not be burdened with administering the usage of students they are not teaching (Harward, et al 2008)

This is a goal that also applies to a SAHARA Labs federation system. The iLaba Shared Architecture (ISA) has two forms, the batched middleware and interactive middleware which refers to the types of rigs that run on them. The batched middleware architecture has:

- **Lab Server** – This resides laboratory side and directly interfaces with the instruments that execute experiments.
- **Service Broker –** This is an application that resides at a user's institution rather than at the laboratory side and serves as the middleware to connect Lab Clients through to the Lab Server.
- **Lab Client –** This is an application that runs on a user's local machine to generate experiment specifications.

The ISA addresses delegating administering accounts to a user's institution by having a 'Service Broker'. Users only ever talk to their 'Service Broker' who forwards the user's 'Lab Client' generated experiment specification to the 'Lab Server' for execution if it can authenticate and vouch for the users.

Another interesting point in the ISA is the strategy of identifying users between the 'Service Broker' and 'Lab Server'. The 'Lab Server' never knows about individual users, only *effective* groups. The 'Lab Server' trusts the 'Service Broker' to authenticate users and provide the correct effective group when a user's experiment specification is submitted.

2.3 A Network of Automatic Control Web-Based Laboratories

Vargas, H. Sanchez, J. Jara C.A. Candelas, F.A. Torres, F. Dormido, S., 2007, 'A Network Of Automatic Control Web-Based Laboratories', IEEE Transactions on Learning Technologies, vol. 6, no. 1, pp 1 – 14

This journal article discusses the 'AutomatL@bs' project which, the journal claims, is a complete network of automatic control web based laboratories. According to the journal several Spanish universities benefit from this project as it allows the sharing of experimental resources thus increasing the number of laboratories for their students.

The pertinent information of this journal is the architecture of the 'AutomatL@bs' system. This architecture is centralised with a single website for all laboratories. Users login to the 'AutomatL@bs' web site to create reservations rather than a web site located at their institution. The laboratories are however located at multiple Spanish universities.

2.4 Design and Implementation Issues for Modern Remote Laboratories

Guimaraes, E. Cardozo, E. Moraes, D. Coelho, P., 2007, 'Design and Implementation Issues for Modern Remote Laboratories', IEEE Transactions on Learning Technologies, vol. 6, no. 1, pp. 1939 - 55

This journal article details attributes of remote laboratories and provides a reference architecture of an *ideal* remote laboratory. The pertinent information in this journal is a list of requirements for federated operation of web labs. These are:

- Management of users
- Management of resources
- Management of SLA, policies and QoS
- Management of access

The second part of this paper is a discussion of the 'REALabs' platform. The federation in this system is only Single Sign On using OpenSAML and the Shibboleth protocol. It does not appear this platform provides any sort of federation authorisation or meets most of this journals stated requirements for federation operation.

2.5 A Consortium of Secure Remote Access Labs for Information Technology Education

Toderick, Lee. Tijjani, M. Tabrizi, M., 2005, 'A Consortium of Secure Remote Access Labs for Information Technology Education', In Proceedings of SIGITE Conference 2005, pp. 295 – 99

This journal article details a consortium that provides access to computer network domain laboratories across five different sites. The laboratories themselves are all equivalent and are Cisco routers to teach information technology skills. The interesting feature of this consortium is it provides a much more fluid demand and surplus relationship between the consortium members than most other federated remote laboratory systems. The system automatically attempts to use an in house resource and if none is available only then attempts to contact another consortium member to use their resource.

3. INVESTIGATION OF SAHARA LABS

The remote laboratory federation system this capstone explores and implements is an extension of the existing SAHARA Labs system. This chapter investigates and describes the SAHARA Labs implementation in terms of user interfaces, a macro view of all the architectural components (stand alone applications that communicate via inter-process communication) and a brief micro view of how the applications that will need to be modified are constructed (internal modules and significant interfaces). Important attributes that will be investigated are the communication mechanisms of SAHARA Labs which will need to be replicated across in the federation system and how customisations to SAHARA Labs are implemented. Customisations (such as specific rig interfaces) may need to be replicated to consuming sites. To do this, the way they are currently implemented must be understood.

In the last section of this chapter, extensions to this architecture are discussed as possible avenues of implementing the use of SAHARA Labs in the federation system. Multiple architectures are shown and one is chosen on the basis of whether it achieves the use goal and how well it fits a list of software quality attributes.

Figure 1 Screenshot of UTS's SAHARA Labs r3.1

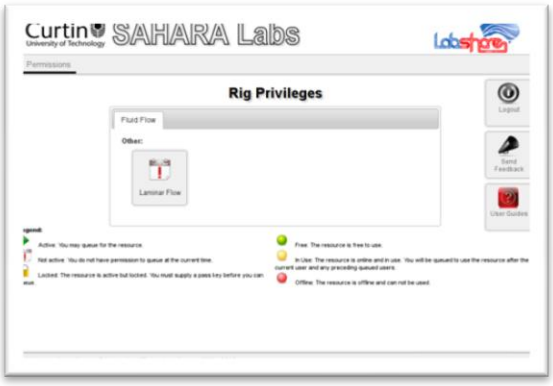


Figure 2 Screenshot of Curtin's SAHARA Labs r2.0

Figure 3 Screenshot of RMIT's SAHARA Labs r3.0

3.1 WHAT IS SAHARA LABS

SAHARA Labs is a software system that allows remote laboratory systems to be built. It is intended to allow physical apparatuses to be implemented for use remotely over the internet in a managed fashion. This means users, if properly authorised, are provided discrete blocks of time for use of the apparatus through scheduling schemes such as priority queuing and reservations. The use cases below show the functionality provided by the system and the subsequent table describes each actor and use case. The functionality may be something the system actually performs such as *'Reserve* Rigs' or it may be an action performed by a user to productively use the system such as '*Create Lessons'*. These are included because whilst they may not be a function the system needs to specifically provide, they have implications on the systems use thus on associated functionality. In the federation architecture these use cases are divided into the *consumer* and *provider* domain.

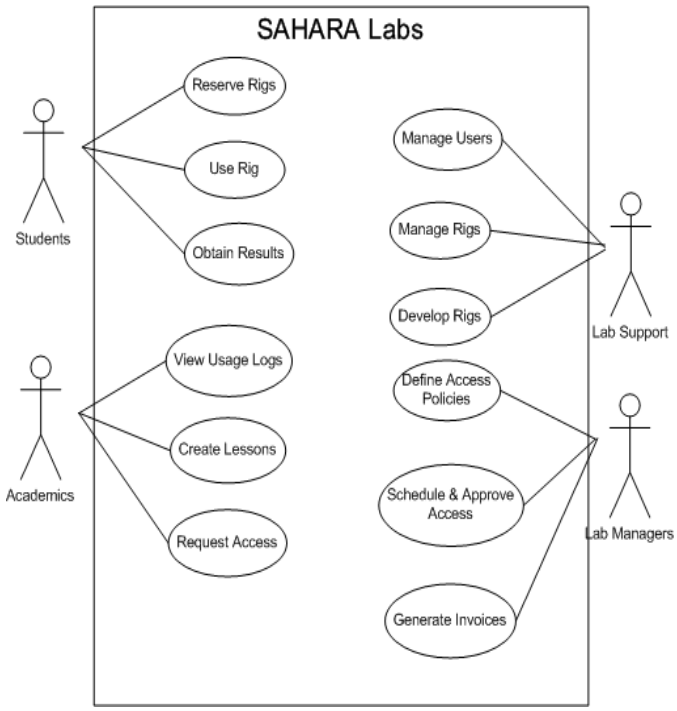


Figure 4 Use cases of SAHARA Labs

Table 1 Use case descriptions

Actor	Use case	Description
Students – consumers of rigs.	Reserve rigs	This allows the student to select a time when the rig is kept for their sole use.
	Use rigs	The actual use of the rig which is the most important part of the system. Every other

		function is peripherally important and supporting of this function. If this doesn't work, there is no point of the system.
	Obtain results	The output of the 'Use rigs' use case which may be persistent for further obtainment after the rig use is complete.
Academics – determines how the rigs are used (lessons) and are requestors of rigs to be provided to students.	Request access	For academics to have their students using rigs they must first request access for their students.
	Create lessons	For academics to have their students using rigs there must be some purpose behind their use. This is about creating that purpose, a *lesson*.
	View usage logs	Verification of whether students actually used rigs and for how long.
Lab Support – technical support that ensures rigs are present and fit for use and ensures users may login and use the system.	Manage users	This is setting up accounts and assigning and revoking access to rigs for those accounts to use.
	Develop rigs	This is developing physical apparatuses for remote use and adding them as rigs.
	Manage rigs	This ensures rigs are operational. This is performing such tasks as scheduling maintenance periods for rigs so they may not be used.
Lab Managers – technical support that ensures equitable access to the rigs and ensures operational function of the lab (rigs are provisioned and maintained).	Define access policies	This is defining which rigs are available and when they may be requested for access.
	Schedule and approve access	This is the corollary to the academic '*Request Access'* use case. Once academics have requested access, the lab manage checks any equivalent requests and approves access to the request.

A useful distinction to make with the existing use cases is dividing them into consumer and provider use cases. The following diagram shows this:

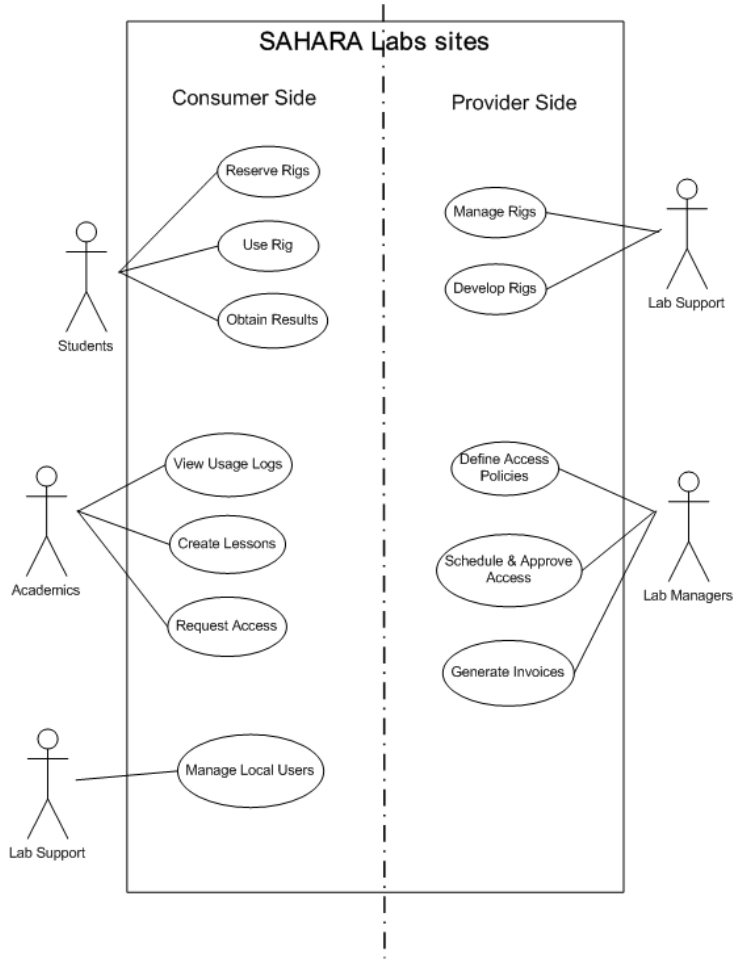


Figure 5 Use cases dividing in consumer / provider

Dividing the use cases into consumer and provider use cases provides a first-pass distinction of SAHARA Labs in a federation context:

- Rigs are provider side and the use of rigs is consumer side. This suggests there is a synchronisation of users and permissions between providers and consumers since it is permissions that define what a user may view in the user interface and what they are allowed to use.
- Requests for use by academics are consumer side but approving requests by lab managers is provider side. This is the same divide as use and rig provision. This may be an advantageous time to perform synchronisation of permissions.
- Users are consumer side entities but providers must know about unique users to correctly schedule users to rigs. One of the assumptions SAHARA Labs has about rigs is that they are interactive and a user may only use one rig at a time.

Not knowing unique users prohibits a provider from only upholding this assumption.

- A new use case is needed to generate invoices. How these are generated and what form they take are unimportant to the provider / consumer relationship in a *system* sense but what is important is accurate records of accesses are kept by both providers and consumers so invoices may be generated and there is agreement these are correct.

3.2 How is it used?

The section shows a walkthrough of using SAHARA Labs using screenshots of the major user interface pages. It is important to note in the walkthrough the interaction between the user and the system and thus the communication source this triggers which may be by the user or some event generated by the system. User communication can be seen as the result of a user request. System event triggered communication can be seen as waiting overlays or waiting pages (such as the queue waiting page). This is because the client web page is polling the system to determine if the desired event has occurred. In a federation system, user initiated communication is likely to be generated at the consumer and requested of the provider and system initiated communication is likely to be generated at the provider which then notifies the consumer.

Figure 6 User logs in

Figure 7 User views resources they have access to.

Figure 8 User determines whether a resource can be used

Figure 9 User viewing status of resources

Figure 10 User creating reservation

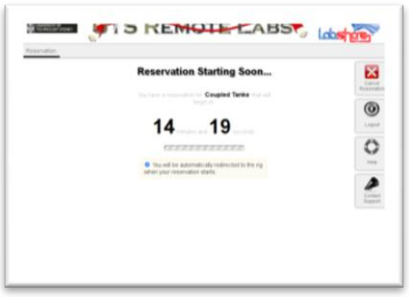


Figure 11 Waiting for reservation to start

Figure 12 User waiting to be assigned to rig

Figure 14 User using a rig

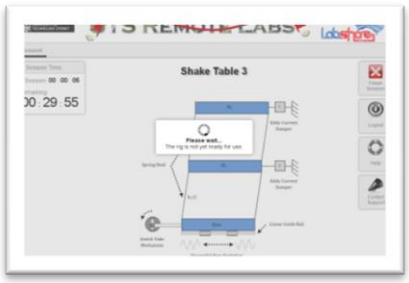


Figure 13 Waiting for a rig to be ready for use

Figure 15 User finishing session

The table below describes each communication suggested by the user interfaces and whether it is required in federation.

Communication for page	Required between consumer and provider	Reason	Initiated by
User login.	No.	User authentication is solely a consumer concern.	N/A
Determining what resources a user can access.	No.	A consumer knows the permissions the user has access to thus the resources they can access.	N/A
Determining if a resource is fit for use.	Yes	The rig is attached to the provider.	Consumer
Queuing for a rig.	Yes	The provider has to manage the intersection of potentially multiple consumers of the rig so it must only assign it to the rig to a single user at a time.	Consumer
Determining queue position.	Yes	The provider holds the queue.	May be consumer (poll) or provider (push)

Determining whether a user has been assigned to a rig.	Yes	The provider holds the queue and the rigs are attached to the provider.	Provider.
Finding when a rig is free for reservation.	Yes	The provider has to manage the intersection of potentially multiple consumers of the rig so it only knows which times it is free.	Consumer
Reserving a rig.	Yes	The provider has to manage the intersection of potentially multiple consumers of the rig so it only knows which times it may be reserved.	Consumer
Determining when a reservation has started.	Yes	The provider holds all reservations of the rig and performs redeeming of reservations.	Provider.
Determining whether a rig is ready for use.	Yes	The rig is managed by the provider.	Provider
Finishing a session.	Yes	The rig is queued and has reservations redeemed by the provider so it must know when sessions are finished to correctly schedule the next session.	May be consumer or provider depending on reason for termination.

3.3 THE CURRENT ARCHITECTURE

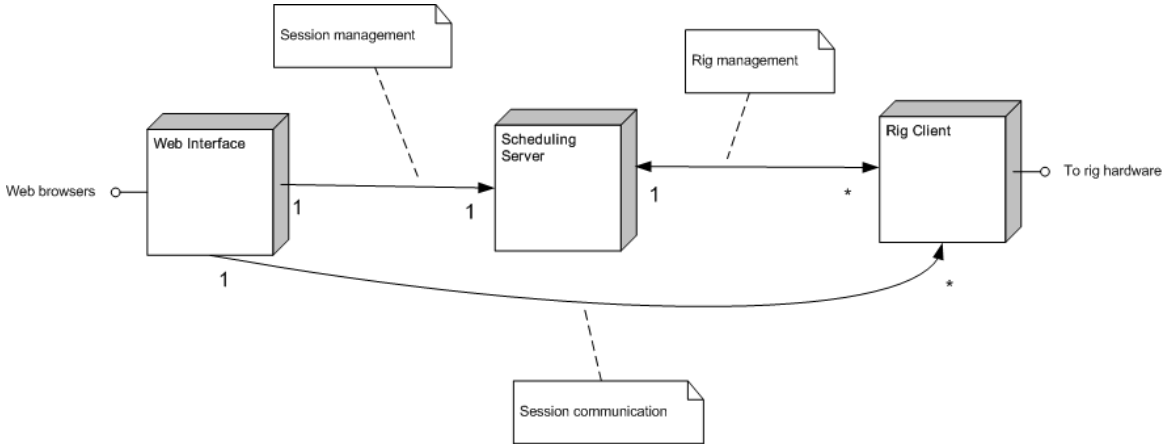


Figure 16 Current architecture

The architecture of SAHARA Labs has the three following components:

- **Web Interface –** The Web Interface is the user interface to SAHARA Labs. Its roles are authenticating a user, providing the interface for users to request to use a resource and providing the interface to control a rig. It provides programmatic interfaces to implement web interface type concerns such as how users are authenticated and presentation. Also, as rigs may come from different types, the Web Interface allows the user interface of the rig to be implemented custom to that rig in collaboration with the rigs *rig client.*
- **Scheduling Server –** The Scheduling Server is the middleware that manages the remote laboratory. It tracks the state of rigs and assigns those to users. Assignment comes in two forms, which are *on demand* to provide immediate or nearest free access (queuing) or *reserved* to allow a time slot to be selected for a rig to be made available for use (reservation). The Scheduling Server also manages running sessions and ensures sessions only run to permission defined allocated time which includes a guaranteed portion and zero or more time extensions if the rig is free from use by other users. The Scheduling Server is agnostic to the specifics of a rig and relies on a *rig client* application to turn scheduling requests in rig specific behaviour.
- **Rig Client –** The Rig Client is the abstraction of a rig that converts abstract requests to rig specific actions. If a user is being assigned to the rig, its Rig Client provides the behaviour to actually allow the user to access the rig. The Rig Client provides a programmatic interface to allow this behaviour to be defined as a set of implementations of Java interfaces that declare this behaviour. The Rig Client manages the rig and sends status updates to the

Scheduling Server. If the Rig Client is not operating, from the perspective of the Scheduling Server, the rig is not operational and no user will be assigned to it. The Rig Client also provides a control channel to directly interface with the rig or as an auxiliary to an external control program.

The relationships between the components are as follows:

- **Web Interface to Scheduling Server** – The Web Interface requests services from the Scheduling Server using SOAP requests over a HTTP binding. This Scheduling Server has no direct knowledge of the Web Interface but trusts any request made using the SOAP interface the Web Interface calls (the Web Interface performs validation using user authentication).
- **Scheduling Server and Rig Client** – There is a duplex communication between these two components. The Rig Client uses SOAP requests over a HTTP binding to send life cycle messages to the Scheduling Server such as when it has started up, when it is shutting down and regular status updates about the rig. These act as a keep alive and may notify the Scheduling Server of detected rig errors. The Scheduling Server uses SOAP requests over a HTTP binding to send session messages to the Rig Client including when a user has been assigned to the rig and when a user should have access revoked at the end of their session. The communication between the Scheduling Server and Rig Client is authenticated using a mutable token generated by the Scheduling Server that is provided to the Rig Client on start-up registration. If the correct token is not provided to the Rig Client, the Rig Client will not honour the request, instead returning an error to the caller.
- **Web Interface and Rig Client –** The Web Interface communicates with the Rig Client during sessions. It may obtain rig specific parameters such as audiovisual URLs from the Rig Client or it may be directly used to control the rig using the Rig Clients control channel. The communication between the Web Interface and Rig Client is authenticated using the assigned user's name. If no name is provided or the name is not the same as the assigned user's name, the request is not honoured, instead returning an error to the caller.

The interfaces to the Scheduling Server and Rig Client are typically private and are not usually externally usable. They are typically protected by firewalls or are sand-boxed on private networks. The only external accessible interface is the web interface. The

web interface allows web browsers to externally request the remote laboratory as page loads and using a REST / JSON interface for AJAX interaction with the rig.

3.4 SCHEDULING SERVER

The Scheduling Server is the middleware of SAHARA Labs and manages the remote laboratory. This section briefly describes the internal structure of the Scheduling Server. This is described because it is likely the Scheduling Server is the component that requires most modification to support federation.

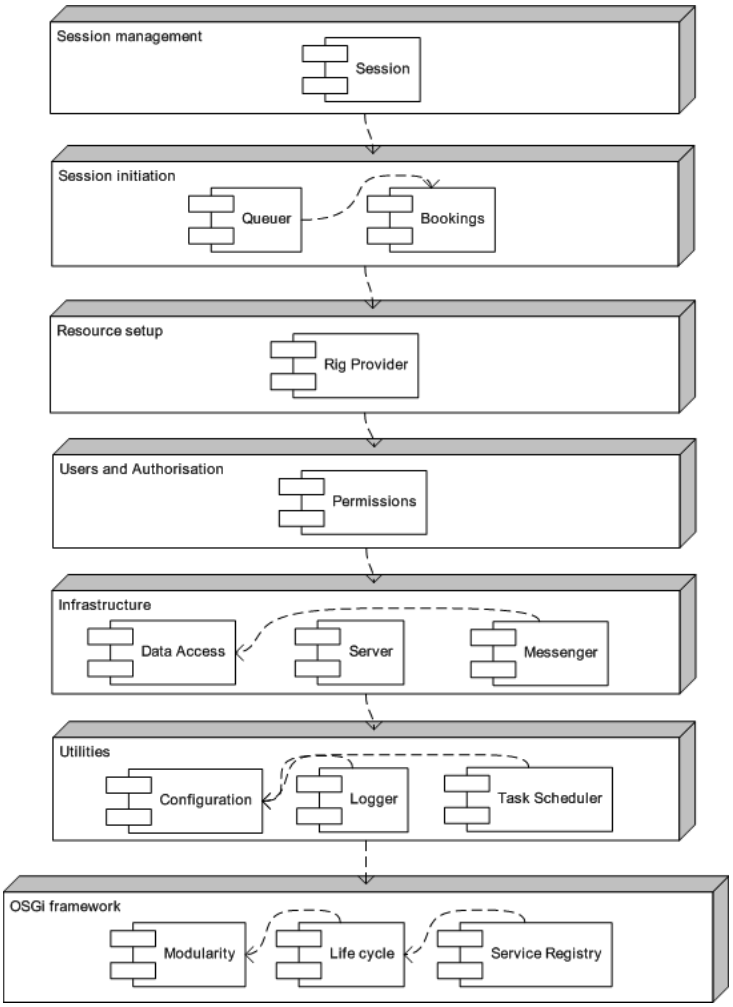


Figure 17 Scheduling Server structure

The Scheduling Server is implemented in Java as a set of OSGi bundles that run on the OSGi framework. Each bundle performs a task and in collaboration with the other bundles makes up the Scheduling Server. The above diagram shows a subset of these bundles which are significant to a federation implementation. It shows a layered diagram of the bundles, where the top most bundles depend on the lower bundles. The bottom layer of the diagram is the OSGi framework which runs and controls communication with the other bundles. The other bundles are:

- **Utilities –** This is a combination of three bundles that provide utility functions to the rest of the application. These are logging, configuration and a common thread pool that other bundles can register threads with.
- **DataAccess –** This bundle is the persistence of the application and interfaces with a relational database, typically MySQL. Through this bundle other bundles can store, retrieve, update and delete records.
- **Messenger –** This bundle allows notifications to be sent to users using email.
- **Server –** This bundle allows HTTP servlets to be hosted. Each of the bundles that require a SOAP service to be hosted, register it as a servlet for this module to host. This bundle also hosts a web interface that diagnostic information about the Scheduling Server application.
- **Permissions –** This bundle provides a SOAP service that allows a user's permissions to be obtained.
- **Rig Provider –** This bundle provides a SOAP service that allows Rig Clients to connect and notify the Scheduling Server of rig statuses such as when they start up and shutdown. It also manages the connected rigs ensuring rigs that do not send timely status messages are taken offline.
- **Reports –** This bundle provides a SOAP interface to generate reports about rig use.
- **Queuer –** This bundle queues users to assign them to rigs using a priority queue. It also provides a SOAP interface to determine a user's session status and add or remove a user from the queue and determine the position of a user in the queue.
- **Bookings –** This bundle is the reservation system of SAHARA Labs. It also provides a SOAP interface that allows reservations to be created or cancelled.
- **Session –** This bundle handles sessions and ensures sessions are kept to the time and possibly extended if no other user is waiting to use the rig or a reservation is not starting for it. It also provides a SOAP interface that allows a users session details to be obtained such as remaining time and allows a user's session to be terminated if a user requests.

Another important part of the Scheduling Server that is critical to understanding its implementation is the database structure. This is an abridged database schema which omits irrelevant tables to federation (such as `'config'`) is shown below.

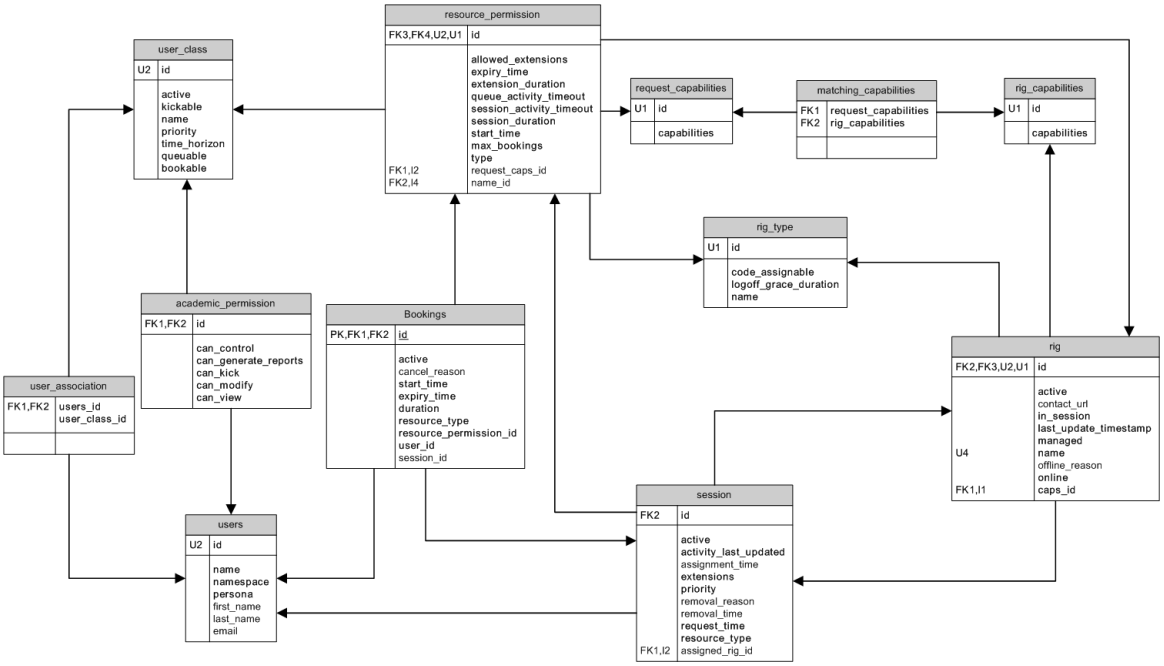


Figure 18 Scheduling Server database

The database structure, as the structure of persistent information stored in the Scheduling Server, shows some important relationships between concepts of the system. Users (‘`users`’ table) are directly associated with groups (‘`user_class`’ table) and groups are indirectly associated with permissions (‘`resource_permission`’ table) through group. Permissions define what may be accessed in terms of rigs which are actual rigs, rig types which are collections of rigs which are identical and *‘capabilities’* which are computed matches between requested ‘*tags*’ and rig declared ‘*tags*’. A user can only access a rig through permission, and as users (and groups) are consumer concerns and rigs are provider concerns, it is the permissions that will need to be distributed between consumers and providers.

The Scheduling Server actually queues, reserves and manages the remote laboratory and is the terminal destination for components that display, request or must be notified of such things. It is likely that the Scheduling Server requires the most implementation to develop the SAHARA Labs federation and this implementation will likely involve interfacing with the existing bundles that perform the functionality that is required to be distributed between consumer and provider: hosting of a communication service that provides duplex communication between consumer and provider; modifications to the underlying database structure to hold further permission constraints of multisite access; hosting of a user interface to mange multisite access. This will be further explored in the subsequent implementation chapters.

3.5 WEB INTERFACE

The Web Interface provides the user interface to the SAHARA Labs system and it is the place where a user authenticates, selects a rig to use, creates reservations, downloads data files and ultimately interacts with rigs. As diverse as the rigs are, so are the rig user interfaces (called *session* pages). Some session pages provide web mark-up generated controls such as sliders and push buttons with AJAX interaction with the rig. Others provide Java applets to launch a remote desktop session to provide a desktop application to interact with the rig, even others use Flash applets to provide the rig interface. In a multisite web interface, some mechanism will be needed to provide these customised interfaces so the user may interact in the same way as if the rig is hosted at the same remote laboratory and as the web interface is provided even if it is not. The following screenshots show some of the diversity of session pages:

Figure 19 UTS FPGA laboratory

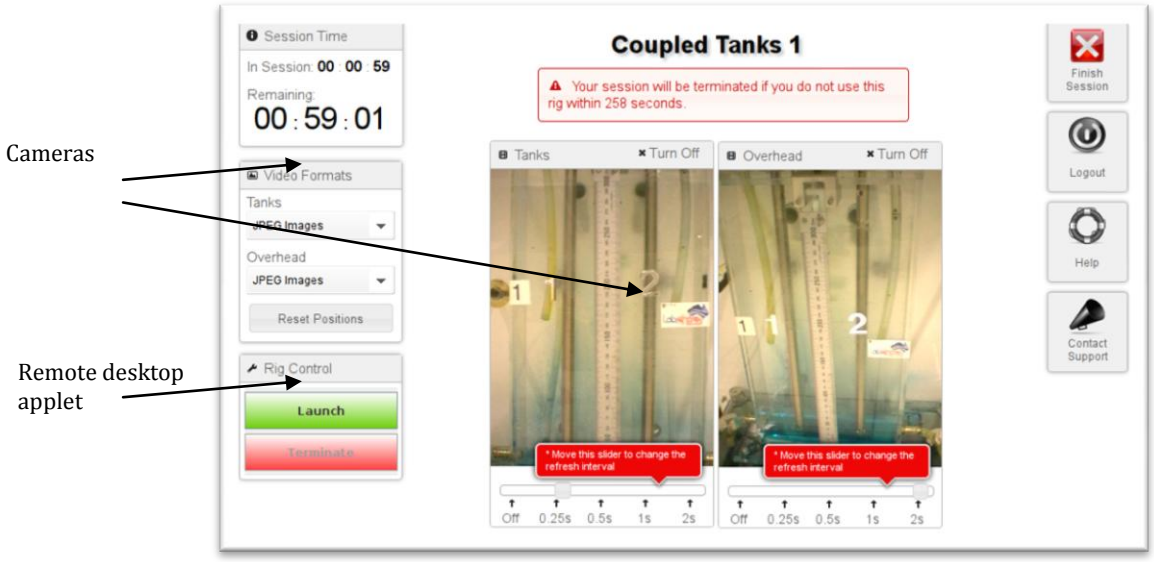


Figure 20 UTS Coupled Tanks laboratory

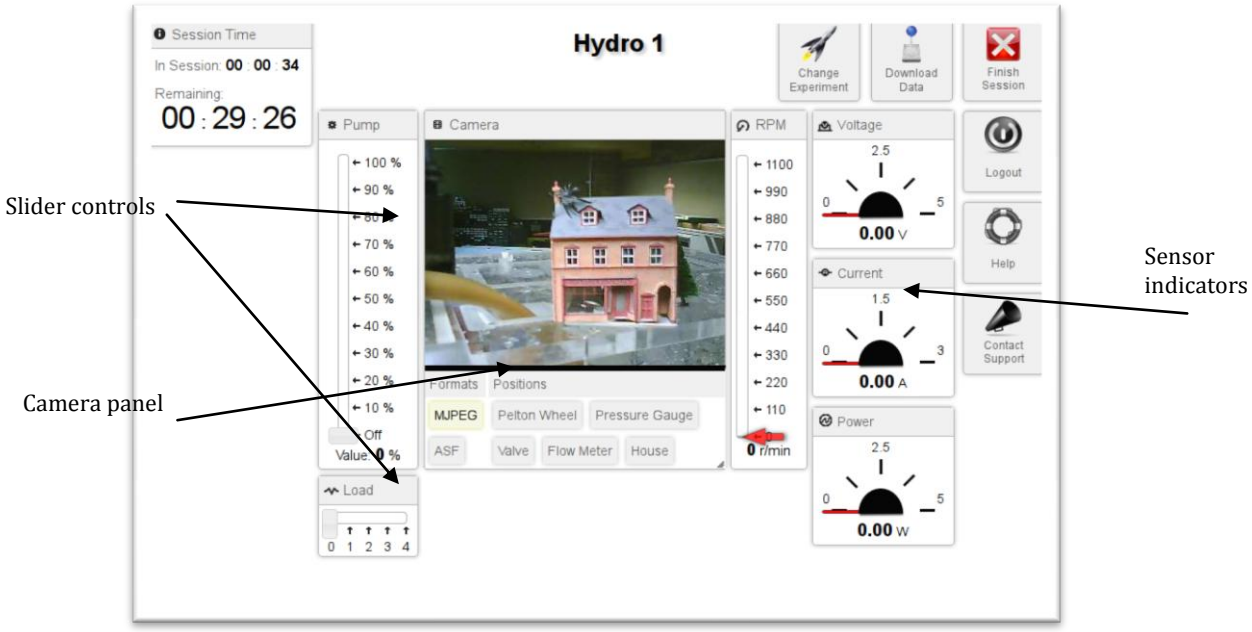


Figure 21 UTS Hydro laboratory

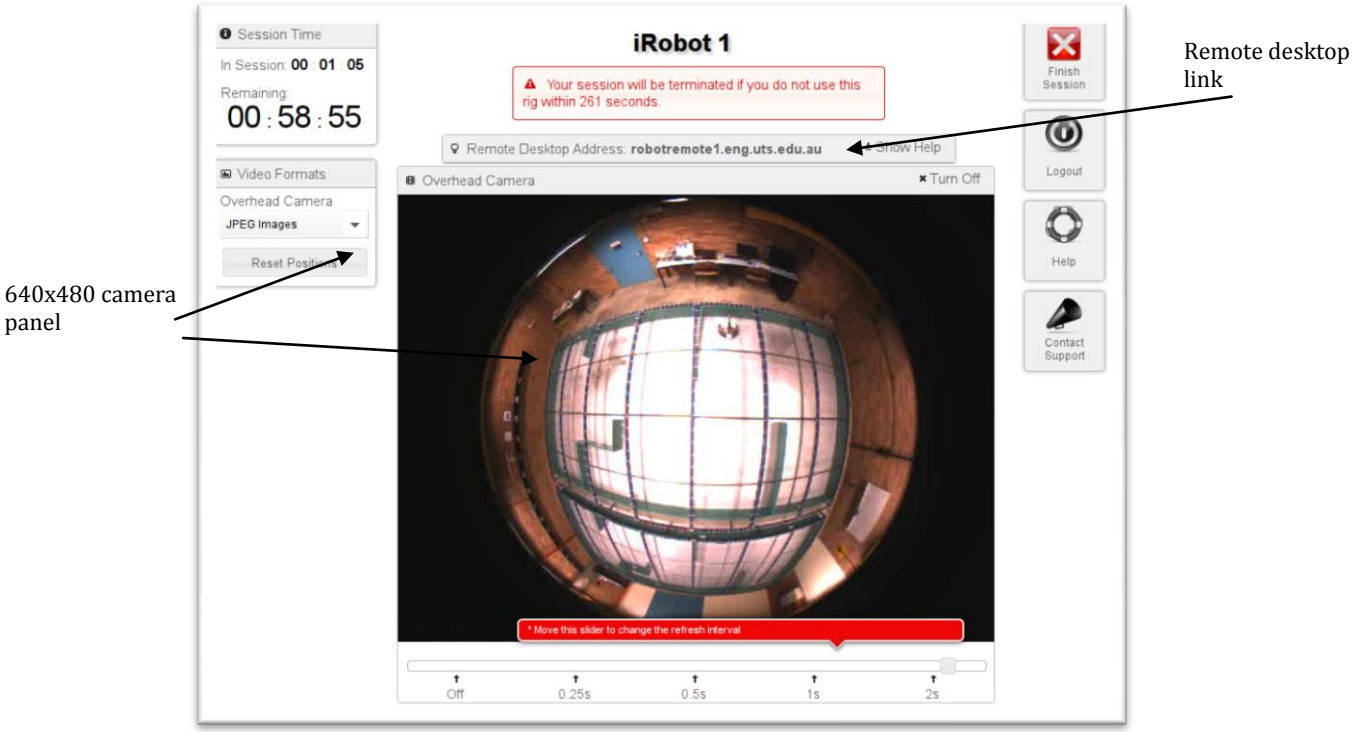


Figure 22 UTS iRobot laboratory

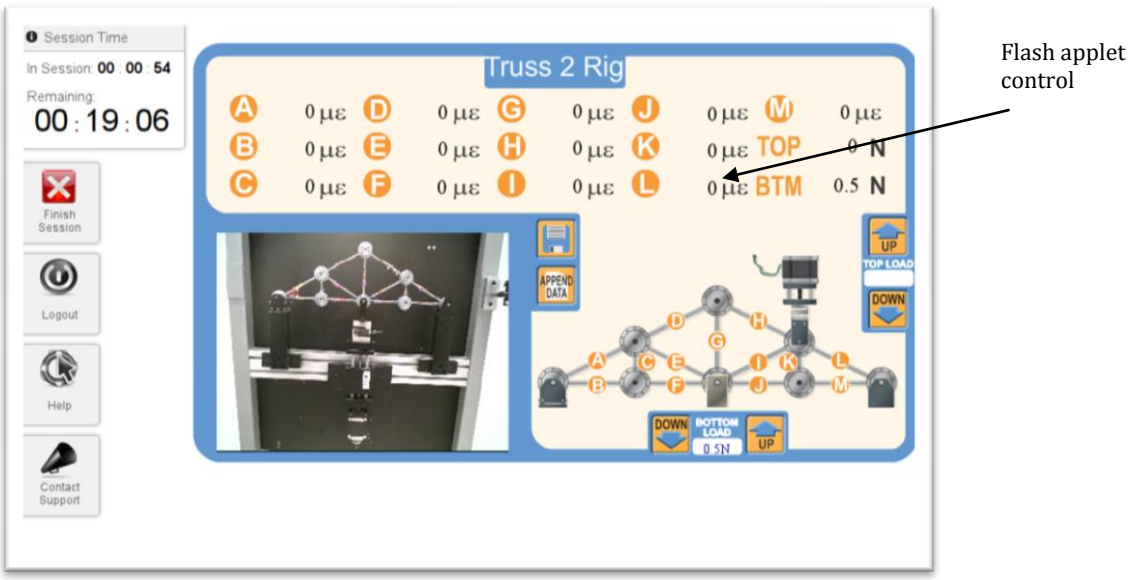


Figure 23 RMIT Truss laboratory

The Web Interface is a web application written in PHP using the Zend Framework. The top level folder structure is:

- `/application` – The PHP code that implements the logic and presentation of the web application. The code in this should be consistent for each SAHARA Labs deployment and it is not intended to be customised by SAHARA Labs implementers.

- `/institution` – A container for deployment specific code which includes the implementation customised rig type pages.
- `/library` – Library code including that includes the Zend framework. The code in this is intended for reuse in other parts of the Web Unterface, including the session pages but is not intended for modifications by SAHARA Labs implementers. Included here are implementations of *'session elements'* which are reusable web interface widgets that provide standard functionality. In the screenshots of existing session pages a few common interface themes are present such as camera panels and a Java applet that provides remote desktop functionality. These are examples of existing session elements.
- `/public` – The web server document root that holds static resources such as images, style sheets, JavaScript files and applets. If any resource is to be downloadable by web browsers, they must be located in this directory.

Session pages are divided into two parts, a standard session page template which adds common session content such as timers of session duration and remaining time and a button to finish the session and a customised portion. A blank session page without customised content is shown below:

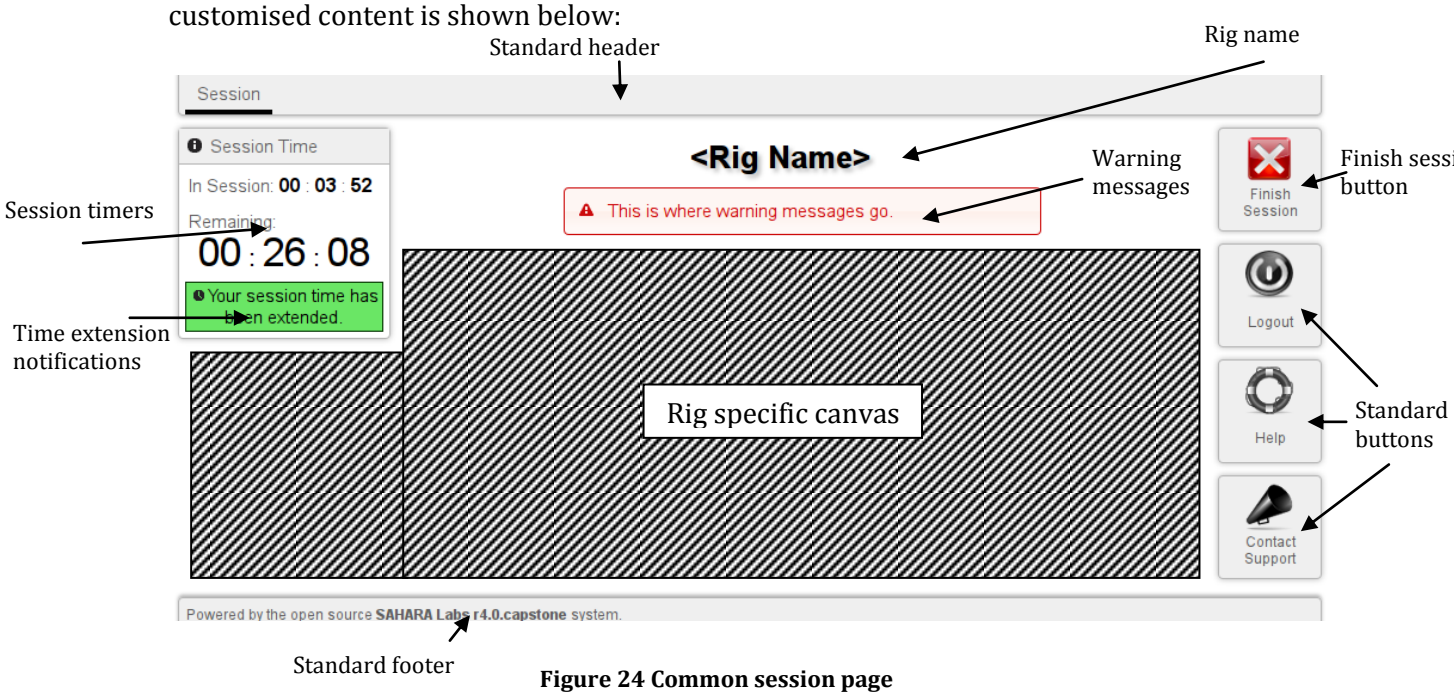


Figure 24 Common session page

To generate the session page, the common session page script file is rendered then a second script file specific to the rig type located in `'/institution/<namespace>/scripts/'` and called `'<rigtype>.phtml'` is rendered. In the location, `<namespace>` refers to a prefix configured by the web interface to segregate what is specific to that web interface and what is common among

SAHARA Labs web interfaces. When both scripts are rendered, a session page is generated. In a multisite scenario, it is plausible the second rig type specific script file along with additional static web resources is distributed across consumers to provide the same interface as the local Web Interface and this will be investigated further in the 'Lab Use Implementation' chapter.

3.6 Rig Client

The Rig Client is an application that manages a rig, abstracts the specifics of the rig from the rest of the remote laboratory and provides rig details and control during sessions. The Rig Client's management of a rig ensures life cycle events are propagated to the Scheduling Server so it does not allow the rig to be used if it is not operational and involves:

- Notifying the Scheduling Server when the rig has started up.
- Testing the rig and notifying the Scheduling Server when an error has been detected.
- Providing a keep-alive message to the Scheduling Server. This allows the Scheduling Server to determine whether the rig client computer or network communication has errors through the lack of these messages.

Rig Client life-cycle management only applies to the provider, so this does not need to be accounted for in the federation system.

During sessions the Rig Client is involved in the following ways:

- Setting up and revoking access. This is requested from the Scheduling Server and is used by the Rig Client to perform rig specific behaviours. An example of this is a rig which uses remote desktop to provide access to a Windows desktop application which controls the rig. In this case the Rig Client generates a local user account and adds that user to the '*Remote Desktop Users*' group so they may login. Revoking access deletes this user account so the user cannot login to use the application.
- Determining whether the rig is in use. This is requested by the Scheduling Server so it may terminate sessions of users who do not actively use the rig. Following the example in the previous point, the user account is checked to determine whether the user is logged into Windows.
- Determining specific details about the rig. These details may be static values from the rigs configuration or they may be dynamically generated from the

context of the rig and session such as the network details of the machine the Rig Client is running on. These details are requested by the Web Interface to generate the interface dynamically customised to the rig. Examples of this are the IP address of a remote desktop server to display to the user so they may know where to login to use the rig or a camera stream URL to provide to a video plug-in to display the rigs video.

- Providing a channel control to the rig. This channel is through a *front controller* pattern to allow Java classes implementing a special interface to be added to the Rig Client. During a session these are instantiated if requested and have certain methods following a special signature invoked with provided parameters. Object state is preserved during the session to allow stateful control to be developed. This is requested by the Web Interface to either directly control the rig or to perform auxiliary functions for the session interface. An example of an auxiliary function is a web interface which provides a list of rig generated data files that the user may download for offline use. To generate the display of this list of files the Web Interface requests the Rig Client to check a folder and provide a list of the files contained within. A further control channel request is to pass a selected file to the Web Interface so the user may download it.

In a federation system, the Scheduling Server requested session functionality only applies to a provider and need not be communicated across consumers and providers. As it is the providers Scheduling Server that decides session access and revocation the consumer need not and should not request it. Requests for this functionality are credentialed with a token the Rig Client is given by the provider's Scheduling Server so if a *misbehaving* consumer's Scheduling Server requests this it will not know the correct token and the request will not succeed. This token is provided by the Rig Client when it registers with a Scheduling Server on start-up. The token is also regularly changed by the Scheduling Server with a random frequency of not longer than 30 minutes to protect the Rig Client from misuse.

A consumer's Web Interface (assuming it is the component that provides the session interface) would need to request the rig configuration and control channel functionality of the Rig Client. However, this functionality is credentialed with the assigned user's name not the Scheduling Server token. This is a detail the consumer side of a federation should know and so the consumer Web Interface may successfully use the required Web Interface functionality if needed.

The differentiated credentials between the Rig Client and those that request it the Scheduling Server and Web Interface allow the Rig Client to be externally accessible for use in a federation with no modifications. The functionality that must be protected from external abuse(session access and revocation) is credentialed with a token only known internally and the functionality this is required for external use (rig configuration and control channel) is credentialed with a consumer details. This means the Rig Client as it stands is adequate for a federation system and does not need modification.

3.7 Extending the Existing Architecture for Federation Use

Investigating the existing architecture presents some options for developing a federation system with SAHARA Labs and ultimately *strongly* suggests how a federation system should be built within the existing architectural constraints. This includes which components require modifications and what pre-conditions must be established to allow federation use occur. The following diagrams show some possible architectural options the existing SAHARA Labs architecture presents:

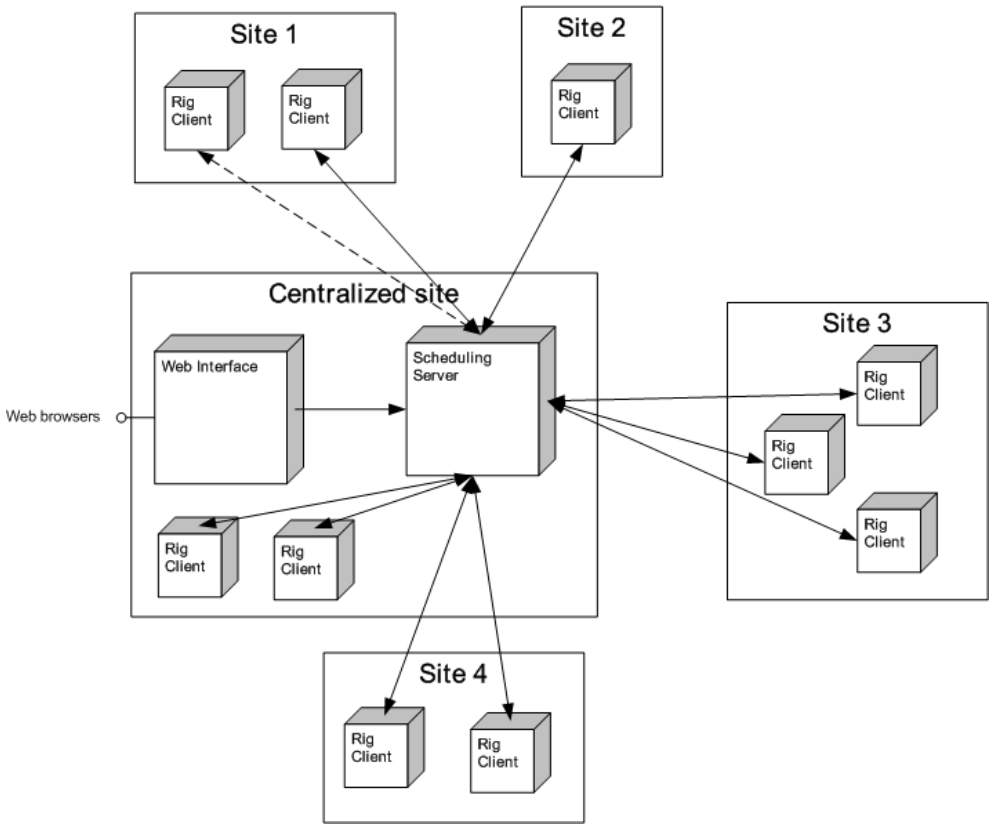


Figure 25 Architecture 1: Federation with one Scheduling Server

The diagram (architecture 1) above shows a 'federation' built with one Scheduling Server and Web Interface. This architecture can be immediately excluded because it is problematic in a few crucial ways. Only through the centralized site can the rigs be used. The other sites need to go to the centralized site to use their own rigs. By definition, a federation provides autonomy to each of its constituent members. This architecture is also problematic because of the centralized sites Scheduling server and Web Interface being a single point of failure. This is not a robust *federation* as it has a single federation wide point of failure. Also, as all use is centralized through a single server there may be a performance issue and does not horizontally scale out. This will not be further evaluated.

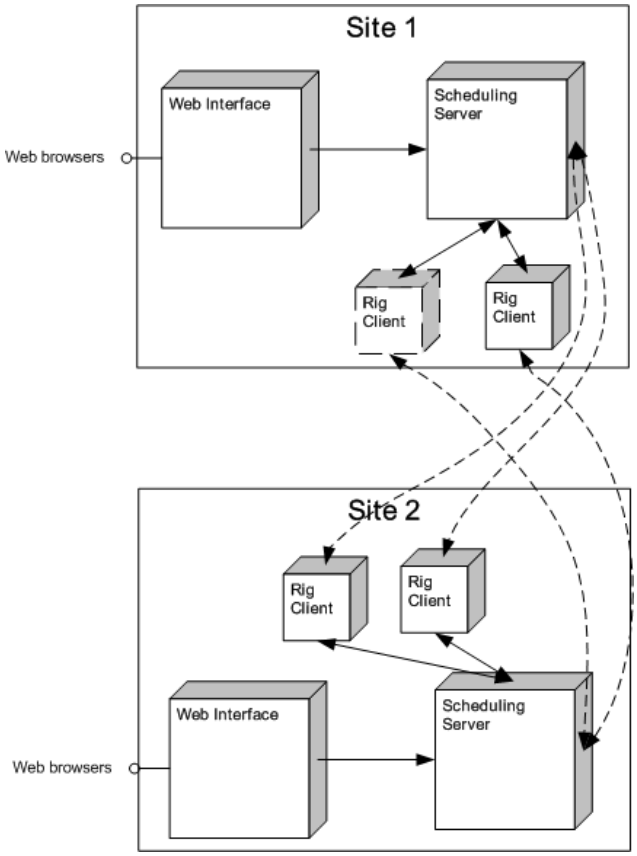


Figure 26 Architecture 2: Federation by multiple control of Rig Clients

The diagram above (architecture 2) adds the autonomy back to remote laboratory sites so they may be use independent of the federation. It shows the Rig Clients as the avenue for cross site communication. The obvious implication is the difficulty in ensuring the Rig Client is only in use by a single user at a time. This is *almost* untenable since two options to allow this are both less than ideal. If the scheduling is moved to the Rig Client, this will remove cross rig scheduling with rig types or tags and this means rig utilisation will be lessened and the scheduling will be less efficient. The other option is the Rig Client broadcasts when it has a user assigned to it to every other Scheduling Server in the federation. This obviously doesn't scale. Thus scheduling impossibility means an architecture which uses the Rig Clients to communicate across the federation sites is implausible. This will not be further evaluated.

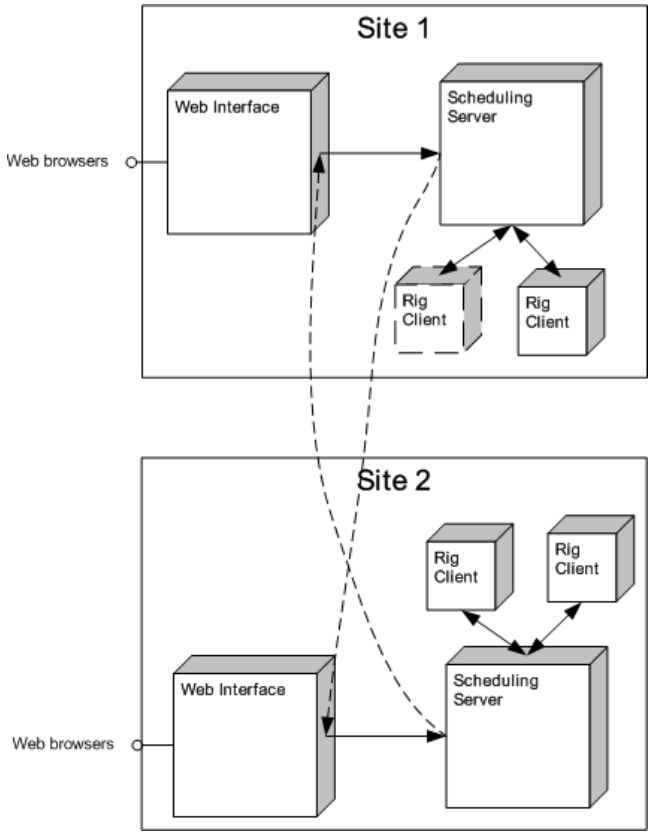


Figure 27 Architecture 3: Federation by Web Interface communication

The diagram above (architecture 3) shows an architecture in which communication between sites is through a consumer's sites Scheduling Server requesting provider functionality through their Web Interface. This is a plausible solution since it recognises it is the Web Interface that is the existing interface to the remote laboratory functionality and thus it proxies the Scheduling Server's functionality through an interface mediated by login credentials of a user. The existing interface programmatic interface to the Web Interface is a REST / JSON interface which is a tenable, if not a comfortable interface for a Java implemented Scheduling Server. However, this REST / JSON interface is not comprehensive as it was developed to provide AJAX style interaction on the Web Interface and things that are currently displayed as page loads are not provided. Operations that are provided by the REST / JSON interface are:

- Determining if a permission's resources are online and free.
- Queuing for a resource.
- Determining a resources free times for reservation.
- Creating a reservation.
- Cancelling a reservation.
- Determining session remaining use time and in use time.

- Terminating a session.

The operations that are not provided by the REST / JSON interface are:

- Determining whether the permission set the user is a member of. This is currently provided by a page load to the 'Rig Selection' page.
- Determining the list of existing reservations. This is currently provided as a page load to the 'Existing Reservation' page.

On the surface, this architecture exhibits no fundamental problems as architectures 1 and 2 do. The work required to implement this architecture would be:

- Synchronising users between consumer and provider as it is a users login credentials that provides the authentication to use the Web Interface provided REST / JSON interface.
- Implementing required operations that the Web Interface REST / JSON interface does not currently provide.
- Developing a Scheduling Server client that can call the Web Interface REST / JSON interface.
- Extending the current Scheduling Server permission structure so there is a permission that resolves to calling another site's Web Interface REST / JSON to perform things like queuing or reservations of that site's resources.

This architecture is essentially a *client – server* architecture and has the advantages and disadvantages of that architectural pattern. The client in this case is the Scheduling Server at the consumer and the provider's Web Interface is the server. The advantage of this architecture is the provider (server) knows very little of the consumer, only of its users and thus requires little modification from the existing SAHARA Labs software and so will probably be the easiest and most straight forward to implement. The disadvantage of this architecture is the provider knows very little of the consumer so it requires the consumer to poll for all state of users and sessions. This is just as a web browser would in a web environment. The consumer will need to continually poll the provider to determine whether the consumer's user has been assigned to rigs or has a reservation cancelled. Another advantage and disadvantage is whilst the communication is ultimately destined for the consumer's Scheduling Server it is proxied through a Web Interface. Putting a barrier between the destination and a source increases security (by definition) but incurs a performance cost.

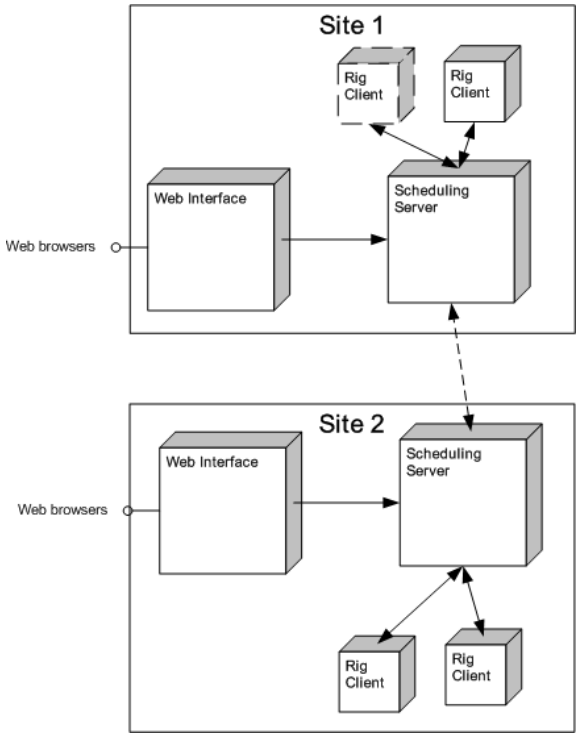


Figure 28 Architecture 4: Federation by Scheduling Server Communication

The diagram above (architecture 4) shows an architecture in which the communication is between Scheduling Servers. This is a plausible architecture as it recognises it is the Scheduling Server which is the ultimate destination of functionality required for a federation remote laboratory. If a user is requesting to use a rig it is the Scheduling Server that ultimately decides whether to queue the user and if and when they are assigned to a rig. The Scheduling Server already has a programmatic interface to provide this through a SOAP interface but this is inadequate for external consumption since it puts the onerous of authentication on the Web Interface (only the authorisation decision is at the Scheduling Server). To implement this architecture:

- A new federation service will need to be developed on the Scheduling Server that provides an interface to user level Scheduling Server operations such as queuing and session termination. This service could be authenticated at the user level as in architecture 3 or it may involve some site to site authentication through such things as certificates.
- Extending the current Scheduling Server permission structure so there is a permission that resolves to calling another site's Scheduling Server federation interface to perform things like queuing or reservations of that site's resources.

This architecture is essentially a '*peer-to-peer*' architecture with consumer and provider Scheduling Servers being peers of each other. The advantage of this

architecture is the provider knows about the consumer and so there may be duplex communication between consumer and provider. For example a consumer can request to put the user into the queue on the provider then wait for the provider to notify the consumer the user has been assigned to a rig.

Both architecture 3 and 4 are possible candidates for implementation. The table below evaluates the architectures using software architecture quality attributes in terms which best exhibits the attribute. This may be *strong*, *weak* or *neutral* where strong is one exhibits the attribute substantially more than the other; weak is one exhibits the attribute slightly more than the other; neutral is each exhibits the attribute the in a equivalent fashion.

Table 2 Architecture quality attribute evaluations

Quality Attribute	Architecture 3 – Scheduling Server to Web Interface	Architecture 4 – Scheduling Server to Scheduling Server	Best Exhibits
Performance **How much work a system can perform for a given amount of resource (Reekie 2006, p. 23)**	Does not. The consumer is required to poll the provider.	The provider Scheduling Server can push updates to the consumer.	Strongly 4
Usability **Consideration of the inter-links between software and the person directly interacting with it (Reekie 2006, p. 23)**	Does not. Polling introduces a lag between an event occurring and the next poll.	Pushed statuses are temporal at the time an event occurs.	Strongly 4
Reliability **How infrequently the system fails to perform its intended function (Reekie 2006, p. 23)**	Does not require as much state to be preserved between consumer and provider as it requires a consumer to pull through all details with a poll.	Requires state synchronisation between consumer and provider to correctly push updates.	Weakly 3
Security **Ability of a ensure that it can and will be used in the intended**	Has the Web Interface as the	Uses a customised interface	Strongly 4

way (Reekie 2006, p.23)	authentication barrier using user authentication details.	specifically developed for federation consumption.	
Maintainability **Ease with which the system can have defects corrected (Reekie 2006, p. 24)**	Reuses the existing Web Interface REST / JSON code which is production tested. Hard to upgrade as it is a PHP application with lots of separate files.	Requires new code to be written. Changes confined to Scheduling Server which as a Java application with installers that upgrade existing installations, it is straight forward to upgrade.	Weakly 4
Reusability **Ease with which software elements can be used in multiple contexts (Reekie 2006, p. 24)**	The Web Interface REST / JSON interface may be used by other applications.	Does not.	Strongly 3
Scalability **Ease with which the system can be scaled (Reekie 2006, p. 25)**	Does not.	Push updates eliminate unneeded poll communication.	Strongly 4.

Looking at the table of quality attributes, architecture 3 strongly exhibits one attribute and weakly exhibits one attribute. Architecture 4 strongly exhibits four attributes and weakly exhibits one attribute. The architecture that will be developed to implement federation use will be architecture 4.

4. Federation Rig Use Implementation

This chapter describes the implementation of the use portion of the SAHARA Labs system. *Use* is essentially logging to a local instance of SAHARA Labs, the consumer, to use a rig at another instance (provider). The use and behaviour of the rig across the consumer and provider divide should be equivalent as if the rig was hosted at the same instance as consumption.

The implementation of the SAHARA Labs federation use will also determine the preconditions and setup required to allow this use to occur.

4.1 Architecture in Detail

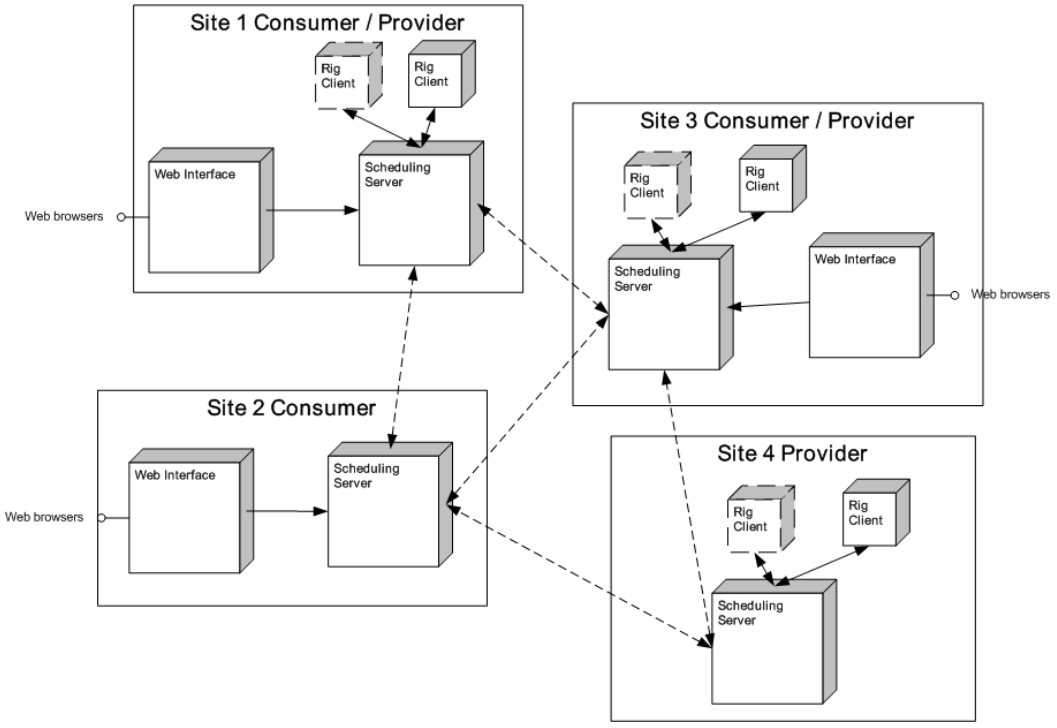


Figure 29 Use architecture

The diagram shows the architecture of the federation use of rigs. Each site is self contained and the sites rigs may be used without intervention by other external sites or entities. The diagram shows a basic demarcation of the types of sites that could exist in a federation. These are consumers which just consume rigs in the federation, providers which just provide rigs to the federation and consumer / providers which both consumer and provide rigs to the federation.

4.1 System Scenarios

The following scenarios show how remote laboratory use functionality is provided in this architecture. Each scenario is described in the format of:

- **Name –** the name of the scenario.
- **Goal –** the objective of the scenario.
- **Pre-conditions –** the conditions that must hold for the scenario to be executed.
- **Primary actor / Trigger –** the actor that initiates the scenario or a trigger to start a scenario.
- **Secondary actor –** the other actors involved in the scenario.
- **Step –** the list of steps to execute the scenario.
- **Alternative steps –** if a step is a condition and leads to alternative outcomes, these are listed here to branch from the condition step.
- **Notes –** any important notes that should be highlighted by the scenario.

Table 3 Scenario of viewing a user's permissions

Name	**Viewing a list of user's permissions.**
Goal	To go to the 'Rig Selection' page to view the permissions of the logged in user. These permissions include a provider access permission.
Pre-conditions	The user is logged in and is a member of a group that a provider access permission.
Primary actor	User
Secondary actors	Consumer Web Interface Consumer Scheduling Server
Steps	1. The user navigates to the 'Rig Selection' page. 2. The consumer Web Interface requests the provider Scheduling Server to provide the specified users permissions. 3. The consumer Scheduling Server checks the user's record and its associations with user classes. The permissions contained in the user classes the user is a member of are returned. 4. The Web Interface renders the 'Rig Selection' page and returns it to the requesting browser.
Alternative steps	None.
Notes	• This is exactly the same list of steps for the equivalent scenario as the existing SAHARA Labs system. The implicit difference is that there is a permission type that specifies the permission is for access of a provider rig.

	• The provider is not involved in this scenario. Only the consumer is as the consumer stores the user and their access to groups.

Table 4 Scenario of determining provider resource status

Name	Determining provider resource status
Goal	Determining whether a provider's resource is online and free from use.
Pre-conditions	The user has a permission which provides access to a provider's resource.
Primary actor	User
Secondary actors	Consumer Web Interface Consumer Scheduling Server Provider Scheduling Server
Steps	1. The user clicks a button on the 'Rig Selection' page that specifies a provider resource. An AJAX request is made to the consumer Web Interface to determine the status of the permission providing the permission identifier. 2. The consumer Web Interface makes a request to the consumer Scheduling Server to determine the status of the permission providing the permission identifier. 3. The consumer Scheduling Server determines the permission is a provider permission and makes a request to the provider Scheduling Server to determine the status of the permission providing the permissions identifier and the identifier of a consumer. 4. The provider Scheduling Server validates the permission exists and the consumer is allowed to access it. 5. The provider Scheduling Server checks the status of the permission mapped rigs and returns these to the consumer Scheduling Server. 6. The consumer Scheduling Server returns the status to the consumer Web Interface who returns it back to the requesting browser to update the 'Rig Selection' page

	with the provided resource rig statuses.
Alternative steps	From 3. If the request fails to be made to the provider a provider communication failure error is returned and the scenario continues at step 6. From 4. If the provider fails to find the permission that is identifier or that the consumer does not have access to the permission, an error is returned. The scenario continues at step 6.
Notes	• Permissions referring to the same resources must be synchronized between consumer and provider and they must be commonly identified using some form of a credential. • The provider must be able to uniquely identify the consumer using some form of a credential.

Table 5 Scenario of creating a reservation

Name	**Creating a reservation**
Goal	The user creates a reservation for a provider's resource.
Pre-conditions	The user has a permission which provides access to a provider's resource.
Primary actor	User
Secondary actors	Consumer Web Interface Consumer Scheduling Server Provider Scheduling Server
Steps	1. The user clicks the 'Reserve' button on the 'Rig Selection' page that selects to reserve a provider resource. The web browser navigates to the 'Reservation' page. 2. The consumer Web Interface makes a request to the consumer Scheduling Server to determine the list of bookable times for the permission. 3. The consumer Scheduling Server determines the permission is a provider permission and makes a request to the provider Scheduling Server to determine the list of bookable times for the permission.

	4. The provider Scheduling Server validates the permission exists and the consumer is allowed to access it. 5. The provider Scheduling Server checks it's booking system to obtain the list of free times for the permission's resource and returns them to the provider Scheduling Server. 6. The consumer Scheduling Server returns the status to the consumer Web Interface who returns it back to the requesting browser to render the 'Reservation' page with the times the resource can be reserved. 7. The user selects a time to create a reservation. This makes a request to the consumer Web Interface for the permission and selected time. 8. The consumer Web Interface requests the consumer Scheduling Server to create the reservation with the user's credentials, permission and selected time. 9. The consumer Scheduling Server determines the permission is a provider permission and makes a request to the provider Scheduling Server to create a reservation with the users details, permission and selected time. 10. The provider Scheduling Server validates the permission exists and the consumer is allowed to access it. 11. The provider Scheduling Server checks the user exists. 12. The provider checks the permission's resource is free for the selected time and creates the reservation. 13. The outcome of the reservation is returned back to the user through the consumer Scheduling Server and consumer Web Interface.
Alternative steps	From 3. If the request fails to be made to the provider a provider communication failure error is returned and the scenario ends. From 4 and 10. If the provider fails to find the permission that is identifier or that the consumer does not have access to the

	permission, an error is returned. The scenario ends. From 11. If the user does not exist at the provider, the user is created by the provider. From 12. If the selected time is determined by the provider as not being free, a list of alternative times is returned to the user.
Notes	• The list of reservation times for a resource is only kept by the provider. A consumer only requests this from a provider and does not make any assumptions on whether a resource is free. • A user is *lazy created* at the provider end by declaration of the consumer. That is, the consumer specifies a user and the provider checks whether they exist. If not, the user is created.

Table 6 Scenario of accessing a rig through queuing

Name	**Accessing a rig through queuing**
Goal	The user attempts to access a rig through queuing.
Pre-conditions	The user has a permission which provides access to a provider's resource.
Primary actor	User
Secondary actors	Consumer Web Interface Consumer Scheduling Server Provider Scheduling Server Provider Rig Client
Steps	1. The user clicks on the 'Queue' button on the 'Rig Selection' page that specifies to queue for a provider resource. An AJAX request is made to the consumer Web Interface to queue the user for the selected resource. 2. The consumer Web Interface makes a request to the consumer Scheduling Server to queue the user. 3. The consumer Scheduling Server determines the permission is a provider permission and makes a request to the provider Scheduling Server to queue the user. 4. The provider Scheduling Server validates the

	permission exists and the consumer is allowed to access it. 5. The provider Scheduling Server checks the user exists. 6. The provider Scheduling Server checks whether the rig in the resource identified by the permission is online. 7. The provider Scheduling Server checks whether the rig in the resource identified by the permission is not free. 8. The provider Scheduling Server adds the user to the queue and returns the queue position to the consumer Scheduling Server. 9. The consumer Web Interface receives the queue position from the consumer Scheduling Server and navigates to the queuing page. 10. The user waits in the 'Queuing' page which periodically polls the consumer Scheduling Server to determine whether the user has been assigned to the rig. 11. When the user is assigned to a rig, the provider Scheduling Server assigns the user to the rig's Rig Client. 12. The provider Scheduling Server contacts the consumer Scheduling Server and notifies it that the user has been assigned to a rig and provides the details of the rig and its type. 13. The consumer Scheduling Server checks whether the rig and associated rig type already exist and creates it if it does not. 14. The next poll from the 'Queuing' page specifies the user has been assigned to a rig and the page navigates to the 'Session' page. 15. The 'Session' page is displayed with the correct rig type session page of the assigned rig.
Alternative steps	From 3. If the request fails to be made to the provider a provider communication failure error is returned and the scenario ends. From 4. If the provider fails to find the permission that is identifier or that the consumer does not have access to the permission, an error is returned. The scenario ends.

	From 5. If the user does not exist at the provider, the user is created by the provider. From 6. If the all the rigs in the permission's resource is offline, an error is returned and the scenario ends. From 7. If a rig in the permission's resource is free, the user is immediately assigned to a rig and the scenario continues at step 13.
Notes	• The status of a provider's resources rigs is kept at the provider. • The queues for resources are kept at the provider. • Notifying the consumer side of a users being assigned is a call-back. • Details of providers rigs and rig types are also kept at the consumer to provide session details. • The interface to the rig is hosted at the consumer side.

Table 7 Scenario of accessing a rig through reservation

Name	**Accessing a rig through reservation**
Goal	The user attempts to access a rig through reservation.
Pre-conditions	The user has a permission which provides access to a provider's resource. The user has created a reservation for a provider's resource using scenario 'Creating a reservation'.
Primary actor	User
Secondary actors	Consumer Web Interface Consumer Scheduling Server Provider Scheduling Server Provider Rig Client
Steps	1. The user logs in within 30 minutes of their reservation start time and are directed the 'Reservation waiting' page. The user is directed to this page because the consumer Scheduling Server has the user's reservation record. The user waits at this page and the page polls the consumer Scheduling Server to determine when the reservation has started.

	2. When the reservation is scheduled to start, the provider Scheduling Server assigns the user to the rig's Rig Client. 3. The provider Scheduling Server contacts the consumer Scheduling Server and notifies it that the user has been assigned to a rig and provides the details of the rig and its type. 4. The consumer Scheduling Server checks whether the rig and associated rig type and creates it if it does not. 5. The next poll from the 'Reservation Waiting' page specifies the user has been assigned to a rig and the page navigates to the 'Session' page. 6. The 'Session' page is displayed with the correct rig type session page of the assigned rig.
Alternative steps	From 2. If all the rigs in the reserved resource are offline, the consumer Scheduling Server is notifying the reservation cannot be redeemed and has been cancelled. The scenario ends. From 3. If the provider Scheduling Server fails to contact the consumer Scheduling Server the reservation is cancelled.
Notes	• The consumer Scheduling Server keeps a record of a user's reservations. • The reservation redemption occurs at the provider site. • The communication between the provider and consumer to notify the consumer a reservation has started is a call-back initiated from the provider. • The consumer is waiting for the provider to notify them about a reservation starting. If this does not occur within a reasonable amount of time, the consumer will need to deal with the eventuality that the provider or the communication with the provider is down.

Table 8 Scenario of finishing a session

Name	**Finishing a session**
Goal	An in progress session from a provider's rig is terminated and the provider's rig is released for use by another user.
Pre-conditions	The user has a session using a provider's rig.

Primary actor	User
Secondary actors	Consumer Web Interface Consumer Scheduling Server Provider Scheduling Server Provider Rig Client
Steps	1. The user clicks the 'Finish Session' button on the 'Session' page that specifies to finish the session the user is in.. An AJAX request is made to the consumer Web Interface to terminate the session. 2. The consumer Web Interface makes a request to the consumer Scheduling Server to finish the session. 3. The consumer Scheduling Server determines the permission is a provider permission and makes a request to the provider Scheduling Server to finish the session, providing the name of the user. 4. The provider Scheduling Server validates the permission exists and the user has a session with it. 5. The provider Scheduling Server terminates the session and removes the user from the assigned rig's Rig Client. 6. The provider Scheduling Server returns to the consumer Scheduling Server that the session has been finished. 7. The consumer Scheduling Server terminates the locally mapped session for the user. 8. The consumer Scheduling Server returns to the consumer Web Interface that the session has finished. 9. The consumer Web Interface redirects the user back to the 'Rig Selection' page.
Alternative steps	From 4. If the specified user does not have a session, the provider Scheduling Server returns an error back to the consumer Scheduling Server. The scenario continues at step 7.
Notes	• There is a session mapped at the consumer side which is synchronised by the canonical provider side session which holds the in use rig. Both are terminating when terminating a session.

Using the scenarios a number of details about the architecture are illustrated. These include how communication occurs with what parameters, what must be synchronised between consumers and providers, where things occur (such as the rig interface) and when they occur and how things recover from errors.

The communication from the consumer Web Interface is predominately to the consumer Scheduling Server with the only exception being the provider Rig Client during sessions. This communication is *exactly* the same as the communication as the existing SAHARA Labs system and uses the same interfaces. However, new parameters may need to be able to be introduced to these interfaces to discriminate with things such as provider permissions and the different sorts of errors that may occur in federation use such as communication failures. Whether there is in fact discrimination is an interface issue. The actual determination that a permission is a provider permission occurs solely within the consumer Scheduling Server and the Web Interface remains agnostic to this decision.

Permissions are used to identify what a user has access to and the constraints around this access. This is a precedent from the existing SAHARA Labs system. In federation use, these permissions need to synchronized between consumer and provider. As the identifier of a user's access, a credential scheme will need to be developed to map the permissions during communication between consumer and provider. The table below shows the fields in the existing permission structure of SAHARA Labs (`resource_permission` table) and what will need to be synchronised.

Table 9 Permission fields

Field	Definition	Synchronized	Reason
Display name	A textual name to annotate the interface. It is used to given a name to the permission buttons on the 'Rig Selection' page.	No	This value is used for the user interface and only applies to the consumer side where the interface resides.
User class	The class the permission belongs to.	No	User classes are only a consumer side concept.
Start time	The date and time when a permission may start to be used. This sets the earliest bound for a	Yes	Permission validation must occur at both the consumer side and provider side. The consumer Web Interface also

	reservations start time.		specifies which permissions have started, waiting to start or are finished.
Expiry time	The date and time when a permission may no longer be used. This sets the latest bound for a reservation end time.	Yes	Permission validation must occur at both the consumer side and provider side. The consumer Web Interface also specifies which permissions have started, waiting to start or are finished.
Type	The type of permission which specifies the resource the permission resolves to. This is an enumeration of: • TYPE – rig types • RIG – specific rigs • CAPS – rig tag list.	No	Fundamentally a consumer permission and provider permission resolve to different resources. A provider permission resolves to actual rigs, whether they be specific rigs, rigs in a rig type or a rig with a specific tag but a consumer permission resolves to a permission at a provider which then resolves to a rig. This means a new enumeration will need to be added called 'CONSUMER' for this type of permission.
Rig	The rig of this permission if the type of permission is for a rig.	No	Resolving permission to rig only applies to the provider.
Rig type	The rig type of this permission if the type of permission is for a rig type.	No	Resolving permission to rig only applies to the provider.
Request capabilities	The list of tags requested of this permission if the	No	Resolving permission to rig only applies to the provider.

	type of permission is for a list of tags.		
Maximum bookings	The number of maximum concurrent reservations a user may make.	Yes	Validation of the number of concurrent bookings occurs on both consumer and provider side. It is also displayed on the consumer Web Interface 'Reservation' page.
Session duration	The guaranteed duration of a session when the user is assigned using the queue. This with the addition of time extension time forms the upper bound on the duration of reservations.	Yes	Whilst the determination on when to terminate an elapsed session occurs on the provider side, the display of session timers is on the consumer Web Interface.
Allowed extensions	The number of time extensions that may be given to a user if the rig is free from subsequent.	Yes	Whilst extending a session occurs on the provider side, the display of session timers is on the consumer Web Interface.
Extension duration	The length of time each time extension.	Yes	Whilst extending a session occurs on the provider side, the display of session timers is on the consumer Web Interface.
Queue activity timeout	The timeout after which to remove the user from a queue if a 'Queuing' page ping is not received.	No	Checking stale queue sessions occurs only on the consumer side.
Session activity timeout	The timeout after which to remove the user from a rig session if no activity is detected for using the	Yes	Whilst removing stale rig sessions occurs at the provider side, the warning messages on the consumer

	session's rig.		Web Interface.
Use activity detection	Whether activity detection is enabled for the permission. Activity detection terminates a session if the user is not actively using the session within a period of time.	Yes	Whilst terminating a session occurs on the provider side, the display of activity detection occurs on the consumer side Web Interface.

As can be seen in the above table, multiple permission fields are synchronised. The permission type and fields that map to what the permission is providing (rig, rig type or request capabilities) are not synchronized since they only apply to the provider. This alludes to the dichotomy in permissions between consumers and providers. Fundamentally consumer permissions involve getting to a provider, provider permissions involve getting to a rig. Ultimately, both get to a rig with the consumer permissions through the synchronized provider permission and this is what the user uses.

The scenarios also show reservations and sessions are synchronized between consumer and provider. This is advantageous for a few reasons. First, the SAHARA Labs interface coerces a user between specific pages dependant on their temporal state in the system. If they are using the session, they must be on the 'Session' page, not being on this page means they are not using the rig and this wastes a finite resource (rig time). If the user is queuing, they are on the 'Queuing' page and are waiting to be assigned to a rig. If they are queuing but not on this page, the 'Queue Waiting' page ping is not received by the system and after a period of time the user is removed from the queue. Removing a not present user from the queue means they are not assigned to a rig saving a finite resource from being wasted whilst another user may be waiting for it. To illustrate why mapped reservations and sessions are needed to ensure correct user state and position in the system consider the case on not having these synchronized. If they were not, each site in the federation would need to be polled to determine whether a user is in session, the queue or waiting for a reservation to start. This doesn't scale as this means the more sites that are added to the federation the more sites that need to be polled to determine a user's state. With synchronised sessions (when the user us in the queue they have a session as well as when they are assigned to a rig) and

reservations allows the consumer to determine the correct state of the user without the intervention of any other site.

Having mapped sessions and reservations in combination with mapped permissions allows the consumer to provide session timings to a user without having to communicate with the provider. The consumer can use the formula below to determine this:

In the above formula, duration, allowed extensions and extension duration are known by the consumer as it is stored in the synchronized permission and assignment time and used extensions are known by the consumer as it is stored in the synchronized session. With these fields the consumer can determine the user's remaining time in session. The Web Interface requests session information every 30 seconds so that it is not having to having to communicate with the provider to determine session information which saves many superfluous provider requests being made. The state chart below shows the possible user states. Only when a transition occurs does the provider need to update the consumer of the change of a session variable. For example, if a time extension is given to the user's session, the provider will notify the consumer so it may keep the synchronized sessions consistent and thus the remaining time calculations accurate.

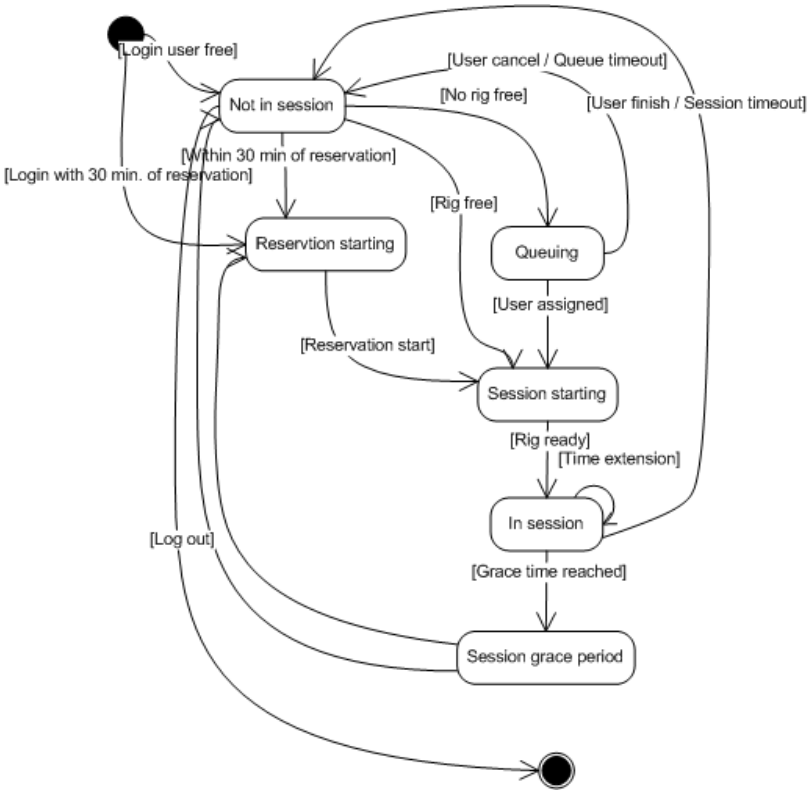


Figure 30 User states

In the state chart, the transition from the 'Not in session' to the 'Reservation starting' and 'Queuing' states are by the consumer (the consumer's user selects to create a reservation or queue) and the rest to transitions are by the provider, except for the user cancel transitions. The provider needs to notify the consumer of the following events to keep the consumer session and reservation mappings consistent:

- Reservation start – A user's reservation has been assigned to a rig. The consumer will create an assigned session.
- User assigned – A user has exited the queue and been assigned to a rig. The consumer will modify the corresponding consumer session to be marked as assigned.
- Rig ready – The rig has finished having the user assigned and is ready for use. The consumer will modify the corresponding consumer session to be marked as ready for use.
- Time extension – The session has had one of its sessions time extensions used. The consumer will modify the corresponding session to decrement the number of remaining uses.
- Grace time reached – The session has reached the grace which specifies no further time extensions will be granted and the session will be terminated

when the remaining time expires. The consumer will modify the corresponding consumer session to be marked as in grace.

- Session finished – The session has been terminated. The consumer will mark the session as finished.

The scenarios show rigs are mapped between the consumers and providers but only at session time. The pertinent information about rigs the consumer must know is the address to communicate with the rigs rig client and the rigs type. The rig type is used to correctly deploy the customised 'Session' page as this is defined by the rig's type. The rig client address is used by the 'Session' page to determine rig details or interface with the rig. The mapping of rigs is *lazy* as it is only needed during session. On first use, the mapping is performed with provider provided details or if the mapping already exists the details are updated.

The last thing the scenarios show is the relationship of users between consumer and providers. Users from a consumer are supplied to providers at such time that they need to queue a user or create a reservation. The *user* supplied by the consumer need not have a credential that uniquely identifies a physical user such as consumer identity (such as their name or identity management user name). The only requirement is the user's credential which uniquely identifies a user between sessions and reservations. If a consumer does not want a user to be tracked at the provider it may choose to change the credential or use different credentials across providers. In this capstone, this has not been implemented but may be without altering the architecture this capstone implements. The mapping between users and user classes is purely on the consumer side. The provider is unconcerned with user to user class relationships and any user a consumer supplies is treated as valid and allowed to use the permission requested provided the consumer can use the permission requested. Codifying this, the authorisation between users and consumers is user class associations, as the existing SAHARA Labs precedent and the authorisation between user and provider is consumer declaration.

.2 SCHEDULING SERVER MODIFICATIONS

To implement the federation use modifications for the Scheduling Server involves the following steps:

1. Deciding modifications to the existing application architecture that adds the required functionality into the Scheduling Server. The Scheduling Server is developed as OSGi bundles. This step is deciding new modifications that are needed to be developed to implement federation SAHARA Labs use. The federation bundle is called the *Multisite* bundle.
2. Modifying the persistence of the Scheduling Server to store the required details for federation use. As the Scheduling Server uses a relational database as persistence, this is modifying the database schema and the associated Java Persistence Architecture (JPA) mappings.
3. Developing a communication interface between consumers and providers. The existing SAHARA Labs communication interfaces use SOAP and so will this interface.
4. Modifying the existing bundles so they export required services that the Multisite bundle requires. The OSGi framework has a service layer which allows *services* to be registered in a bundle and consumed in another while preserving the implementation details of the service within the registering bundle. A service in OSGi framework is a Java POJO, a plain-old-Java-object registered for consumption with its class object or the class object of its base interface.
5. Adding an event notifier for sessions events so these events may be listened on to provide call-backs from providers to consumers when these events occur. The mechanism that allows this to occur is using the *whiteboard* pattern. From the OSGi documentation the whiteboard pattern is:

> The whiteboard pattern leverages the OSGi framework service registry instead of implementing a private registry. Instead of having event listeners track event sources and then register themselves with the event source, the whiteboard pattern has event listeners register themselves as a service with the OSGi framework. When the event source has an event object to deliver, the event source calls all event listeners in the service registry (OSGi Alliance 2004, p. 5).

The modules that require notification of a session event registers an implementation of a session event listener interface with the OSGi framework

service registry. The modules that generate session events (such as the 'Queuer' bundle which assigns users to rigs) invokes a method on all the registered session event listener interfaces in the service registry providing notification an event has occurred.

6. Implement the federation use service.
7. Implement the federation use client and extend the permission processing of existing modules so they call the federation use client when a consumer permission is requested.

.2.1 APPLICATION ARCHITECTURE MODIFICATIONS

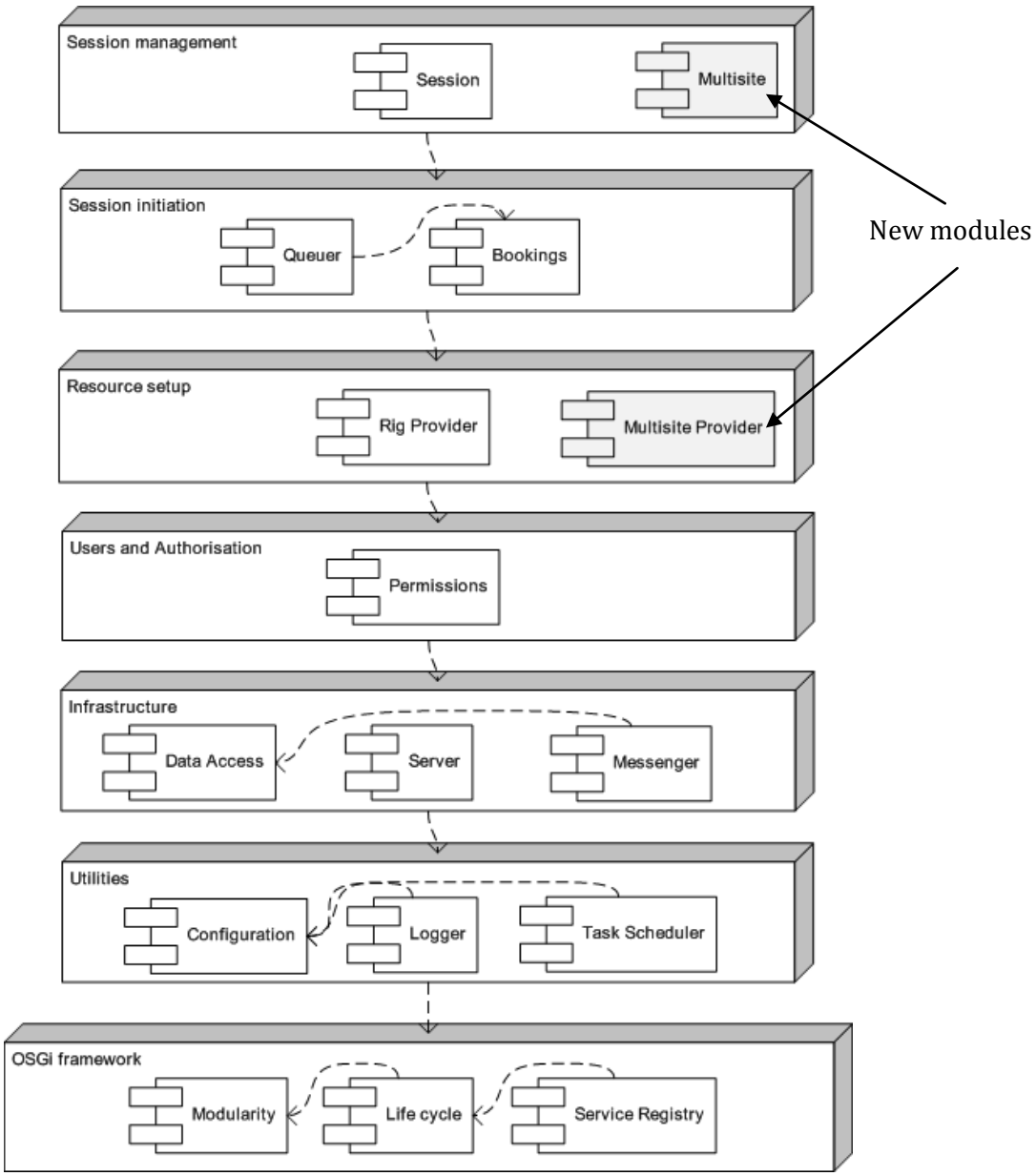


Figure 31 Multisite Scheduling Server application architecture

The diagram above shows a modified version of the Scheduling Server application architecture with the addition of two extra modules. These are:

- **Multisite** – This implements the provider side of the federation system and as such implements the federation external interface. It is responsible for allowing consumers to connect to the provider system and queue, reserve and perform all other federation use functionality. It also listens on session events so if a session event occurs and the session is from a consumer, the consumer is notified of the session event that has occurred. This module sits at the top level of the Scheduling Server layers as it consumes services provided by lower level

modules such as 'Queuer' and 'Bookings' to perform queuing and reservation respectively among other things.

- **Multisite Provider** – This implements the consumer side of federation use and provides a client to the corresponding federation use external interface exported by the 'Multisite' module hosted at a provider. The nomenclature of *provider* in this modules name is because it provides internal functionality to higher level modules in the Scheduling Server for when they must process a federation permission. This is equivalent to the 'Rig Provider' module in the case of local rig permissions.

The relationship between the 'Multisite', 'Multisite Provider' and existing modules can be seen in the following quick and dirty sequence diagram of queuing a user.

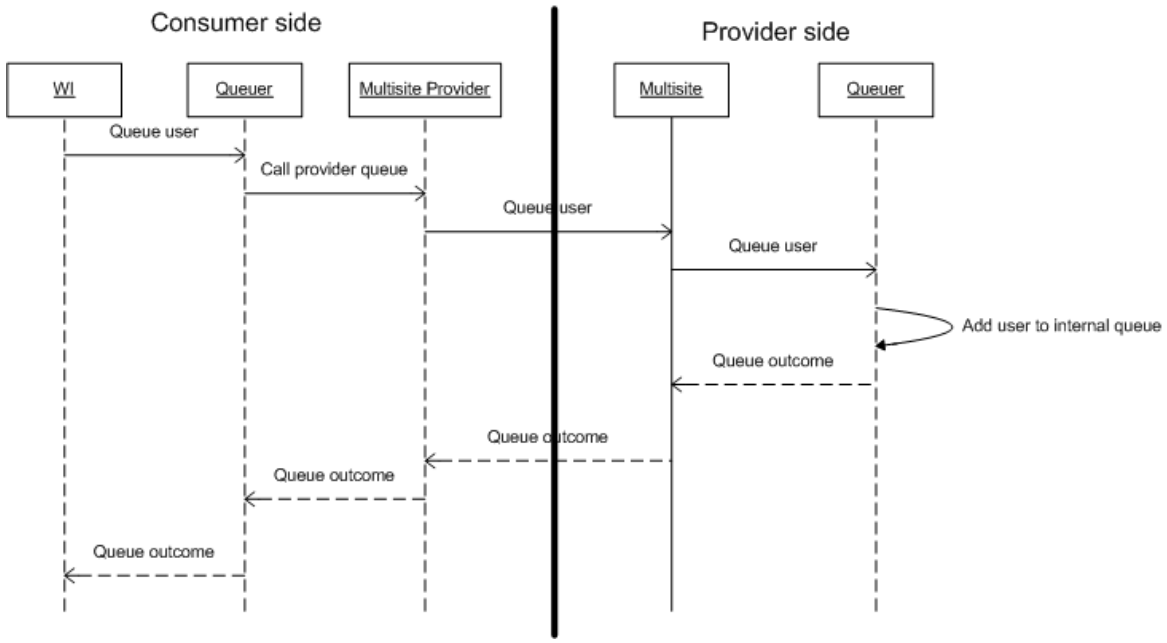


Figure 32 Queuing a user across the federation

The diagram shows the existing 'Queuer' module uses the 'Multisite Provider' to remotely queue a user at a provider and the 'Multisite' module uses the 'Queuer' module to queue a remotely provider user from a consumer. In theory, nothing prevents the provider 'Queuer' calling another provider to queue the user, being effectively a man-in-the-middle between the consumer and intended provider. However, this scenario is unintentional and is not intended to actually be used because it adds a point of failure and a performance bottle neck.

.2.2 PERSISTENCE STRUCTURE

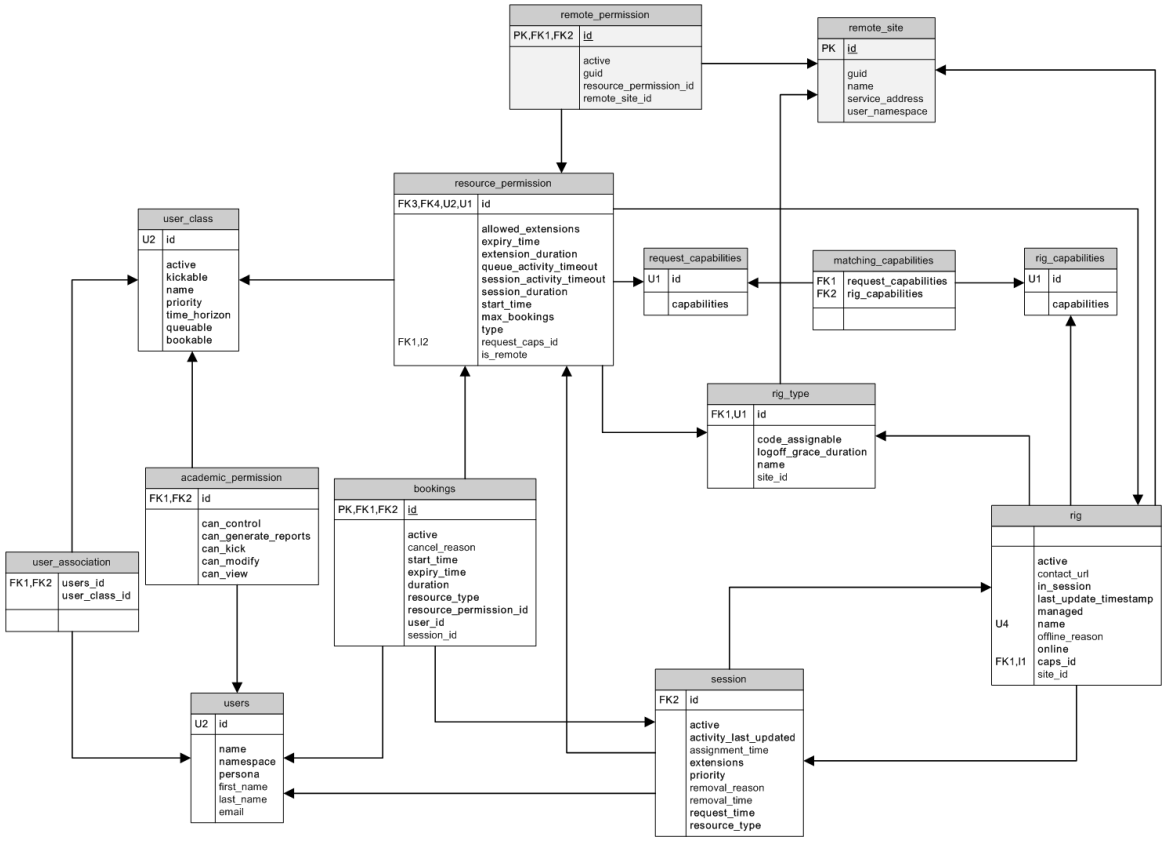


Figure 33 Scheduling Server schema with federation use persistence changes

The diagram above shows an updated database schema for the Scheduling Server. The changes are the addition of two tables `remote_permission` and `remote_site` and additions of fields to the tables `resource_permission`, `rig` and `rig_type`.

The `remote_site` table stores details about other sites in the federation this site has a relationship with. The fields in this table have the meanings shown in the table below:

Table 10 remote_site table

Field	Data type	Definition	Optional
id	bigint(20)	The records primary key.	No
guid	varchar(32)	The credential that is supplied to uniquely identify the remote site.	No
name	varchar(255)	The textual display name of the remote site that is used as part of the user interface if details of the remote site must be shown.	No
service_address	varchar(1024)	The address of the endpoint of the	No

		remote site Multisite SOAP service.	
user_namespace	varchar(255)	The namespace which segregates users from remote sites. A user who comes for the records remote site will have this value in the 'user_namespace' column.	No

Of particular note in the remote site table is the 'guid' field as is it shows the identification strategy of sites in the federation. The requirement for a sites credential is it must be unique within the federation. Using a serial (incremented integer) credential would be a poor choice as it supposes there is a federation wide registry that manages serial allocations to sites. Without a federation registry site, generated credential numbers would be likely to collide. Another option for an identify credential would be a domain name. Domain names would be unique, guaranteed by the assigning domain registrar. The problem with domain names is it would limit the ability to change the address of the site as the credential is mapped to its address. The chosen credential to identify a site in the federation is a generated globally unique identifier (GUID). A GUID, is 32 character hexadecimal string generated with an algorithm the guarantees the GUID is unique as the name states. This does not have the problems identified with a serial number or domain name and so is an adequate choice for site identification.

The `remote_permisison` table stores a mapping between a resource permission and a remote site. The fields in this table have the meanings shown in the table below:

Table 11 remote_permission table

Field	Data type	Definition	Optional
id	bigint(20)	The records primary key.	No
active	bit(1)	Whether the permission can be used.	No
guid	varchar(32)	The credential that is supplied to uniquely identify the permission.	No
resource_permission_id	bigint(20)	The foreign key to the mapped resource permission.	No

remote_site_id	bigint(20)	The foreign key to the mapped remote site.	No

The identification strategy of remote permissions is the same as the remote sites though the unique requirement only matters between consumer and provider which have the mapped permission.

The fields added to the existing tables are:

- `is_remote` on `resource_permission` – This specifies the permission is a provider permission and it may be remotely accessed by consumers.
- `site_id` on `rig` – This specifies which site the rig is from. If the rig is locally hosted this field is null.
- `site_id` on `rig_type` – This specifies which site the rig type is from. If the rig is locally hosted this field is null.

As the Scheduling Server uses the Hibernate library for database access and this library requires configuration of the database structure, two new classes were added to the package '`au.edu.uts.eng.remotelabs.schedserver.dataaccess.entities`' that corresponds to the new database table. Each class was annotated with Java Persistence Architecture (JPA) annotations to specify all the details about the table.

.2.3 *MULTISITE EXTERNAL INTERFACE*

The Multisite external interface contains two SOAP services. The first service is used by a consumer to connect to the provider and is called 'Multisite'. The second service is the call-back service that a provider uses to contact a consumer during session. Whilst each service runs in the same bundle (Multisite bundle) and both use the common types as parameters, they are kept separate as they each have distinct purposes and keeping them separate allows a more cohesive implementation to be developed. Each is defined in a Web Service Definition Language (WSDL) file and is implemented as a separate reflective class. The common parameters are defined in a separate XML schema (XSD) that both WSDL files import. The tables below give the descriptions of each of the service operations and the list of input and output parameters. The input and output parameters only list the parameter type contents and do not follow the same structure as defined in the WSDL. Consult the WSDL files attached as an appendix for the defined parameter structure.

.2.3.1 Multisite operation descriptions

Table 12 checkAvailiabity Multisite operation

Name	checkAvailability
Description	Checks whether the specified permission has rigs that are online and free.
Pre-conditions	• The site requesting the permission is allowed to access the permission.
Post-conditions	None, this operation is informational only.
Input parameters	• site – the site that made the request. • permission – the permission that is requested.
Output parameters	• viable – whether the permission may be used. For a permission to be used at least one of its rigs must be online. • hasFree – whether at least one of the permissions rig is free from use. • isQueueable – whether the permission may be queued. • isBookable – whether the permission may be reserved. • isCodeAssignable – whether the permission is batch. • queuedResource – the resource that is actually resolved by the provider permission. This is either a specific rig, rig type or list of requested rig tags.

	• queueTarget – each of the permission rigs statuses so whether it is online and free from use.

Table 13 addToQueue Multisite operation

Name	addToQueue
Description	Adds the user to the queue for a rig in the specified permission.
Pre-conditions	• The site requesting the permission is allowed to access the permission. • The permission is viable as specified by the checkAvailability operation.
Post-conditions	• The user is added to the provider if do not already exist. • The user is added to queue or assigned to a rig depending on whether a rig is free.
Input parameters	• site – the site that made the request. • permission – the permission that is requested. • user – the user to add to the queue.
Output parameters	• successful – boolean flag to specify whether adding the user to the queue was successful. • inQueue – whether the user is currently in the queue. • inSession – whether the user has been assigned to the rig. • queuedResource – the permissions resource that is actually queued. • session – session details (consult the get session information operation for the list of these details).

Table 14 getUserStatus Multisite operation

Name	getUserStatus
Description	Determines the users status of the user at the provider. A user's status may be one of: • In the queue of a permission's resource. • Waiting for a reservation to start. • In session

	• None of the above.
Pre-conditions	• The user exists as a consumer user at the provider.
Post-conditions	None, this operation is informational only.
Input parameters	• site – the site that made the request. • user – the user that is having their status checked.
Output parameters	• inQueue – whether the user is currently in the queue. • inSession – whether the user has been assigned to the rig. • inBooking – whether the user is waiting for a reservation to start. • queuedResource – the permissions resource that is actually queued. • bookedResource – the resource that is reserved in the starting reservation. • session – session details (consult the get session information operation for the list of these details).

Table 15 getQueuePosition Multisite operation

Name	getQueuePosition
Description	Returns the position of the user if they are in queue.
Pre-conditions	• The user exists as a consumer user at the provider. • The user is in the queue.
Post-conditions	None, this operation is informational only.
Input parameters	• site – the site that made the request. • user – the user that is having their status checked.
Output parameters	• inQueue – whether the user is currently in the queue. • inSession – whether the user has been assigned to the rig. • inBooking – whether the user is waiting for a reservation to start. • queuedResource – the permissions resource that is actually queued. • position – the position of the user in the queue. This is an increased integer with 1 being first in the queue, 2 being

	second in the queue and so on. • time – the amount of time the user has been in the queue in seconds.

Table 16 getSessionInformation Multisite operation

Name	getSessionInformation
Description	Returns details about a users session. Including how long they have been in session, the remaining session time, session parameters and the in use rig details.
Pre-conditions	• The user exists as a consumer user at the provider. • The user is in session.
Post-conditions	None, this operation is informational only.
Input parameters	• site – the site that made the request. • user – the user that is having their status checked.
Output parameters	• isReady – whether the rig is ready to be used. Once a rig is ready to user may launch any rig control interfaces and actually use the rig. • rigType – the name of the assigned rigs type. • rigName – the name of the assigned rig. • contactURL – the address to access the rigs rig client. • duration – the time in seconds the user has been in session. • extensions – the number of extensions the user has remaining. • timeLeft – the time in seconds the user left before they either have their session time extended or are removed from the rig. • inGrace – whether the session is about the expire and the user has been marked for removal. • warningMessage – a textual warning message for display on the consumer WI.

Table 17 finishSession Multisite operation

Name	finishSession
Description	Finishes a users in progress queue or rig session. If the user is in the queue, they are removed from it. If the user is using a rig, the rig is released from user by another user.
Pre-conditions	• The user exists as a consumer user at the provider. • The user is in the queue or is in session.
Post-conditions	• The user is removed from the queue or session.
Input parameters	• site – the site that made the request. • user – the user that is having their status checked.
Output parameters	• successful – boolean flag to specify whether finish the users session was successful

Table 18 createBooking Multisite operation

Name	createBooking
Description	Creates a reservation for user a rig in the specified permission's resource.
Pre-conditions	• The site requesting the permission is allowed to access the permission. • The permission allows reservation at the user selected time. • A rig is free at the user selected time.
Post-conditions	• The user is added to the provider if do not already exist. • A reservation is created for redemption at the user specified time.
Input parameters	• site – the site that made the request. • permission – the permission that is requested. • user – the user to add to the queue. • start – the time at which the reservation is scheduled to start. • end – the time at which the reservation is scheduling to be either extended or finished.
Output parameters	• successful – boolean flag to specify whether creating the reservation was successful.

	• bookingID – identifier for the created reservation. • bestFits – list of alternative times the user may make the reservation if they requested time failed to be reserved.

Table 19 cancelBooking Multisite operation

Name	cancelBooking
Description	Cancels a users reservation freeing the held rig for use by another user. This does not notify the user of the cancellation. It is expected to the consumer system notifies the user.
Pre-conditions	• The specified reservation exists and belongs to a user from the requesting site.
Post-conditions	• The reservation is cancelled.
Input parameters	• site - the site that made the request. • bookingID – the identifier of the reservation to cancel.
Output parameters	• successful – boolean flag to specify whether creating the reservation was successful.

Table 20 findFreeBookings Multisite operation

Name	findFreeBookings
Description	Finds the times with which a permission may be reserved in a specified time window.
Pre-conditions	• The site requesting the permission is allowed to access the permission.
Post-conditions	None, this operation is informational only.
Input parameters	• site – the site that made the request. • permission – the permission that is requested. • start – the time at which to start providing free times. • end – the time at which the to stop providing free times.
Output parameters	• slot – a list of time periods where the permission is free to be reserved.

.2.3.2 Multisite Callback operation descriptions

Table 21 bookingCancelled Multisite Callback operation

Name	bookingCancelled
Description	Notification that the provider has cancelled a reservation.
Pre-conditions	• The specified reservation exists.
Post-conditions	• The consumer cancels the synchronized reservation and notifies the user their reservation has been cancelled.
Input parameters	• site – provider site that had the reservation. • bookingID – the identifier of the reservation that has been cancelled.
Output parameters	• successful – boolean flag to specify the consumer acknowledges the notification.

Table 22 sessionStarted Multisite Callback operation

Name	sessionStarted
Description	Notifies a consumer site a users session has started at the provider. A session may be started if a users reservation is redeemed or the user has exited the queue and been assigned to a rig.
Pre-conditions	• The user exists as a consumer user. • The user is in the queue or has a reservation starting.
Post-conditions	• The consumer creates or updates a synchronized session mapped to a provider session.
Input parameters	• site – provider site that the session has started at. • user – the user whose session has started. • permission – the permission the session was started from. • isReady – whether the rig is ready to be used. Once a rig is ready to user may launch any rig control interfaces and actually use the rig. • rigType – the name of the assigned rigs type. • rigName – the name of the assigned rig. • contactURL – the address to access the rigs rig client. • duration – the time in seconds the user has been in session.

	• extensions – the number of extensions the user has remaining. • timeLeft – the time in seconds the user left before they either have their session time extended or are removed from the rig. • inGrace – whether the session is about the expire and the user has been marked for removal. • warningMessage – a textual warning message for display on the consumer WI.
Output parameters	• successful – boolean flag to specify the consumer acknowledges the notification.

Table 23 sessionFinished Multisite Callback operation

Name	sessionFinished
Description	Notifies a consumer site a users session has finished and they are being released from the rigss Rig Client.
Pre-conditions	• The user exists as a consumer user. • The user is in session.
Post-conditions	• The consumers finishes the synchronized session.
Input parameters	• site – provider site that the user had a session on. • user – the user that has had their session finished.
Output parameters	• successful – boolean flag to specify the consumer acknowledges the notification.

Table 24 sessionUpdate

Name	sessionUpdate
Description	Notifies a consumer site a user's session has had a parameter change. This may be that a time extension has been given, the rig is ready for use, the grace period has started or a warning message should be displayed.
Pre-conditions	• The user exists as a consumer user. • The user is in session.
Post-conditions	• The consumer updates a synchronized session mapped

	to a provider session.
Input parameters	• site – provider site that the user has a session on. • user – the user that has had their session finished. • permission – the permission the session was started from. • isReady – whether the rig is ready to be used. Once a rig is ready, the user may launch any rig control interfaces and actually use the rig. • rigType – the name of the assigned rigs type. • rigName – the name of the assigned rig. • contactURL – the address to access the rigs rig client. • duration – the time in seconds the user has been in session. • extensions – the number of extensions the user has remaining. • timeLeft – the time in seconds the user left before they either have their session time extended or are removed from the rig. • inGrace – whether the session is about to expire and the user has been marked for removal. • warningMessage – a textual warning message for display on the consumer WI.
Output parameters	• successful – boolean flag to specify the consumer acknowledges the notification.

.2.4 MULTISITE PROVIDER MODULE

The diagram shown in the 'Application Architecture Modifications' shows the Multisite Provider module sites at the resource layer of the Scheduling Server. It is used by session initiation (queuing and reservations) and session management (session) layers and it is implemented first so when they are subsequently modified, they have their required dependencies. The Multisite Provider module implements a client to the Multisite SOAP service. When a higher level module needs to interact with a provider, this client is used. The class design is shown in the diagram below:

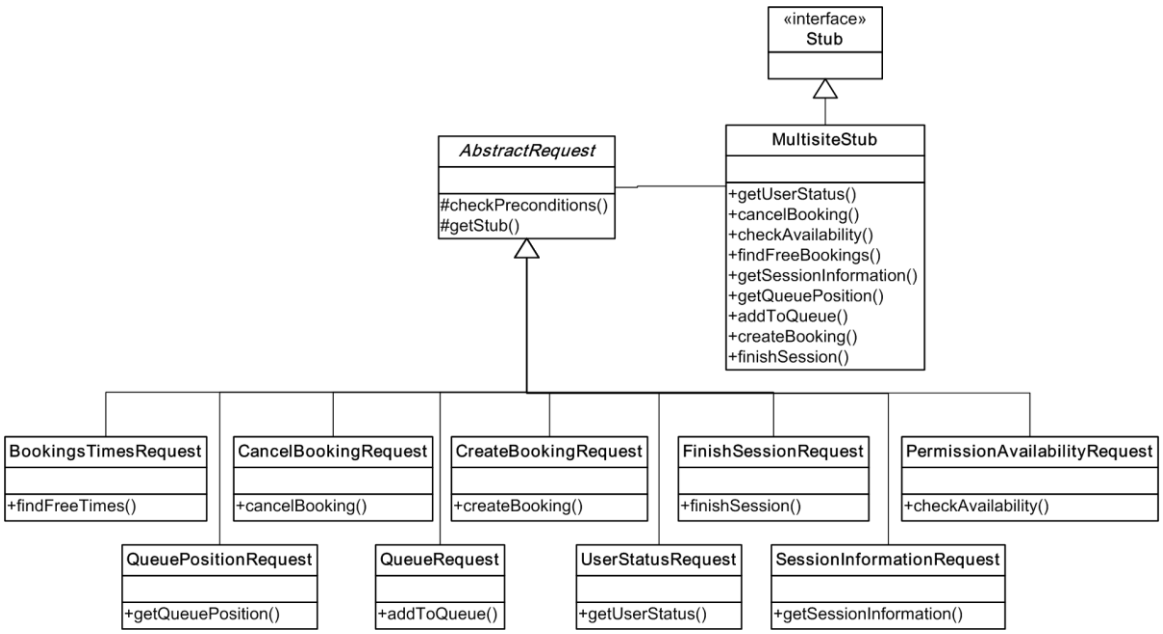


Figure 34 Multisite Provider class diagram

The class diagram shows a base class '`AbstractRequest`' that is extended with derived classes for each request. Each derived class has a method used to initiate synchronous communication with the provider using the class '`MultisiteStub`' class which implements synchronous SOAP communication using the Apache Axis2 library.

To understand the utility of wrapping requests in request classes, an understanding of the OSGi framework modularity layer is needed. The modularity layer allows a module to declare code that is part of the modules public interface and that which is part of the modules private implementation at the granularity of Java packages. In the case of this module, the implementation of service using Apache Axis is a private implementation detail. The consumption of requests is publicly available using request classes with methods that take database entities (record objects) as parameters. Database entities best encapsulate the concepts that are being communicated. For example, a permission is specifically a '`resource_permission`' record and in Scheduling Server terms a

‘ResourcePermission’ entity object so these provide convenient interface parameters.

Decoupling the use and implementation of the Multisite service and its implementation allows the Multisite service to be modified independently of the components that require it. It may be that multiple versions of the Multisite service will be created as the federation system is further developed and this can be seamlessly supported without tainting other modules with its details. For example, the ‘getStub’ method in the request object base could be modified to detect the protocol version using information contained in the ‘RemoteSite’ entity provided to it and return a protocol specific Apache Axis2 stub. This shows the design rationale behind the Multisite Provider module.

.2.5 *QUEUING MODULE*

The Queuing module implements the queuing functionality of the Scheduling Server. It provides an external SOAP service with the operations:

- `addUserToQueue` – This adds a user to the queue.
- `removeUserFromQueue` – This removes a user from the queue.
- `gerUserQueuePosition` – This returns the position of the user in the queue.
- `checkResourceAvailiability` – This returns the status of a permissions rigs and thus determines whether the permission can be used.
- `isUserInQueue` – This determines the status of the user including whether the user is in the queue, in session or waiting for a reservation to start.

These operations are used by the Web Interface to interact with the queue. These operations also have equivalents in the Multisite service as it must also interact with the queue. The modifications required to be made to the Queuing module operations are two-fold. These are:

- For a consumer system, the implementation of the above operations should dispatch to the provider to actually perform the desired functionality if the requested permission is of type 'CONSUMER'. For example, the '`addUserToQueue`' operation should create an instance of '`QueueRequest`' from the Multisite Provider and invoke the `addToQueue` operation to ask the provider to queue the user.
- For a provider system, the Multisite service implemented in the Multisite module will receive '`addToQueue`' request to queue a user. It will need to '`addUserToQueue`' operation after suitable validation to actually add the user to the providers queue. To do this requires an internal service to be exported by the Queuer module for consumption by the Multisite module.

The strategy to implement both these modifications is to develop an internal service that wraps the existing SOAP service implementation and exports it as an OSGi service. An OSGi service is just a normal object registered against some class signature (i.e. its interface) with the OSGi framework service registry. The existing Queuer SOAP operation implementations are extracted into a separate object instead of directly implementing the operation's functionality. The extracted object is then called by the SOAP operation and registered as an OSGI service. As with the Multisite Provider public interface, database entities will be used as parameters instead as they best encapsulate the data that must be passed around. This provides a much more straight forward

interface then directly exporting the SOAP service implementation object. The diagram below visualises this change:

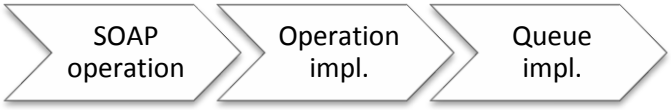


Figure 35 Existing Queuer operations

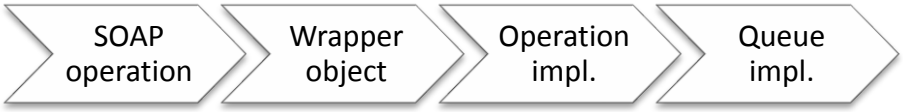


Figure 36 Queuer operations wrapped

Once this is implemented the operation implementations are modified to switch on the permission type enumeration. If the permission type is 'CONSUMER', the Multisite Provider will be called instead of the internal queue.

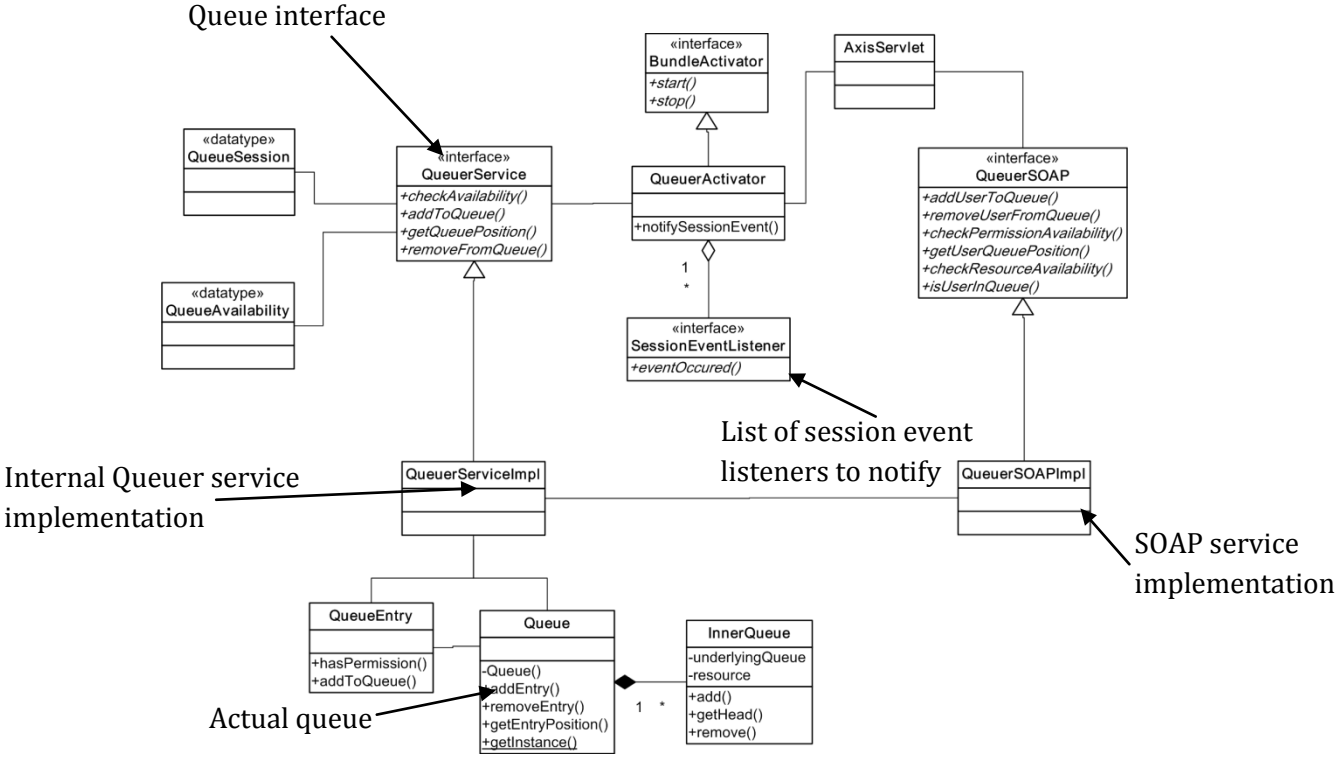


Figure 37 Queuer class diagram

The class diagram above shows this implementation with the '`QueuerService`' interface declaring the object wrapper for the Queuer service the

'QueuerServiceImpl' implementing the service. The 'QueuerSOAPImpl' which implements the Queuer SOAP service uses the 'QueuerServiceImpl' to implement its functionality. Formally, the 'QueuerService' takes database entities as parameters (such as a 'User' object to specify a user) so request validation and loading of these entities is left to the caller of the object wrapper. For example, in the case of the Queuer SOAP service 'addUserToQueue' and the Multisite 'addToQueue' operations, whilst the Queuer operation validates the user exists, the Multisite operation validates the calling site and whether they have the permission they are requesting, adds the user if they do not already exist and assigns the permission to the user. The point of convergence between these two operations is verifying the user has permission and adding them to the permission's queue and this is exactly what the object wrapper provides.

The class diagrams also shows the interfaces 'BundleActivator', 'SessionEventListener' and the class 'QueuerActivator'. A 'BundleActivator' (also called 'activator') is the life-cycle entry point of a module in the OSGi framework. The activator for the Queuer module ('QueueActivator') does three things:

- Registers the Queuer internal Scheduling Server interface 'QueuerService' with the OSGi framework service registry.
- Registers an 'AxisServlet' instance with the OSGi framework service registry which allows it to be picked up by the Server module and hosted as a servlet on the Server module servlet engine.
- Sets up an OSGi framework service registry listener to be track 'SessionEventListener' service registrations. This allows each of these to be obtained for invocation when the Queuer session must notify other modules of session events.

The session event notifications that the Queuer module provides are:

- QUEUED – The user has been added to the queue.
- ASSIGNED – The user has been assigned to a rig.

.2.6 *BOOKINGS MODULE*

The same modifications made to the Queuer module described in the previous section are required to be made to the Bookings module so it calls a provider in its consumer operation and may be called as a provider internally. The layout of the Bookings

module is superficially similar to the Queuing module and the exported internal service is:

Figure 38 Bookings module internal service

The bookings module assigns users to rigs when a booking has been redeemed so it provides session event notifications of:

- `ASSIGNED` – A users booking has been redeemed.

.2.7 *Session Module*

Again, similar to the Queuer and Bookings modules, the Session module is required to be modified so it calls a provider in its consumer operation and may be called as a provider internally. The layout of the Bookings module is superficially similar to the Queuing module and the exported internal service is:

Figure 39 Session module internal interface

The Session module manages in progress sessions so it provides session event notifications of:

- `EXTENDED` – A session has been extended.
- `GRACE` – A session has been marked for termination when its remaining time has elapsed.
- `FINISHED` – A session has been terminated.

.2.8 MULTISITE MODULE

This section details the implementation of the Multisite module and is the terminal destination of federation use implementation. It depends on other modules to provide functionality for it but no module depends on it and so sits in the top layer of the Scheduling Server. The Multisite module must:

- Provide the server side of the Multisite SOAP service, so when requested by a consumer, perform the requested functionality in collaboration with other Scheduling Server modules.
- Provide the service side of Multisite Callback SOAP service, so when requested by a provider, update the state of synchronized sessions and reservations.
- Listen for session events and on receipt of a session event of a federation use session, use a client of the Multisite Callback SOAP service to notify the consumer.

The diagram below shows the class structure of the Multisite module.

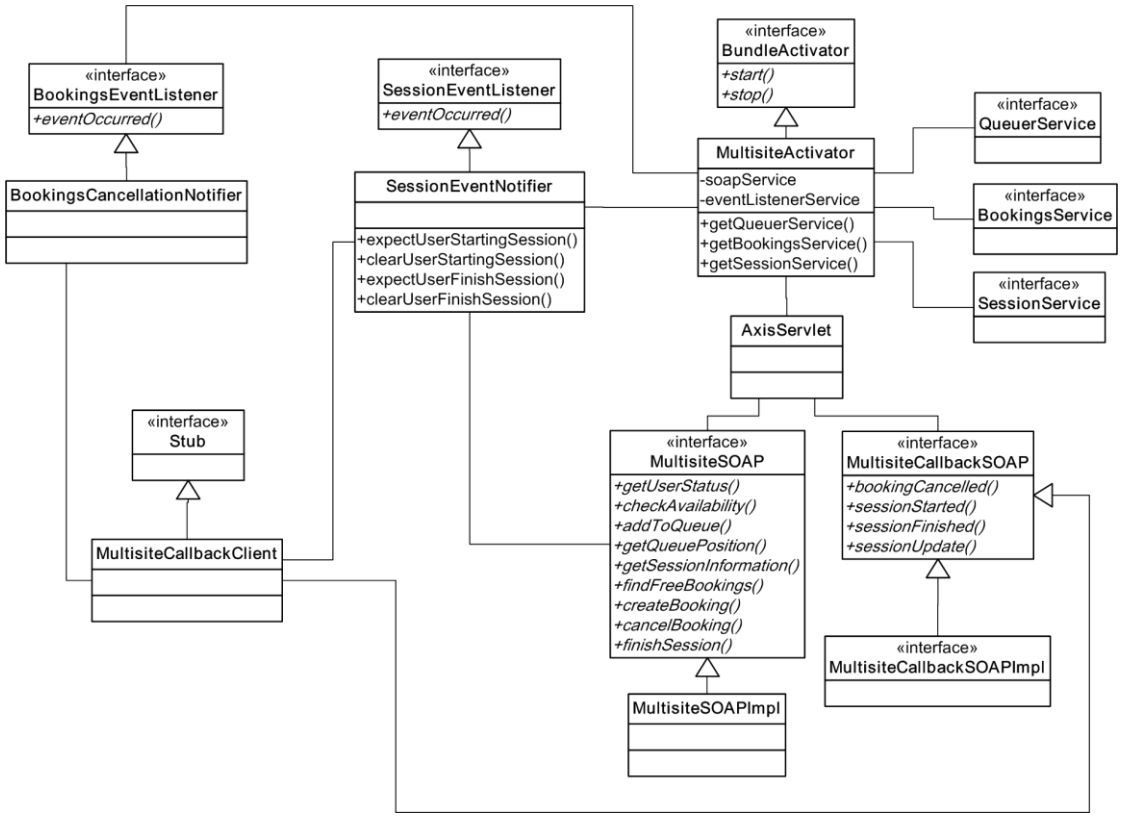


Figure 40 Multisite class structure

The class diagram shows how the module was implemented. The module activator (‘`MultisiteActivator`’) is used to start and stop the module. During start, the required services are obtained from the OSGi service registry and the Multisite SOAP

are registered and session event listeners services are registered so they are hosted on a server for external requests and actively receiving session event notifications respectively.

The '`MultisiteSOAPImpl`' is the implementation of the Multisite SOAP service and as described in previous sections uses the internal Queuer, Bookings and Session services.

The '`SessionEventNotifier`' class is an implementation of a session event listener that on receipt of a session event notifies the consumer Scheduling Server of that event provided the following conditions are true:

- The session is from federation use. If the session is from local use, there is no need for notification to be sent out.
- The session has not been masked out for notifications. A session may be masked out to prevent duplicate notifications being sent out if an initial requests response gives the same information as the notification would. For example, when a user is added to the queue they may be directly assigned to a rig and the queue request will respond the user has been assigned to a rig. In this case, sending a session started notification is redundant as the consumer will already know of the assignment because of the queue request response.

With the possible session event types, the following Multisite Callback notifications are called:

Table 25 Multisite Callback operations called from session events

Event	Callback Operation	Notes
QUEUED	N/A	The consumer already knows the user has been queued because they requested it.
ASSIGNED	`sessionStarted`	This is sent when a session starts from either waiting in the queue or reservation redemption.
READY	`sessionUpdate`	The consumer must update the user.
EXTENDED	`sessionUpdate`	The consumer must update the user.
GRACE	`sessionUpdate`	The consumer must update the user.
FINISHED	`sessionFinished`	This is sent when a session is finished because its time has elapsed and cannot be extended.

Similarly to the way session event notifications are sent to the consumer, reservation cancellation notifications are provided by registering an implementation of the 'BookingsEventListener' interface as an OSGi service. This provides notifications when a reservation is cancelled and the nature of cancellation which is either the reservation was cancelled because of a user request or cancelled by the system. Only system cancellations are used to send a cancellation notification to the consumer.

.3 Web Interface Modifications

The Web Interface requires much less modifications than the Scheduling Server. The modifications have two goals. The first is differentiating federation rigs from local rigs and is entirely optional. The second is supporting rig specific session pages and is required to productively use a rig.

.3.1 *User Interface Changes*

Typically a student user would not be concerned where a rig is hosted, either at their university or at another. The students institution may however be concerned. If the user interface of a remote laboratory broadcasts or markets another university this *may* prove a disincentive for use of that remote laboratory. This suggests the user interface should not present federation rigs markedly different to local rigs. The changes made in this regard were to slightly differentiate the permission selection button and the loading dialogue whist the permission status is being retrieved. The following screenshots compare a local rigs selection interface with a federation rigs selection interface:

Local rig **Federation rig**

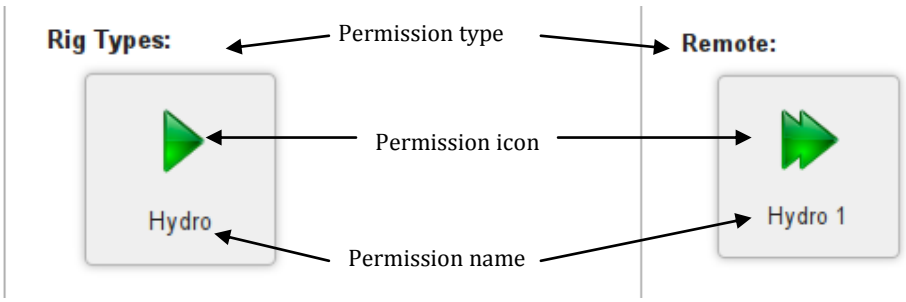


Figure 41 Local rig permission button

Figure 43 Federation rig permission button

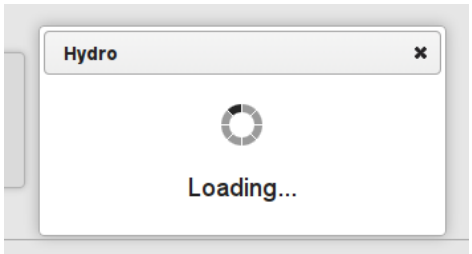


Figure 42 Local rig loading dialogue

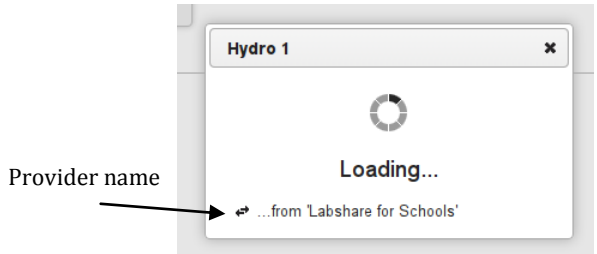


Figure 44 Federation loading rig dialogue

The screenshots show slight differences with the permission categorisation of a federation permission under a 'Remote' grouping. The iconography is also slightly different with a green double arrow instead of a single arrow as in the local permission

case. The federation permission loading dialogue also shows the provider name that is being contacted to load the permissions availability.

The case when strong differentiation should be made is when loading a permission fails. This is the first point of contact and drives what the user may perform with the permission. An error caused by not being able to communicate with the provider is fatal in terms of a user use of the permission. If the provider could not be contacted, then the user will not be able to queue or reserve a rig. The loading dialogue displays an error message no 'Queue' or 'Reserve' buttons so they may not request either of these actions.

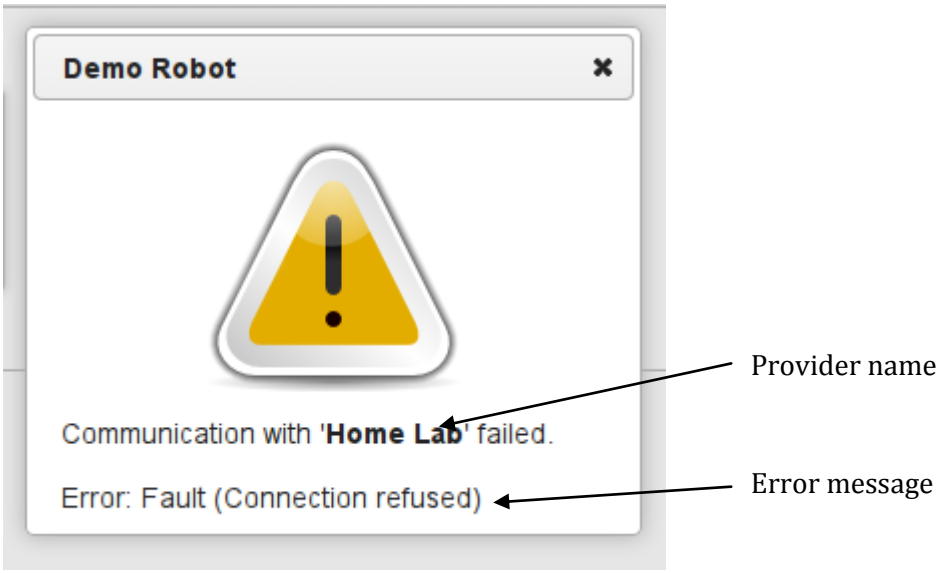


Figure 45 Provider communication error

4.3.2 *RIG SPECIFIC SESSION INTERFACE*

As described in the Web Interface section of the previous section, rigs have specific session page interfaces. For a rig to be productively used in the federation these interfaces need to be preserved at the point of use. Two possible options were redirecting the user to the provider Web Interface or melding the rig specific session page into the consumer Web Interface session page.

Redirecting the user to the provider Web Interface has the advantage of the rig specific session page is already at the provider. This does not need to be transferred to the consumer and if the rig specific page is upgraded it does not require all the consumer sites that use it to upgrade their version. The disadvantage is the user is no longer present on the consumer Web Interface which has implications of branding and style. The provider site appears different and has different branding to the users consumer site. This includes the header icons and banner image and the footer text. If the consumer site has a customised style with different colours, fonts or iconography, these are not preserved on the provider Web Interface. This makes the use of a federation rig or a local rig quite noticeable to the user.

Another significant disadvantage is the SAHARA Labs Web Interface has a 'Contact Support' button to allow the user to alert the technical support of the site of errors and provide feedback. If the user is redirected to the provider site, using this button will no longer go to the consumer which is the most appropriate place as the user is a member of the consumer not the provider.

Using consumer Web Interface melding of the rig specific session pages alleviates the problems identified with redirecting the user to the provider. It also makes the distinction between a federation rig and local rig only as noticeable as explicitly implemented on the Web Interface. There are no imposed distinctions as redirection would force. The disadvantage is a consumer would need to have the files of the rig specific session page so they can be combined with the common session page portions during use. This does not seem to be an onerous task as federation use of a rig already involves setup of permissions for use. This seems to be an opportune time to conduct transfers of rig specific session page generating files for consumer use.

One could argue there may be intellectual property considerations if files are being transferred between providers and consumers but this argument does not hold credence with the intended use of the files. The files are for a Web Interface that is consumable with an implied agreement between the consuming and providing parties

(the permission) so just the same as the file can be transferred, they can be downloaded from the provider's web server. For example, much dynamic behaviour with web pages is implemented using JavaScript for execution by a web browser. For a web browser to do this it must download it from a web server. Transferring JavaScript files between web servers (i.e. Web Interfaces) or downloading the file directly from the providers web server provides equivalent access to the intellectual property and is imposed by the way web browsers work.

Melding a rig specific session page client side using content partially provided by the consumer Web Interface (common portions) and partially from the provider Web Interface (rig specific) would impose constraints on what areas of the page could be modified because of the same origin policy of web browsers. For example the following screenshot shows an existing rig specific page hosted at RMIT University and shows the rig specific portion modifies the common portion. The buttons usually located on the right of the page have been moved across to the left of the page under the session times. This would not be possible if the rig specific portion was deployed client side even if the deployment mechanism was HTML frames.

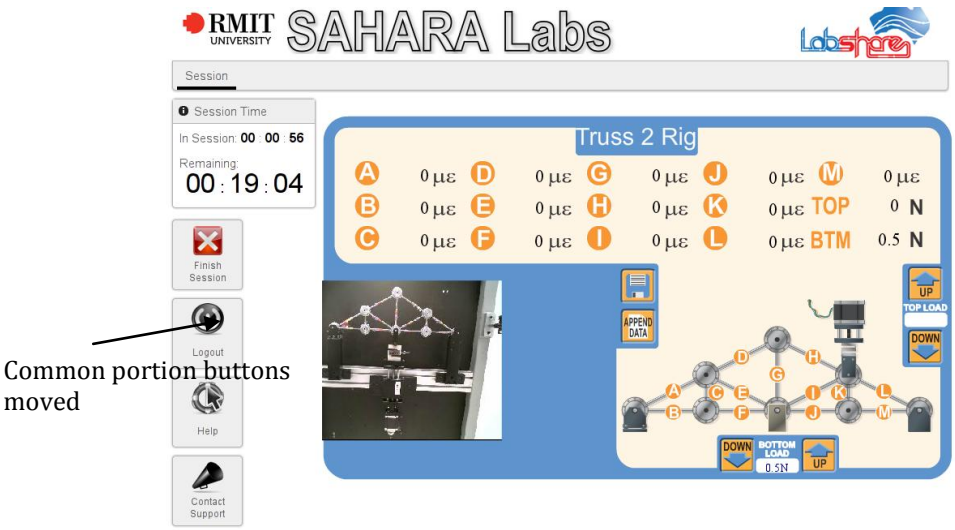


Figure 46 Rig specific session page with modified common portion

The most ideal solution and the one that is implemented is consumer Web Interface melding of the rig specific portion into the consumer Web Interface session page. The implementation of this was straight forward. If the session is using a provider rig, the rig specific session page partial script called '`<rigtype>.phtml`' is loaded from '`/institution/<provider namespace>/scripts/`' instead of the usual'`/institution/<consumer namespace>/scripts/`'. Provided the script file exists in that location and the page required dependencies (such as applets, scripts,

images, style sheets) are present on the consumer web server the page is delivered from the consumer Web Interface the same as it would if it was being delivered from the provider Web Interface.

Using consumer Web Interface melding of rig specific and common portions requires caution by rig developers where customisations are added to support a rig. They should not be made to base SAHARA Labs files but specific rig files. For example, if changes were made to the page layout style sheet, this could not be used by a consumer as it would change the layout of their Web Interface which itself might be customised. The screenshots below show an error that has occurred because the rig specific portion expects the common style sheet to have customised styling for it:

Figure 47 IPR local access

Figure 48 IPR federation access

.4 VERIFICATION

This section verifies the implementation of federation use of rigs using scenario testing and some limited performance testing. The test setup is shown in the diagram below where nodes represent sites in the federation and lollipops represent rigs.

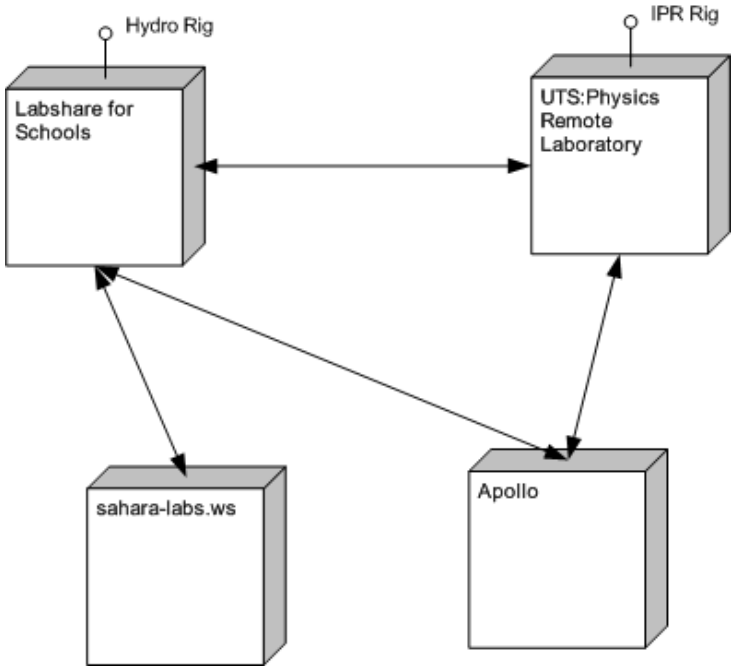


Figure 49 Test setup

The sites shown in the diagram are:

- **Labshare for Schools (LHS)** – This site is hosted within the UTS Remote Laboratory. It is the provider of the Hydro rig and a consumer of the Inclined Plane rig (IPR). This runs on CentOS 5.6 and may be accessed with the address http://sahara3.eng.uts.edu.au.
- **UTS:Physics Remote Laboratory (PHYS)** – This site is hosted within the UTS Remote Laboratory. It is the provider of the IPR rig and a consumer of the Hydro rig. This runs Windows XP and may be accessed with the address http://sahara1.eng.uts.edu.au.
- **Apollo (AP) –** this site is hosted within the UTS Remote Laboratory test infrastructure. It is a consumer of the Hydro and IPR rigs. This runs CentOS 5.6 and may be accessed with the address http://apollo.eng.uts.edu.au:7070. This site is notably authenticated using single-sign-on (SSO) through the Australian Access Federation (AAF) and shows there is no contradiction between SSO and remote laboratory federation.

- **sahara-labs.ws (SLWS) –** This site is hosted on the Amazon EC2 cloud, United States East Coast availability zone as 'micro' instance which provides the lowest possible processing speed and bandwidth. This runs Redhat Entreprise Linux 6.1 and may be accessed with the address http://www.sahara-labs.ws.

.4.1 *Scenario Testing*

Scenario testing utilises a list of steps with expected outcomes to determine whether a function of the system has been successfully achieved. The scenarios that are being tested are:

- Using a rig with direct assignment
- Using a rig with queuing assignment
- Using a rig with reservation assignment

Each of these scenarios is executed against using the following matrix showing the consumer / provider relationships.

Table 26 Test matrix

	LHS	PHYS	AP	SLWS
Hydro on LHS	N/A	X	X	X
IPR on PHYS	X	N/A	X	

Note: The IPR is not being tested with the SLWS system. This is a firewall issue of not having enough ports open to allow the Web Interface, Scheduling Server, Rig Client and remote desktop to be externally accessible. As SLWS is external, it would not be able to communicate with all the PHYS infrastructure.

Between each of the consumer and providers a permission was setup which was for the rig type of the rig that was hosted at provider. The permission had the following constraints:

- Allows queuing and reservations.
- Session duration of 900 seconds.
- Three time extensions with a duration of 900 seconds each.
- A maximum of three concurrent extensions.
- Session idle timeout is enabled with a timeout period of 600 seconds.

Table 27 Test scenario of using a rig with direct assignment

Name	Using a rig with direct assignment
Description	This test scenario verifies using a consumer site to access a rig at a provider site. The rig is used to the maximum session allowed time including time extensions after which the user is removed from the session and directed back to the 'Rig Selection' page. When accessing the rig, it is free at the provider so the consumer's user is directly assigned to the rig.
Steps to execute	1. Login to the consumer site. 2. Click the permission button for the providers rig. 3. Click the 'Queue' button. 4. Verify the rig specific session page is shown with an overlay saying 'Please wait...'. 5. Wait 10 seconds and verify the overlay is cleared and the rig specific session page is visible with a warning to specify the session will timeout in 600 seconds. 6. Launch the rig. 7. Wait 900 seconds and verify a time extension is given. 8. Wait a further 2400 seconds and verify a warning is displayed saying the session will be end in 300 seconds. 9. Wait a further 300 seconds and verify the browser is redirected to the 'Rig Selection' page.
Expected outcomes	From 1. The user is logged in and sees the provider permission on the Rig Selection page. From 2. A loading dialogue opens and after a short period the status of the rig is shown as free. Buttons to queue and reserve the rig are shown. From 3. The dialogue from 2. changes to 'Requesting...' then the browser is redirected to the session page. From 6. If the Hydro rig is being tested, the camera and slider controls should be visible and when changed an indicator should change value. If the IPR is being tested, the Java RDP applet should be visible and when clicked launch a window with the rig control application.

	From 7. The in session time plus the remaining time should equal 1800 seconds.
PHYS using Hydro	PASS
AP using Hydro	PASS
SLWS using Hydro	PASS
LHS using IPR	PASS
AP using IPR	PASS

Table 28 Test scenario of using a rig with queuing

Name	**Using a rig with queuing assignment**
Description	This test scenario verifies using a consumer site to access a rig at a provider site after entering the providers queue. The rig is already in use by a provider's user and when the consumer's user enters the queue they are first in the queue. The provider' user finishes their session and the consumer's user is assigned to the rig. The rig is used briefly then the session is finished by the consumer's user.
Steps to execute	1. Login to the provider site and select to use the local rig. 2. Login to the consumer site. 3. Click the permission button for the providers rig. 4. Click the 'Queue' button. 5. Verify the queue position is 1st. 6. Finish the provider's user session. 7. Wait up to 30 seconds for the consumer's browser to be redirected to the 'Session page'. The page should have an overlay visible. 8. Verify the rig specific session page is shown with an overlay saying 'Please wait...'. 9. Wait 10 seconds and verify the overlay is cleared and the rig specific session page is visible with a warning to specify the session will timeout in 600 seconds. 10. Launch the rig.

	11. Click the 'Finish Session' button and finish the session.
Expected outcome	From 1. The provider's user is assigned to the rig. From 2. The consumer's user is logged in and sees the provider permission on the Rig Selection page. From 2. A loading dialogue opens and after a short period the status of the rig is shown as in use. Buttons to queue and reserve the rig are shown. From 3. The dialogue from 3. changes to 'Requesting...' then the browser is redirected to the Queuing page. From 6. The provider's user is redirected to the 'Rig Selection' page. From 10. If the Hydro rig is being tested, the camera and slider controls should be visible and when changed an indicator should change value. If the IPR is being tested, the Java RDP applet should be visible and when clicked launch a window with the rig control application. From 11. The consumer's user's browser is redirected to the 'Rig Selection' page.
PHYS using Hydro	PASS
AP using Hydro	PASS
SLWS using Hydro	PASS
LHS using IPR	PASS
AP using IPR	PASS

Table 29 Test scenario of using a rig with reservation

Name	**Using a rig with reservation**
Description	This test scenario verifies using a consumer site to access a rig at a provider site after reserving the provider's rig. When the reservation is scheduling to start, the user is assigned to the rig. The rig is not used and the session is terminated by the provider because of the session idle timeout expiring.
Steps to execute	1. Login to the consumer site. 2. Click the permission button for the providers rig. 3. Click the 'Reserve' button.

	4. Choose a timeslot and create a reservation for 3600 seconds. 5. Login to the provider's Web Interface, go to the rigs reservation page and view the timeslots the reservation was created in. 6. Login to consumer site 1 minute before the scheduling reservation time. 7. Wait for 30 seconds after the countdown timer reaches 0. The consumer's browser should be redirected to the 'Session page'. The page should have an overlay visible. 8. Verify the rig specific session page is shown with an overlay saying 'Please wait...'. 9. Wait 10 seconds and verify the overlay is cleared and the rig specific session page is visible with a warning to specify the session will timeout in 600 seconds. 10. Close the browser. 11. Open a browser and log back into the consumer site.
Expected outcomes	From 1. The consumer user is logged in and sees the provider permission on the Rig Selection page. From 2. A loading dialogue opens and after a short period the status of the rig is shown as in use. Buttons to queue and reserve the rig are shown. From 3. The user is redirected to the reservation page for the provider's rig. The page lists the rigs free and reserved times. From 4. The reservation should be created. An email should be sent to users stored email address and be from the consumer system. From 5. The timeslots should be marked as not available. From 6. The user should be redirected to the 'Reservation Waiting' page with the timer counting down to reservation start time. From 7. The user is redirected to the session page. From 11. The user should be on the 'Rig Selection' page.
PHYS using Hydro	PASS
AP using Hydro	PASS

SLWS using Hydro	FAIL at step 4. The time slots are shown with incorrect statuses.
LHS using IPR	PASS
AP using IPR	PASS

4.4.2 Verification Outcome

Scenario testing of the federation use implementation showed the system predominately works. The things verified to work were:

- Viewing the status of a provider's rig.
- Allowing the user to be assigned to a provider's rig when it is free.
- Having the ready overlay cleared when the rig was free to use.
- Having the session idle timeout warning correctly display the time remaining before the session is terminated.
- Having a rig specific session page deployed to present the rig specific controls.
- Having the customer Web Interface request rig parameters from the Rig Client to deploy rig controls and the camera feeds.
- Allowing the user interface with the rig using the consumer Web Interface communicating with the provider Rig Client to control a rig (Hydro rig control).
- Allowing the user to launch a remote desktop application to control the rig (IPR rig control).
- Having time extensions being given to the session if the rig is free and having the consumer Web Interface display the time extension notification and update the remaining time left.
- Having the session expiry warning being displayed with 300 seconds of the session expiry.
- Having the provider terminate a consumer user's session at the session expiry time and notifying the consumer so the user is redirected off the 'Session' page.
- Allowing the user to be added to a rig's queue at the provider if the rig is in use at the request time.
- Having the consumer web interface correctly display the queue position at the provider.
- Having the user being assigned to a provider's rig when the rig is next free.
- Allowing the user to finish session when they request it.

- Allowing the user to view the times at which a provider rig can be used (with an exception explained below).
- Allowing the user to create a reservation for a provider's rig.
- Having the user being put in the 'Reservation Waiting' page when the user has a reservation scheduled to start with the correct time remaining displayed until the reservation starts.
- Having the user assigned to the provider's rig at the scheduled start time.
- Having the user removed from the provider's rig when the session idle timeout elapses.

The failure of viewing the times a provider's rig can be reserved with the sahara-labs.ws consumer and Labshare for Schools provider was caused because both systems are hosted in a different time zone. The sahara-labs.ws system is hosted in the Amazon EC2 east availability zone which is in North Virginia, United States east and has the time zone 'Eastern Standard Time' at -5 hours from GMT. The Labshare for Schools system is hosted at the UTS Remote Labs in Sydney, Australia and has the time zone 'Eastern Daylight Time' at +11 hours from GMT. It appears communication between the two time zones has triggered a bug in the underlying SAHARA Labs reservation system causing incorrect times to be displayed.

5. Federation Management Architecture

The federation use architecture and implementation described in the previous two chapters provides the preconditions that must be setup so federation use may be performed. These are:

1. Discovery of other SAHARA Labs federation sites and determining the credentials to uniquely identify them and the address to communicate with them.
2. Identification of the rigs each SAHARA Labs federation site has and the times they are willing to allow these rigs to be used.
3. Negotiating of permissions so that if there is agreement on sharing both consumer and provider systems, both have the permission identified by a known credential.
4. Transfers of customised rig session pages from providers to consumers so the consumer Web Interface can correctly deploy the rig specific session page during use of the rig.

With the above four preconditions, point 4 can be discounted because transferring of rig session pages would occur infrequently because rig interface modifications do not (currently) have a high churn rate. Also, as their need is predicated on a permission to access a rig, the negotiation process presents an ideal time to perform transfers of these files. Therefore point 4 does not affect the work flow of actions in federation use and so has little bearing on this discussion.

Before discussing the criteria to determine the most appropriate architecture, a basic definition of what the architecture is intended to achieve is needed. This architecture is for a *technical* solution to achieve points 1 to 3 above, not an *operational* solution. A technical solution connects up systems and provides user interfaces to automate accomplishing a goal. An operational solution is a process to follow so a goal can be accomplished. The Labshare Institute (TLIs) already provides this operational solution being an organisation that conducts sharing and negotiation through a *catalogue* of rigs. Both technical and operational solutions may be complementary and the synergy between these will determine the success of the SAHARA Labs federation management system.

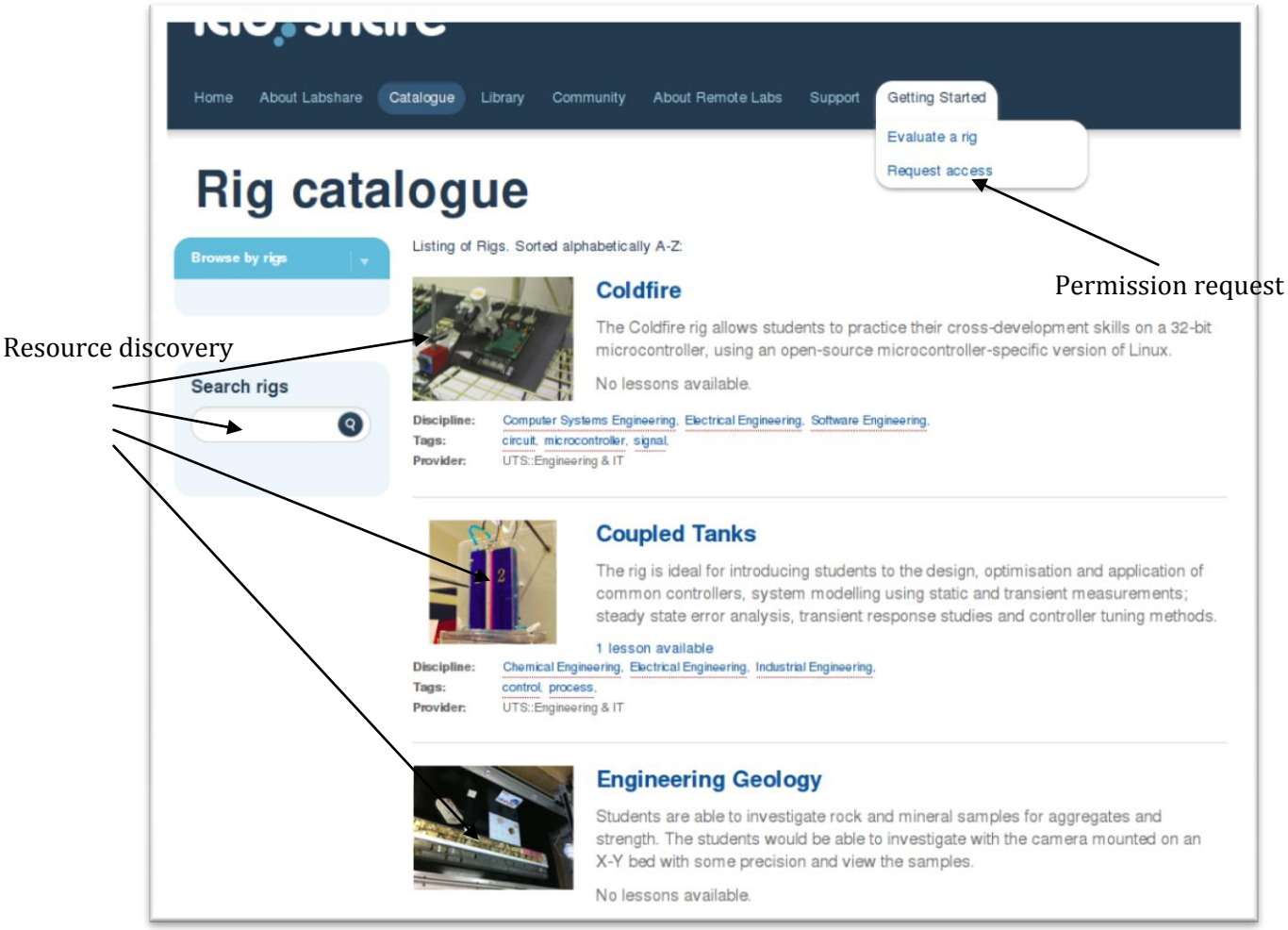
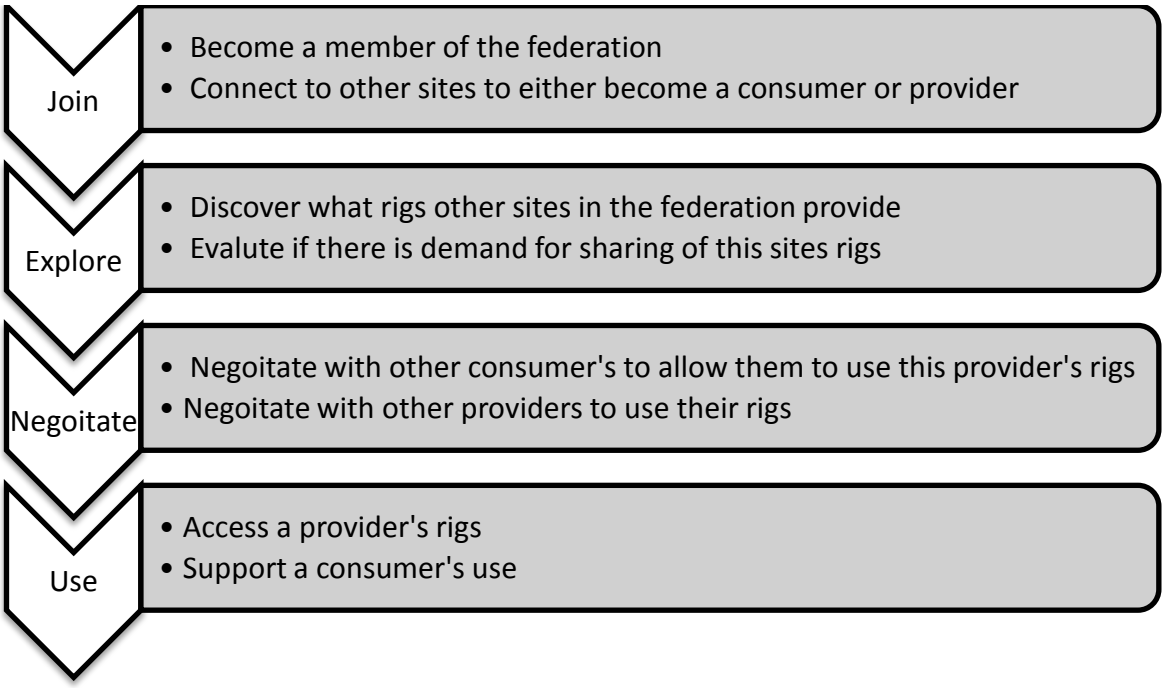


Figure 50 The Labshare Institute catalogue

The work flow in the graphic below gives the basic phases of a federation's site.



Each of these phases has implications on the attributes of a federation. These are articulated in the table below as questions which form the criteria to investigate architectures that may provide federation management.

Table 30 Federation management architectural questions

Phase	Description	Questions Posed
Joining / Membership	This phase is the process of becoming a member of the federation.	• Who decides who can be part of the federation? • What constraints are imposed for membership? • What privileges are provided for membership? • How are errant sites dealt with?
Exploring / Discovery	Requesting a permission is predicated on knowing what is available to be requested. This phase is about discovering this.	• Who may view what a provider provides? • Is the list of provider resources curated and by whom?
Negotiation / Agreement	Negotiation is a question of ownership and authorisation. This phase is having a suitable authorised entity allowing a consumer to use the provider's rigs.	• Who decides on whether to accept or deny a request? • Can this be delegated out to another entity to decide?
Use / Support	The end goal of the federation management and in a general sense SAHARA Labs –	• How is failure dealt with?

[redacted]	shared use of a resource.	

5.1 DISTRIBUTED ARCHITECTURE

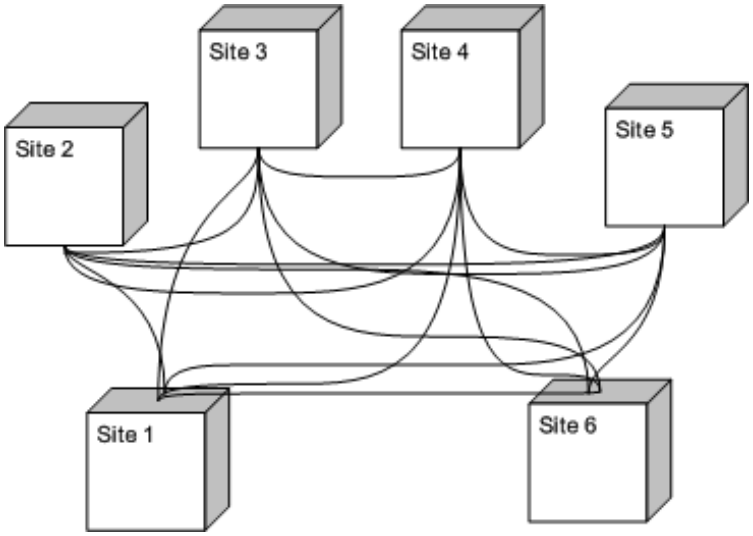


Figure 51 Distributed architecture

The first architecture described to meet the federation management requirements is a completely distributed architecture. Every site is equivalent and every site in the federation talks to every other site. If a site wishes to access a resource hosted at another, it requests the list of resources at that site and then creates a request to access a chosen resource. It is then up to the providing site to choose whether to allow the requesting sites access.

Becoming a member would involve connecting to another site and through some mechanism such as a shared database or distributed hash table the details of all other members are synchronised. Therefore in this architecture there is little effective control on who is a member.

As federation membership connects every site to every other site, the only effective point of control in this architecture is at a provider level. This would force a provider to always handle its requests and not have the option of delegating its requests.

5.2 CENTRALIZED ARCHITECTURE

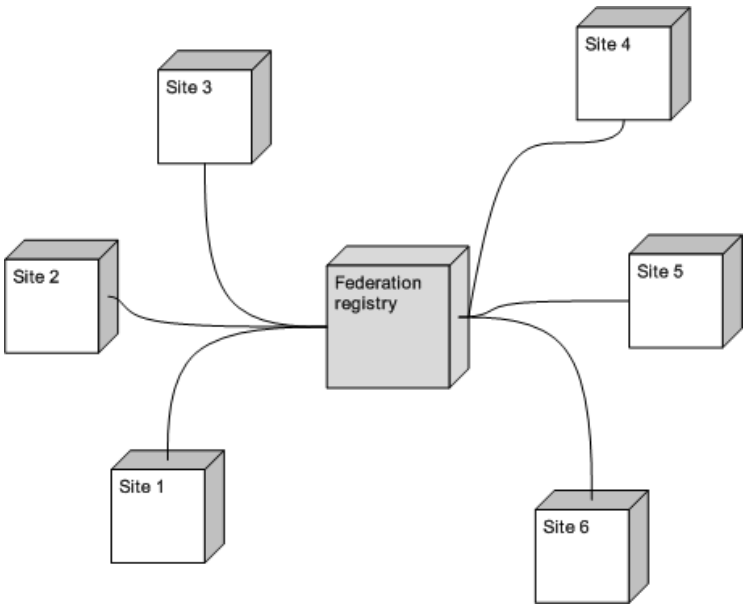


Figure 52 Centralized architecture

This centralized architecture is the antithesis of the distributed architecture. There is no direct communication between sites. Rather, it is a hub and spoke arrangement where a federation registry controls all effective access between sites. If a site wishes to access a resource hosted at another, it requests the federation registry. It is then up to the federation registry to choose whether to negotiate access with the providing site.

Becoming a member of this federation is determined by the conditions set by the federation registry as it is up to the federation registry to decide whether to allow membership. The resources that are useable in this federation are controlled by the federation registry. The registry could set conditions on what's available and who may access it. This gives the federation registry the ability to ensure standards are kept at each site and the federation as a whole has a consistent quality.

5.3 PEER TO PEER ARCHITECTURE

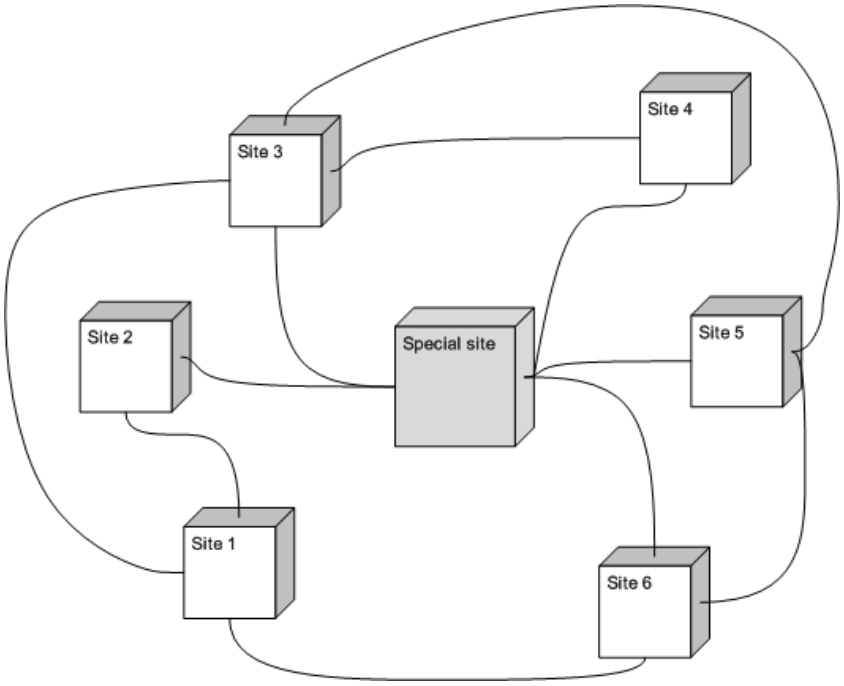


Figure 53 Peer to Peer architecture

The peer to peer architecture is a compromise between the previous two architectures. It allows the organic growth provided by the distributed architecture whilst allowing the controlled and mediated access given by the centralized architecture. The basic tenets of this architecture are every site is equivalent and a site may talk to other sites only if the peer relation has been explicitly created between sites and privilege is assigned between both. Privilege is determined on a per site basis and sets the level of trust between sites. The levels are:

- Viewer – A site given this privilege may view the sites resources.
- Requestor – A site given this privilege may request the sites resources identified by the outcome of the viewer privilege.
- Redirectee – This is a delegation of authorisation to another site. If the site has a resource requested, this request is directed to a redirectee.
- Authorizer – This is another delegation of authorisation where a site may directly add permissions to the site. The only limitation is the permissions that may be created are bounded by the periods the site has declared the resource as being available for federation use.

A special site in an operational sense can be deployed so it is the entity which negotiates permissions with other sites in the federation. Technically, it is built with the same technologies (i.e. SAHARA Labs) as every other site. Operationally it is setup with

peer relationships to all other sites and authority using privilege assignment by the constituent members of the federation who desire the special sites services.

5.4 Evaluation of Architectures

To determine which architectural model best fits with the context SAHARA Labs is current deployed in, the motivations of the current actors must be known. These actors are the providers, the consumers and The Labshare Institute. In the description of each of these, their motivations are used as a basis for critical requirements of a federation's functionality. These requirements are then applied to the architectural models to determine which architecture to implement.

The first set of actors is the federation providers. The providers' motivations are to first satisfy local demand that caused the rigs to be developed, then to provide any surplus capacity to other users in return for some form of compensation. This may be credits for use of another provider's rigs or a monetary value. The critical requirement of providers is *inclusiveness*. They are not much concerned with who uses their rigs provided they are compensated. This is best addressed by the distributed architecture as this allows the most unfettered access to the provider's resources. This is also addressed by the peer-to-peer architecture. It is least addressed by the centralized architecture.

Another critical requirement of providers is delegation of management of the providers sharing (which includes SAHARA Labs required setup and operational tasks such as generating invoices). This is completely not addressed by the distributed architecture. It is however, addressed by the peer-to-peer architecture as it allows a provider to flexibly choose whether to delegate and to whom. Overall, it is the peer-to-peer model which best suits providers as it gives them the most choice on how they wish to operate.

The second grouping of actors is the federation consumers. The consumers' motivations are to use a provider's laboratory to satisfy a local pedagogic need. The critical requirement is that remote laboratory access is consistent and reliable so their students can perform their laboratory tasks. This is not *reliability* as a quality attribute of the SAHARA Labs software (though this does factor in). It is operational reliability of the provider site so whether their systems are maintained and rigs are functioning correctly. This is best addressed using the centralized architecture as the federation registry has the ability to monitor and curate providers and their rigs. If a provider has ongoing maintenance problems which are not addressed, the federation registry can remove them from the registry. Depending on the deployment of the peer-to-peer architecture (privilege assignments), this can be equivalently addressed.

The last significant actor is The Labshare Institute (TLI). Their motivation is best summed up on their website:

> As a not-for-profit organisation, the Labshare Institute will act as an independent service broker to a national network of educational and research institutions who provide and/or utilise shared remote laboratories. Fundamentally, Labshare is about establishing a remote labs sharing community and fostering development of high quality, cost-efficient labs for education. (The Labshare Institute 2011)

In the architecture models, the 'federation registry' and 'special site' are synonyms for TLI. The architectural model that best suits TLI is the centralized architecture. This gives TLI effective and enforced control over the federation and thus the best ability to affect and ensure quality and cost. If a provider fails to supply quality labs or has a suspect history of reliability, in the centralized architecture TLI has the ability to remove the provider and ensure they are not used by a consumer. In the peer-to-peer model, provided the consumers are negotiating through the 'special' site, TLI also has this control. In the distributed architecture, TLI has no control and no real ability to ensure quality labs. In fact, the role of TLI in this architectural model is questionable at best. This is not to say TLI's role in a federation is questionable (it certainly isn't), but it does suggest the distributed architectural model is fundamentally flawed. Without effective control, the quality and therefore usefulness of remote laboratories cannot be ensured and so the federation is not likely to not be sustainable.

The table below shows a matrix comparing the actors and the architectural models. The comparison is uses a weighting of between 0 and 10, with 0 being not at all suitable and 10 being most suitable.

Table 31 Actor and architectural model analysis

Model / Actor	Distributed	Centralized	Peer-to-peer
Providers	6	3	10
Consumers	2	9	9
TLI	0	10	8
(Total)	**8**	**22**	**27**

The table above shows the peer-to-peer architecture is most appropriate and thus it is the architecture that will be implemented.

6. Federation Management Implementation

This chapter describes the implementation of the federation management portion of the SAHARA Labs system. Management is essentially setting up the pre-conditions of use which are creating links between sites and then creating permissions between sites. Implicitly, the federation architecture is to develop administration interfaces to automate or simplify managing the federation.

The federation management implementation is within the framework of the existing SAHARA Labs applications and no new applications are being added. All the modifications reside within the Scheduling Server as it is the component that holds the permissions of use and is the destination of communication between components. This is not to say other applications couldn't be added or if they were that it would necessarily change this project implementation. In fact, if such an application was developed (such as modifying the existing TLI catalogue), the public interfaces developed in this project would be appropriate for communication between that application and the federation peers for the purpose of managing their permissions.

6.1 Architecture in Detail

The previous chapter chooses a peer-to-peer architecture for the federation. This is shown in the diagram below which shows the peer-to-peer connection in the context of existing SAHARA Labs applications.

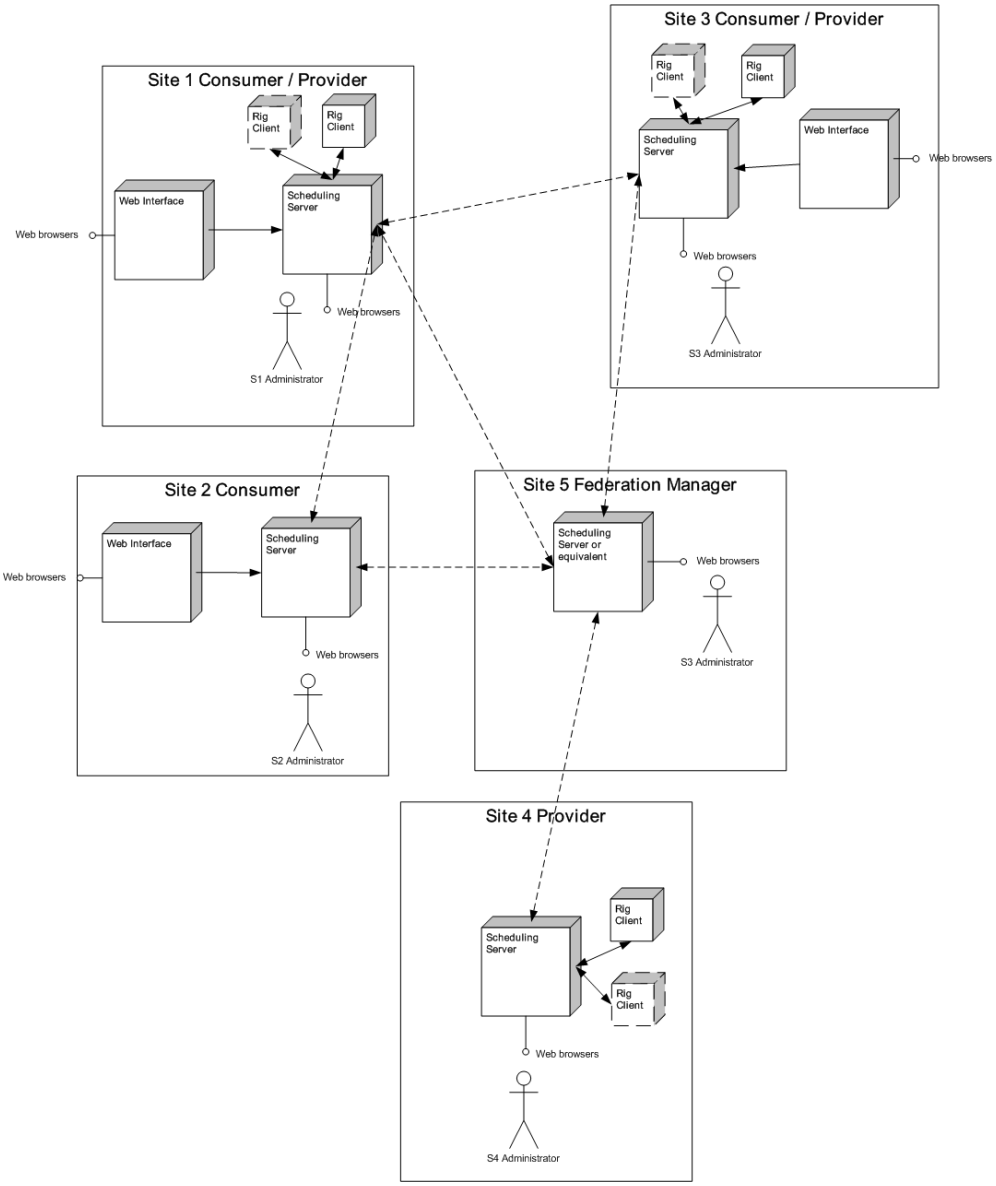


Figure 54 Federation Management detailed architecture

This architecture diagram is a modified form of the one used in the federation use architecture section of Chapter 3. It shows the connections between sites with the intention that these are the *paths* for discovering resources and requesting permissions to use these resources. In the diagram, there is the inclusion of 'Federation Manager' which is neither a consumer nor a provider. Its purpose as a site is to manage discovery and permission requests which delegate access to it. As this is a peer-to-peer its actual implementation may not be different to other sites and may use a Scheduling Server or equivalent which has the same interfaces.

Codifying how this occurs describes the relationship between peer sites. Any peer site can connect to any other peer site. However the ability to discover and request access is governed by the provider determined privileges a consumer is granted which are:

- Viewer – The consumer is allowed to view which resources the provider is allowing to be used by the federation.
- Requestor – The consumer is allowed to request the resources that the provider is allowing to be used by the federation.
- Authoriser – The consumer is directly allowed to add permissions for the resources that the provider is allowing to be used by the federation.
- Redirectee – The consumer is an endpoint of requests that are from consumers who do not have the 'requestor' privilege.

The four privileges allow a provider, on a per-site basis, to either accept a request from a consumer or delegate these to other peers for negotiation. To illustrate this point, consider the table below which is the privileges assigned to four peers of a provider:

Table 32 Site privilege assignments for example scenario

	Fed. Manager	Cons. 1	Cons. 2	Cons. 3
Viewer	X	X	X	
Requestor		X		
Authoriser	X			
Redirectee	X			

In this scenario, Consumer 1, 2 and 3 wish to use a resource hosted at the provider. Consumer 1 can view and request the provider's resource and the provider will determine whether to allow this consumer use. Consumer 2 can view the provider's resource but requesting the provider to use a resource will cause the request to be redirected to the Federation Manager. The Federation Manager then decides whether to allow use to Consumer 2. Consumer 3 cannot view the provider's resources and so cannot request it from the provider. However, the Federation Manager is allowed to view the provider's resources so it can conceivably export the resources as being allowed for federation. If Consumer 3 is allowed to view and request the Federation Managers resources, it can request the provider's resource from the Federation Manager. The Federation Manager than decides whether to allow use of the providers resource to Consumer 3.

As viewing resources and requesting permissions is a user interface action by an Administrator, a user interface needs to be added to SAHARA Labs to provide this. The diagram shows this as being hosted on the Scheduling Server. The naive choice would be to put this on the Web Interface as the existing user interface of SAHARA Labs.

However, in a pure provider deployment a Web Interface is not needed as no users will connect directly to the provider's resource. The Scheduling Server is always needed no matter the deployment scenario. Thus, it is the most ideal place to host it for a user interface that is always required.

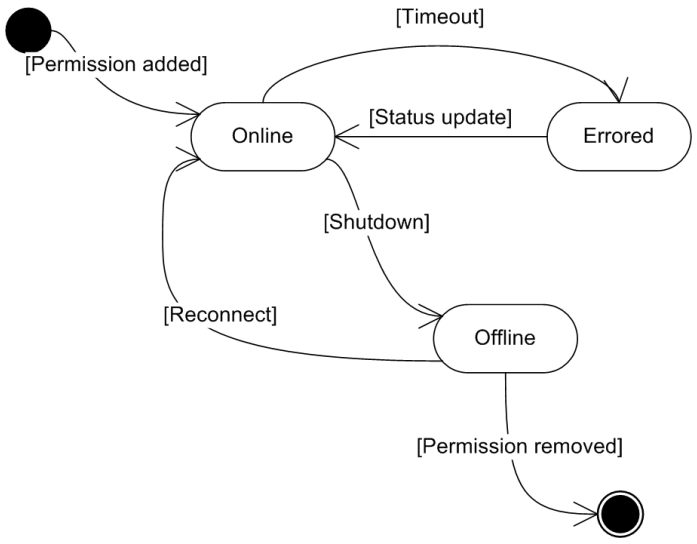


Figure 55 Site states

If there is an active permission between the consumer and provider the that allows a providers resource to be used, the state chart above shows the error handling states between sites and alludes to the error handling strategy. This is at the granularity of sites; a sites resources being offline is handled by a federation use interfaces. The offline and error status are defined as the inability of sending communication to a site and may be network interruptions, firewall blockage or deployment environment failures. In these states no communication is attempted so no use occurs. The difference between these states is the error state is transitioned to as a detected fault by a peer not providing periodic site status updates or failing a communication request. The offline is explicitly entered by telling a site telling its peers it will be online. A reconnect or the resumption of periodic updates transitions a peer back to online and allows federation use to occur again. The start and end states are transitioned in and out using permission addition and removal (or expiry). Peers that do not have these use permissions need not provide status updates as there is no implied level of service between these two sites.

6.1 Scheduling Server Modifications

To implement the federation management modifications for the Scheduling Server involves the following steps:

1. Deciding modifications to existing application architecture that adds the required functionality to the Scheduling Server. This is whether a new OSGi bundle is introduced to provide Multisite functionality or rolled into the existing Multisite bundles.
2. Modifying the persistence of the Scheduling Server to store the required details for federation management. As the Scheduling Server uses a relational database as persistence, this is modifying the database schema and the associated Java Persistence Architecture (JPA) mappings.
3. Developing a communication interface between consumers and providers. The Multisite interfaces implemented for federation use are SOAP services and so will this interface be.
4. Modifying the Server bundle so web pages may be implemented in other modules to run on it.
5. Implementing the federation management SOAP interface.
6. Implementing the federation management web pages.

6.1.2 Application Architecture Modifications

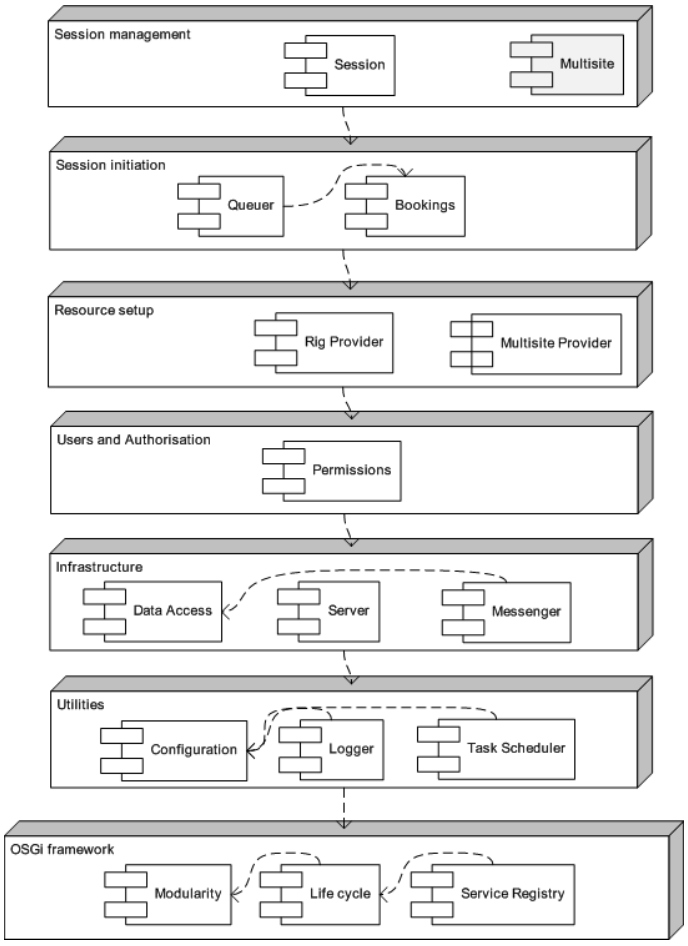


Figure 56 Multisite Federation Management application architecture

The implementation of the federation management is essentially external communication and database interaction. The external communication is through a web interface for user interaction and a SOAP service for peer interaction. None of the required functionality for these has dependency on other modules apart from those in the infrastructure layer. The dependency in the infrastructure layer is with the 'Data Access' module to access the database and the 'Server' module to host the web pages and SOAP service.

Conceptually the functionality belongs in the 'Multisite' module where the federation use implementation is predominately conducted so this is the most ideal place to add the federation management implementation. Also, the Multisite Federation SOAP service uses common types with Multisite and Multisite Callback SOAP services defined in a common XML schema. Adding the federation service into the Multisite module allows the corresponding Java implementation types that parse and serialise these to be reused. If the federation service is implemented in a different module, either the Java implementations of the common types would either need to be exported from the Multisite module polluting the separation of module public interface and implementation or would need to be added separately to that module.

The web interface of the Scheduling Server r3.1 consists of:

Figure 57 Scheduling Server web interface login

Figure 58 Scheduling Server landing page

Figure 59 Scheduling Server web interface diagnostics page

Figure 60 Scheduling Server web interface about page

This web interface is implemented in the Server module of the Scheduling Server. Adding more pages to support the federation management directly to the Server module is untenable as there is no conceptual relationship between the Multisite system and the Server module and it breaks the precedent of cohesive Scheduling Server modules. The options to correctly add pages to the Scheduling Server web interface:

- Implement the pages as a servlet and register these with the Server module. This is equivalent to how the SOAP services are currently added to the Server module to be externally accessible. The advantage is the functionality of registered servlets already exists and no modifications are required to be made to the Server module. The disadvantage is the federation management pages are effectively a completely separate web application. All the things required of a web application would need to be re-implemented such as authentication, styling, layouts, resource downloading and request processing. The commonality with existing Scheduling Server web interface would be that it is accessed on the same address and port but on a different path (i.e. it runs somewhere on the Scheduling Server. Everything else would be foreign).
- Implement a page registration system in the Server module that allows pages to be registered to be hosted as part of the existing Scheduling Server web interface. The mechanism for this would be whiteboard pattern services. The advantage of this is it makes the pages appear seamlessly to the user as part of a coherent web application. The disadvantage is the Server module needs to be modified.

The choice of federation management web interface implementation is implementing page registration functionality in the Server module. The only significant disadvantage of this is actually implementing the change to the Server module but this is likely to be less effort than implementing a whole web application in the Multisite module. Page registration is generic functionality and once implemented will provide another

architectural pillar for further development of SAHARA Labs, irrespective of the utility of the SAHARA Labs federation system.

6.1.3 PERSISTENCE STRUCTURE

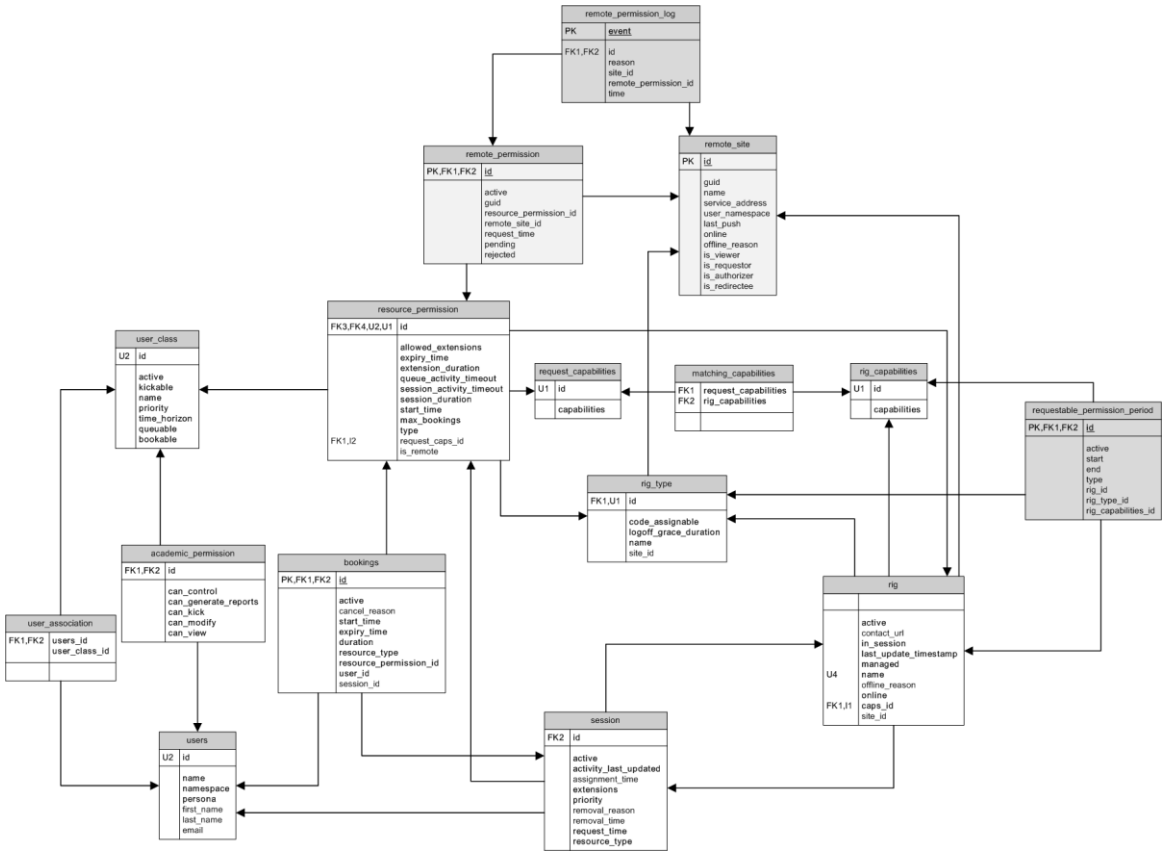


Figure 61 Scheduling Server schema with federation management use persistence changes

The diagram above shows an updated schema for the Scheduling Server and builds on the schema shown in the Chapter 3. It has the additions of two tables and extra fields to the existing `remote_site` and `remote_permission` tables.

The extra fields in the `remote_site` table store life-cycle status information about remote sites such as whether they are online. They also store the authorization the site has with respect to viewing and requesting this site's resources. The table below shows the new fields:

Table 33 remote_site table column additions

Field	Data type	Definition	Optional
online	bit(1)	Whether the site is online.	No
offline_reason	varchar(255)	The reason the site is offline or	Yes

		empty if the site is online.	
last_push	datetime	The time when the last status update was received.	No
is_viewier	bit(1)	Whether the site has the viewer authorization for this site.	No
is_requestor	bit(1)	Whether the site has the requestor authorization for this site.	No
is_authorizer	bit(1)	Whether the site has the authorizer authorization for this site.	No
is_redirectee	bit(1)	Whether the site has the redirectee authorization for this site.	No

The extra fields in the `remote_permission` table store the state and outcome of negotiating a permission. These are shown below:

Table 34 remote_permission column additions

Field	Data type	Definition	Optional
pending	bit(1)	Whether the permission is still in the process of being negotiated.	No
rejected	bit(1)	Whether the outcome of the negotiation process was the permission not being accepted by a peer.	No
request_time	datetime	The time the permission was first requested.	No

The `remote_permission_log` table added to the Scheduling Server's schema stores the history of negotiating a permission. Each communication is stored as a record in this table. The columns definitions are shown in the table below:

Table 35 remote_permission_log table

Field	Data type	Definition	Optional
id	bigint(20)	Record primary key.	No
Event	varchar(20)	The type of event this log record stored.	No
reason	varchar(1024)	Information supplied during the negotiation communication.	No
site_id	bigint(20)	The site which performed the event. If this is not set, the log refers to being initiated by this site.	Yes
remote_permission_id	bigint(20)	The permission this log refers to.	No
time	datetime	The time the event occurred.	No

The `requestable_permission_period` table added to the Scheduling Server's schema stores the periods which a resource may be requested for federation use. The columns definitions are shown in the table below:

Field	Data type	Definition	Optional
id	bigint(20)	Record primary key.	No
active	bit(1)	Whether this period is still in effect.	No
start	datetime	The time this period starts. All requested permissions must start at or later then this time.	No
end	datetime	The time this period ends. All requested permission s must end at or later then this time.	No
type	varchar(10)	The type of resource this requestable permission period provides. It has the same values	No

		as the `resource_permission` type column.	
rig_id	bigint(20)	The rig that is requested if this period has type set to 'RIG'.	Yes
rig_type_id	bigint(20)	The rig type that is requested if this period has type set to 'RIG_TYPE'.	Yes
request_capabilities_id	bigint(20)	The request tags list that is requested if this period has type set to 'CAPS'.	Yes

The `requestable_permission_period` table shows the default exclusion policy for making resources available for federation use. A resource is only ever available to be requested if it is explicitly set as being free for sharing by the provider.

6.1.4 MULTISITE FEDERATION MANAGEMENT EXTERNAL INTERFACE

The Multisite federation management interface is a SOAP service called 'MultisiteFederation'. The operations contained within perform two distinct tasks. The first is site management to connect it to a peer site and update it with the status updates such as when the site is shutting down and starting up. The other is permission negotiation to allow the resources of a site to be viewed and negotiated for consumer access.

As with the federation use SOAP services, the MultisiteFederation service defined in a Web Service Definition Language (WSDL) file. It uses common parameters with the federation use services which are defined in a separate XML schema (XSD) that this service imports. The tables below give the descriptions of each of the service operations and the list of input and output parameters. The input and output parameters only list the parameter type contents and do not follow the same structure as defined in the WSDL. Consult the WSDL files attached as an appendix for the defined parameter structure.

6.3.3.1 Multisite Federation operation descriptions

Table 36 initiateSite Multisite Federation operation

Name	initateSite
Description	The connecting site is added to a peer site of this site with no

	privileges.
Pre-conditions	• A site with the specified site credential does not already exist.
Post-conditions	• A site with the specified site credential is created with no privileges.
Input parameters	• siteID – the credential of the requesting site. • name – the display name of the site. • namespace – the namespace of the sites users and rigs. • address – the endpoint of the sites Multisite service.
Output parameters	• success – whether adding the site was successful. • siteID – the credential of this site. • name – the display name of this site. • namespace – the namespace of this site's users and rigs. • address – the endpoint of this site's Multisite service.

Table 37 siteReconnect Multisite Federation operation

Name	siteReconnect
Description	Alerts a peer site the requesting site has started back and may be used again. It also allows the requesting site to change their details such as service endpoint.
Pre-conditions	• The requesting site is a peer of the requested site.
Post-conditions	• The requesting site may be used.
Input parameters	• siteID – the credential of the requesting site. • name – the display name of the site. • namespace – the namespace of the sites users and rigs. • address – the endpoint of the sites Multisite service.
Output parameters	• success – request acknowledgement

Table 38 siteStatus Multisite Federation operation

Name	siteStatus
Description	Provides a keep-alive message to peers.
Pre-conditions	• The requesting site is a peer of the requested site.
Post-conditions	• If the requesting site was offline, it has been put back

	online.
Input parameters	• siteID – the credential of the requesting site.
Output parameters	• success – request acknowledgement

Table 39 siteShutdown Multisite Federation operation

Name	siteShutdown
Description	Alerts a peer site that the requesting site has shutdown and may no longer be used.
Pre-conditions	• The requesting site is a peer of the requested site.
Post-conditions	• The requesting site is taken offline so it is no longer used.
Input parameters	• siteID – the credential of the requesting site. • reason – the reason for shutting down.
Output parameters	• success – request acknowledgement

Table 40 discoverResources Multisite Federation operation

Name	discoverResources
Description	Provides a list of resources that may be requested for federation use. The list of resources is only provided if the requesting site has the view privilege.
Pre-conditions	• The requesting site is a peer of the requested site.
Post-conditions	None, this operation is only informational.
Input parameters	• siteID – the credential of the requesting site.
Output parameters	• resources – a list of resources that may be requested for federation use and the date ranges for this. If the site does not have the view privilege the list is empty.

Table 41 requestResource Multisite Federation operation

Name	requestResource
Description	Creates a request to use a provider's resource. This is starting the permission negotiating process.
Pre-conditions	• The requesting site is a peer of the requested site.

	• The requested resource is being shared for requested date range.
Post-conditions	• The permission request is created.
Input parameters	• siteID – the credential of the requesting site. • resource – the resource that is to be used. • start – the permission start time. • expiry – the permission expiry time. • name – the name of the permission. • sessionDuration – the queued session guaranteed session duration in seconds. • allowedExtensions – the number of time extensions that may be provided to sessions. • extensionDuration – the duration of each extension in seconds. • isQueueable – whether the permission allows queuing. • isBookable – whether the permission allows reservations. • maxBookings – the number of allowed concurrent reservations. • note – a note to provide the authorising party at the provider.
Output parameters	• success - whether creating the request was successfully created. • permissionID – if successful, the credential to identify the permission in further communication. • requiresRedirect – if not successful, the request may be delegated to another peer to create the requested. • site – if a redirect is required, the site to create the request with instead of this site.

Table 42 notifyAccept Multisite Federation operation

Name	notifyAccept
Description	Notifies a permission requestor that the permission request has been accepted and it may now be used.

Pre-conditions	• The requesting site is a peer of the requested site. • The requested site has a permission request with the requesting site.
Post-conditions	• The permission request is ready for use.
Input parameters	• siteID – the credential of the requesting site. • permissionID – the credential of the permission that is being accepted.
Output parameters	• success – request acknowledgement.

Table 43 notifyModify Multisite Federation operation

Name	notifyModify
Description	Notifies a permission requestor that one or more fields of a permission must be modified before the permission can be accepted.
Pre-conditions	• The requesting site is a peer of the requested site. • The requested site has a permission request with the requesting site.
Post-conditions	• The permission request is modified.
Input parameters	• siteID – the credential of the requesting site. • resource – the resource that is to be used. • start – the permission start time. • expiry – the permission expiry time. • name – the name of the permission. • sessionDuration – the queued session guaranteed session duration in seconds. • allowedExtensions – the number of time extensions that may be provided to sessions. • extensionDuration – the duration of each extension in seconds. • isQueueable – whether the permission allows queuing. • isBookable – whether the permission allows reservations. • maxBookings – the number of allowed concurrent reservations.

	• note – a textual note.
Output parameters	• success – request acknowledgement.

Table 44 notifiyCancel Multisite Federation operation

Name	notifyCancel
Description	Notifies a permission requestor that the permission request has been denied.
Pre-conditions	• The requesting site is a peer of the requested site. • The requested site has a permission request with the requesting site.
Post-conditions	• The permission request is denied and no further communication about it will be accepted.
Input parameters	• siteID – the credential of the requesting site. • permissionID – the credential of the permission that is being denied.
Output parameters	• success – request acknowledgement.

Table 45 getRequests Multisite Federation operation

Name	getRequests
Description	Gets all the permissions requests the requesting site has with the requested site.
Pre-conditions	• The requesting site is a peer of the requested site.
Post-conditions	None, this request is informational only.
Input parameters	• siteID – the credential of the requesting site.
Output parameters	• requests – list of permission requests the requesting site has.

6.1.5 SERVER MODULE

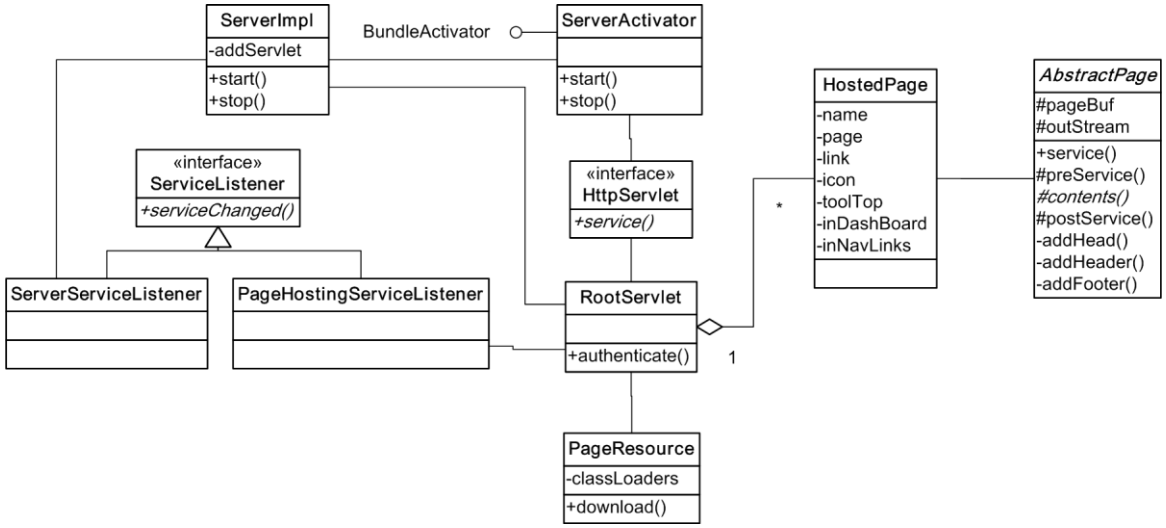


Figure 62 Server module class structure

The Server module was modified to allow web pages to be registered for hosting as part of the Scheduling Servers user interface. To implement a page for hosting, the abstract class `AbstractPage` is extended with the '`contents`' method overridden with a method that provides the page contents. An instance of `HostedPage` is registered on the OSGi framework service registry with the page implementation and other details such as dashboard link name. The Server module detects this service registration and hosts it as part of the web interface.

Apart from HTML content, web pages include static resources such as JavaScript files, style sheets and images. The `PageResource` class which downloads these resources was modified so it looks at the modules with hosted pages class loaders to find these resources. It expects the following layout:

- `/META-INF/web/css` – CSS style sheets.
- `/META-INF/web/img` – Images.
- `/META-INF/web/js` – JavaScript files.

Where '/' is the root of the modules class path.

6.1.6 MULTISITE MODULE

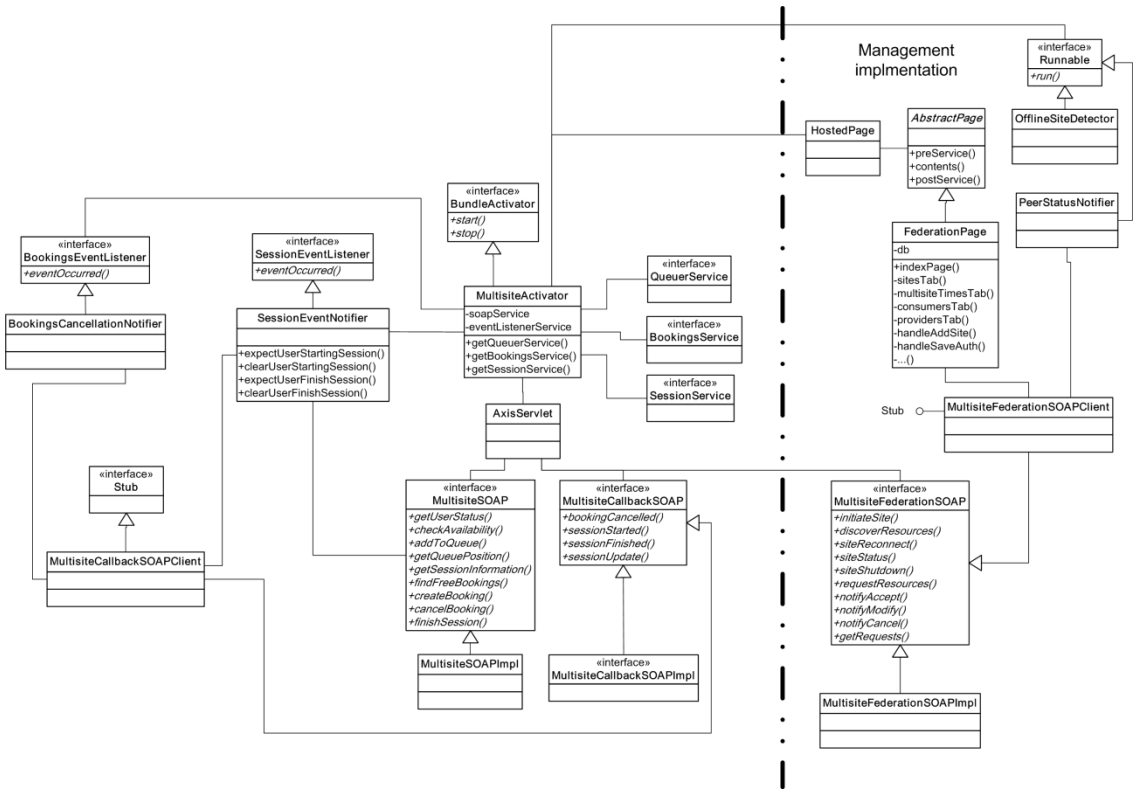


Figure 63 Federation management class diagram

The class diagram above shows the modifications to the Multisite module that have been made to add federation management. This is shown on the right side of the diagram. The '`MultisiteFederationSOAP`' class is the implementation of the Multisite Federation service using the Apache Axis2 library (the same as the other Multisite services). The '`MultisiteFederationSOAPClient`' is the corresponding client to this service.

The classes '`FederationPage`' is the implementation of a Scheduling Server user interface pages. The federation page is implemented as a single web page with all the required federation management interfaces consolidating as tabs on this page. The '`handle...()`' methods such as '`handleAddSite`' are AJAX invoked and return a JSON response to implement the server side of the dynamic web page behaviours.

The classes 'PeerStatusNotifier' and 'OfflineSiteDetector' are periodic tasks that implement the site status notification and site error detection respectively.

6.2 VERIFICATION

The implementation of the federation management system was only partially completed so evaluating its functionality with scenario testing is not possible. This section goes through screenshots of the user interface to highlight what was implemented.

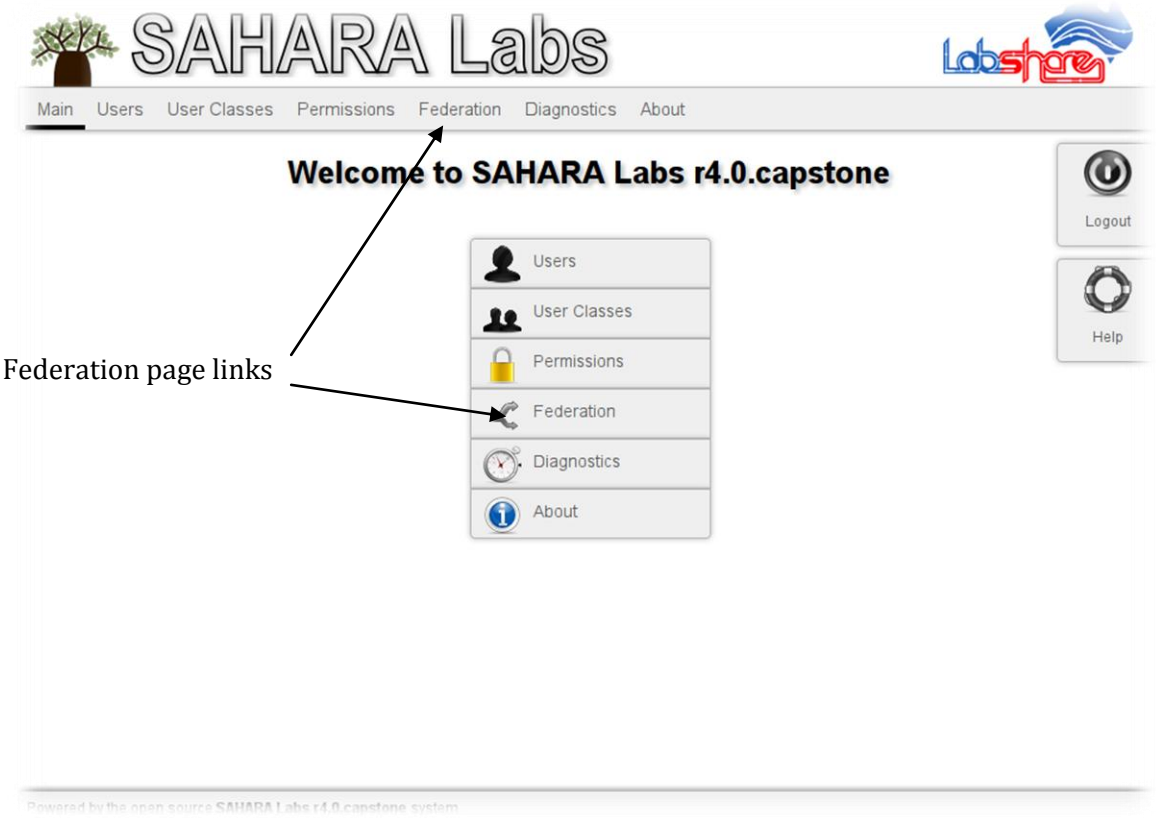


Figure 64 Scheduling Server user interface dashboard

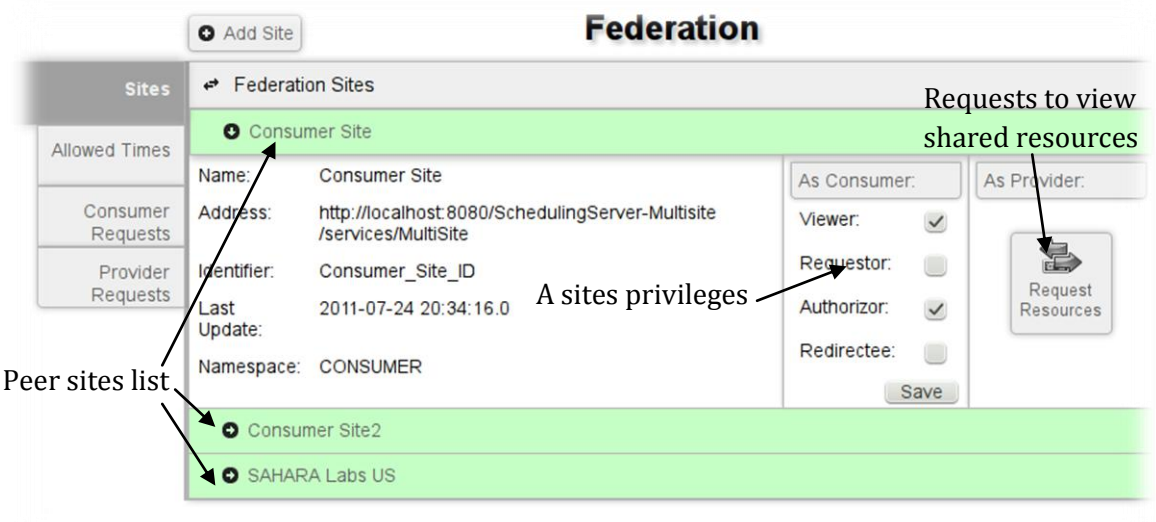


Figure 65 Federation sites list

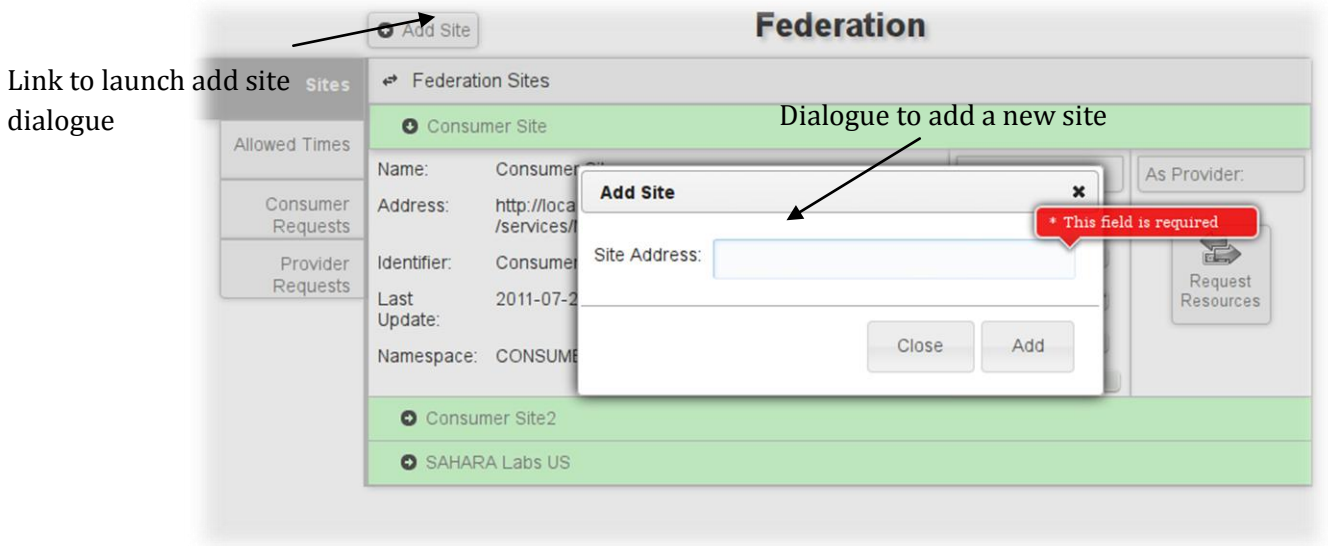


Figure 66 Adding a new peer site

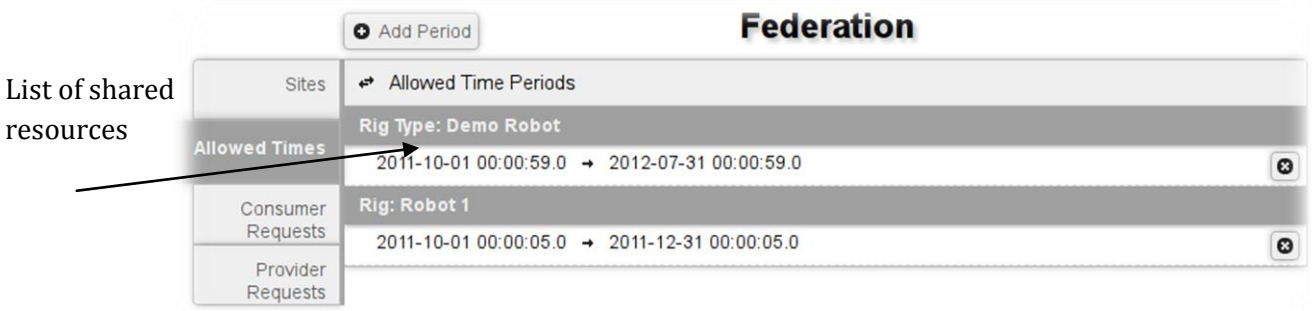
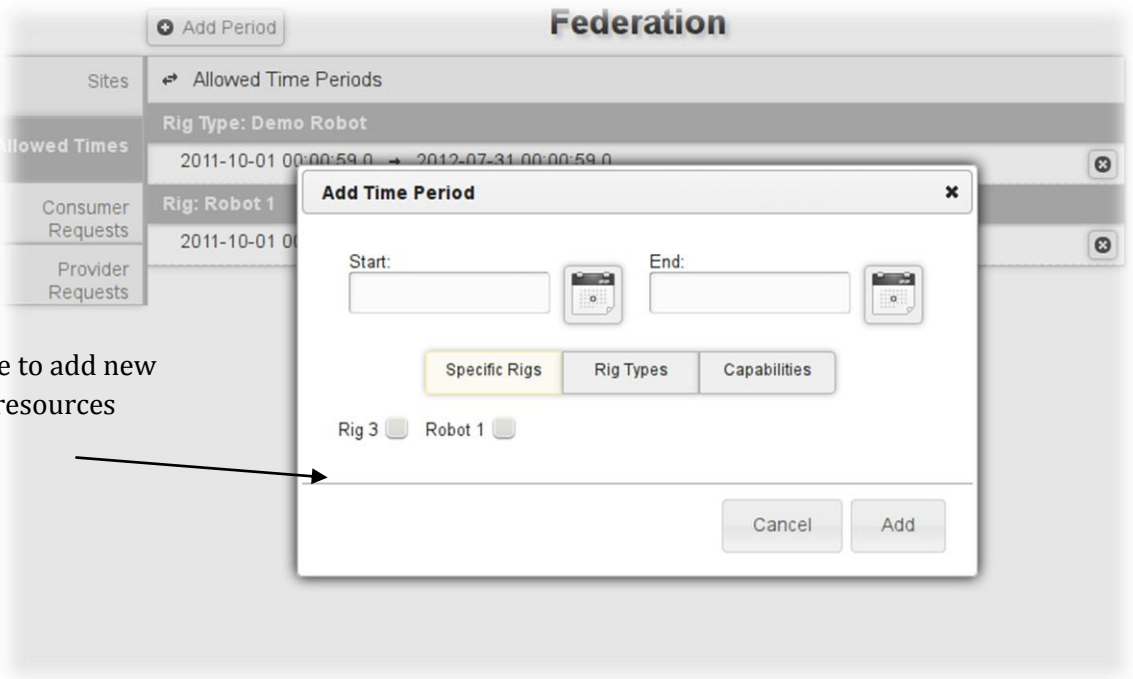
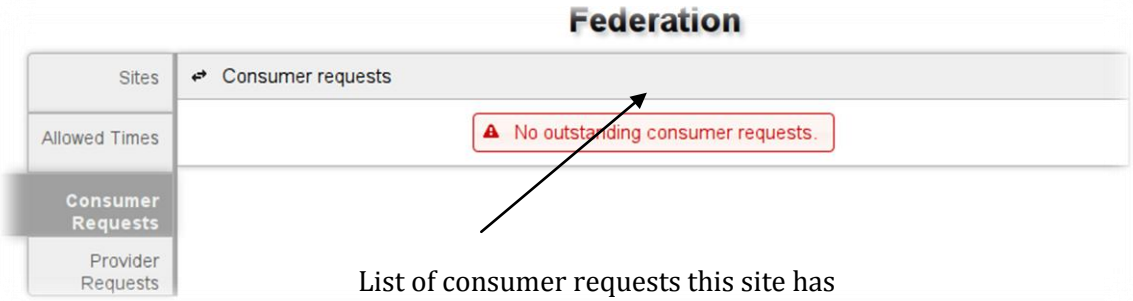


Figure 67 List of resources a provider is sharing



Dialogue to add new shared resources

Figure 68 Declaring a shared resource dialogue



List of consumer requests this site has made. This was not implemented.

Figure 69 List of consumer requests

The functionality not implemented in federation management was the actual creating of a request for a provider's resource, negotiating this request and ultimately deciding the outcome of this request. It was not implemented not because of underlying weakness of architecture or any especial technical difficulties. It was not implemented because of the imposed project end date.

The underlying framework, data model and communication operations were however implemented.

7. CONCLUSION

This project developed into two distinct and intertwined subprojects. The first was investigated and implementing using extending SAHARA Labs in a federation system where a rig at a remote laboratory could be successfully used by logging into another remote laboratory provided appropriate was performed. To do this, the SAHARA Labs architecture was explored and four architectures were explored. Two had serious limitations and were implausible so were excluded from further investigation. The other two had their advantages and disadvantages evaluated in terms of software quality attributes. One of these, a peer-to-peer architecture, had a considerable number of advantages when applied to these attributes and so was chosen as the implementation.

The implementation of this architecture, code named 'Multisite', was performed by modifying the existing SAHARA Labs component. Of these, the Scheduling Server and Web Interface were modified and the Rig Client was found to not require any modifications. When the modifications were completed they were tested and verified to work. Two existing non-trivial rigs were migrated to the SAHARA Labs version implemented in this project and were shown to work in this architecture. One of these was the 'Hydro' rig which currently serves as a model of future development of rigs (web based user interfaces). It was shown that this model is perfectly applicable to the federation use architecture. Perhaps, the federation use architecture suits this model even better then not implementing it considering the locality of users to their consumer sites (this was only tangentially important to the project so was not further explored). The other was the 'Inclined Plane' rig which is implemented with technologies commonly used in many other laboratories (remote desktop) and was shown that this works perfectly well with the implemented federation use architecture. As a proof of concept of SAHARA Labs federation use of rigs, the implemented system was successful.

The second subproject conducted took the preconditions of federation use established in the first subproject and looked at federation management architectures that provide these. The preconditions required of federation use were site connection and identification and agreed permissions. Navigating back from this destination to the status quo of no federation, a work flow was developed to reach the desired destination. Determining this allowed typical distributed architectures to be analysed for their applicability with the outcome of choosing an appropriate architecture. The

architecture chosen was a peer-to-peer architecture providing a nice, if perhaps too convenient, synergy with the federation use architecture.

After the architecture decision, the Multisite codebase was modified to implement this. This was not complete, but it was sufficiently implemented to illustrate what it is the federation management architecture provides.

On a side note, to implement the federation use and management architectures quite a few bits of common functionality were implemented that whilst applicable to the federation system, would also be applicable to many other possible modifications to SAHARA Labs. This increases the code quality of the overall system. For example, if an interoperability standard were to be implemented, the same internal interfaces the federation system uses would likely be reused as part of the interoperability standard. A partial list of the common functionality is:

- Queuer internal service that allows other modules to add users to the queue.
- Bookings internal service that allows other modules to create and cancel reservations.
- Session internal service that allows other modules to terminate sessions.
- Session event broadcasts that allows other modules to register for notification of session events.
- User interface page hosting that allows other modules to register a page to display on the Scheduling Server web user interface.

The SAHARA Labs Multisite system was never intended to be a production quality system ready for deployment at existing SAHARA Labs remote laboratories. It was intended as a proof-of-concept to show how such a system could be built. In this goal it has been markedly successfully. To actually develop a production quality system (assuming the model chosen in this capstone is the production model) it would require:

- Fixing bugs either caused or exposed by the federation system. The time zone bug discovered in the federation use testing is a bug exposed. No other bugs are known at this time, but no comprehensive testing has been conducted to say there aren't any more common use bugs.
- Completion of the federation management interface.
- Implementation of transport layer security of the federation services. The security in this system is at the application level but assumes the entities communicating in the federation behave correctly and aren't intentionally

malicious. Having a service publically available on the Internet means this assumption cannot be made. Therefore, transport layer security is required in addition to application level security, to deal with web security concerns of authenticity, integrity and privacy. The most appropriate for this is SSL for integrity and privacy and certificate authentication for authenticity. This is a standard solution to address transaction layer security and the implementation libraries SAHARA Labs is built on (Jetty and Apache Axis2) have good support for this.

- Some of the control applications that drive rigs during use generate data files which the user subsequently downloads for offline use. The federation system should be able to do this so the consumer users can obtain these results. In some cases, this is already handled. With the IPR that was migrated, consumers see a list of files generated and may download them. In other cases it is not handled. In the case of the UTS Remote Laboratory, the current mechanism of provided data files would not work. The base problem is SAHARA Labs does not address this area of functionality so multiple solutions have been implemented. Until this is addressed it cannot be handled by the federation system.

REFERENCES

Gomes, L. Bogosyan S. 2009, 'Current Trends in Remote Laboratories', IEEE Transactions on Industrial Electronics, vol 56, no. 12, pp. 4746 – 54

Harward, J. Del Alamo, J. Lerman, S. Bailey, P. Carpenter, J. DeLong, K. Felknor, C. Hardison, J. Harrison, B. Jabbour, I. Long, P. Mao, T. Naamai, L. Northbridge, J. Schulz, M. Talavera, D. Varadharajan, C. Wang, S. Yehia, K. Zbib, R. Zych, D. 2008, 'The iLab Shared Architecture: A Web Services Infrastructure To Build Communities of Internet Accessible Laboratories', Proceedings of the IEEE, vol 96, no. 6, pp 931 – 50

Vargas, H. Sanchez, J. Jara C.A. Candelas, F.A. Torres, F. Dormido, S. 2007, 'A Network Of Automatic Control Web-Based Laboratories', IEEE Transactions on Learning Technologies, vol. 6, no. 1, pp 1 – 14

Guimaraes, E. Cardozo, E. Moraes, D. Coelho, P. 2007, 'Design and Implementation Issues for Modern Remote Laboratories', IEEE Transactions on Learning Technologies, vol. 6, no. 1, pp. 1939 – 55

Toderick, Lee. Tijjani, M. Tabrizi, M. 2005, 'A Consortium of Secure Remote Access Labs for Information Technology Education', In Proceedings of SIGITE Conference 2005, pp. 295 – 99

Reekie, J. McAdam, R. 2006, *A Software Architecture Primer*, 1st edn, Angophora Press, Sydney.

OSGi Alliance, 2004, *Listeners Considered Harmful: The "Whiteboard" Pattern,* OSGi Alliance, viewed 20th November 2011, <http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf>

The Labshare Institute 2011, *Labshare – Home*, The Labshare Institute, viewed 22nd November 2011, <http://www.labshare.edu.au/>

APPENDIXES

A. DEVELOPMENT ENVIRONMENT

This appendix briefly details the development environment used in this projects development. It includes:

- Revision control for source code.
- Development environment to facilitate development in the SAHARA Labs source code languages, predominately Java (for Scheduling Server), PHP, HTML and JavaScript (for Web Interface).
- Development test environment to allow portions of code or the system to be run so it may be tested.

REVISION CONTROL

The requirements for revision control of the newly developed source code for this capstone were:

- **Secure –** It should be securely stored so if the development environment has a failure no work will be lost.
- **Available –** it should be available so as not to impede development and be easily transportable across development and test machines (i.e. desktop, laptop and future demonstration machines).
- **Separate –** it should be separate from mainline SAHARA Labs development as it is not yet intended for production release and should not interfere with any production release.
- **Interoperable –** it should be interoperable with mainline SAHARA Labs development to import changes that have been made to mainline making possible later integration easier.
- **Tool support –** it should integrate with the other tools in the development environment to allow making development easier.
- **Not public –** it should not publically suggest features for SAHARA Labs future releases.

The following table evaluates the options that were considered:

	Secure	Available	Separate	Interoperable	Tool Support	Not Public
FEIT Subversion on trunk – this is mainline development repository for SAHARA Labs	**X**	**X**		**X**	**x**	**X**
FEIT Subversion on a separate branch	**X**	**X**	**X**	**X**	**x**	**X**
SAHARA Labs Sourceforge project Subversion	**X**	**X**		**X**	**x**	
Private SVN Repository		**X**	**X**		**x**	**X**
Private GIT Repository		**X**	**X**	**X**		
No revision control			**X**			

The revision control system that was chosen is the FEIT SAHARA Labs Subversion repository. It best satisfied the requirements by:

- Being on Faculty of Engineering and IT servers so it is securely backed up nightly and (almost) always available. The only consideration with this point is my access could be revoked if I am no longer employed with Labshare. If this occurs, then the choice of revision control will need to be reconsidered. The likely choice then will be the Sourceforge SVN repository.
- Being on a separate branch means the code is not interspersed with the mainline development code. It allows easy merging of the changes in mainline to be branch (a feature of Subversion).
- Subversion is supported in Eclipse with a plug-in so will make checking in code, updating code and generate diffs (a list of changes) between revisions easy.
- Partially not public, the members who have access are in a tightly controlled group of Labshare employees, all of who should be aware of the nature of the branch.

The branch that was setup is called '*multisite*' and exists in both the Scheduling Server and WI projects in the repository located at *'https://develop.eng.uts.edu.au/projects/personal/sahara/svn'*. Periodically there were merges between trunk and this branch.

DEVELOPMENT ENVIRONMENT

The development environment choice was straight forward and a continuation of the existing SAHARA Labs development environment. The environment is the Eclipse IDE version 3.6 with plug-ins needed to support development. This is because:

- The Scheduling Server is implemented as a set of OSGi bundles which run on the OSGi framework. As Eclipse itself is implemented with OSGi, it provides excellent support for developing OSGi bundles including having a run configuration to independently run OSGi bundles without using an external OSGi framework.
- Eclipse has an excellent Java editor which provides code completion, incremental compilation with error and warning annotations, code generation for simple code such as (getters and setters) and code formatting.
- Eclipse has a debugging environment for Java code.
- Eclipse with suitable plug-ins has a PHP and web development editors.
- Eclipse has a plug-in to integrate Subversion into the development environment.

The Eclipse plug-ins used to setup the development environment are:

Table 46 Eclipse Plug-in List

Plug-in	Description	Version
Eclipse Java Development Tools	Provides the Java editor, debugging support and JUnit tools	3.6.1
PDE	Provides the plug-in development tools for OSGi bundle development	3.6.1
Eclipse Modelling Framework	Provides XML schema editors used for Web Services development	2.6.0
Eclipse Web Tools Platform	Provides HTML and JavaScript editors	3.2.2
Axis2 Tools	Provides code generation of Axis2 web service	1.1.100

	stubs	
Subclipse	Provides SVN support in Eclipse	1.6.15
EclEmma	Provides code coverage metrics of JUnit test case runs	1.5.1
PDT	Provides a PHP editors and PHP syntax checker	2.2.0

DEVELOPMENT TEST ENVIRONMENT

The development test environment that was setup was to support running two simultaneous SAHARA Labs instances on the same computer, one to be the provider, the other as the consumer. The table below lists each of the SAHARA Components and how they were run:

	Consumer	**Provider**
Scheduling Server Database	A database on a MySQL database server called 'consumer'.	A database on a MySQL database server called 'provider'.
Scheduling Server	Run within Eclipse using the properties file 'conf/consumer.properties' set via a system property.	Run within Eclipse using the properties file 'conf/provider.properties' set via a system property
Web Interface	A virtual host on a Apache Server setup with mod_php which listens on port 82.	A virtual host on the a Apache Server with mod_php which listens on port 80
Rig Client	N/A	Standard SAHARA Labs version 3.0 Rig Client.

All of the above components were straight forward to setup except the Scheduling Server because it took its configuration from a file called '`schedulingserver.properties`' in a directory called '`conf`' located relative to the Scheduling Server working directory. Each Scheduling Server has to have a separate configuration file because they are needed to run on a different TCP port and use a different database. This required a modification to the Scheduling Server configuration module to load the location of configuration file using a system property. This was implemented and the system property '`config.loc`' now specifies the location of the configuration file. If this property is not set, the old behaviour is used as a fallback.

B. Development Metrics

This appendix gives some metrics on development this project conducted. This projects code was developed in the branch 'multisite' and was branched from SAHARA Labs r3.1. The code comparisons are between the branch (called r4.0.capstone) and that version of the code base.

Lines of Code:

The program 'sloccount' on Linux was used to count source lines of code. The results are shown below.

NOTE: sloccount does not count lines of code in Javascript and CSS files so they are not shown.

Table 47 Lines of code developed in this project

SAHARA Labs r3.1			SAHARA Labs r4.0.capstone			Delta
Scheduling Server:						+32517
SLOC	Directory	SLOC-by-Language	SLOC	Directory	SLOC-by-Language	
49028	Bookings	java=48982,xml=46	49237	Bookings	java=49191,xml=46	+209
23518	Permissions	java=23381,xml=137	23736	Permissions	java=23599,xml=137	+218
(Module added in r4.0.capstone)			20353	MultiSite	java=20215,xml=138	+20353
18837	RigProxy	java=18809,xml=28	18837	RigProxy	java=18809,xml=28	
18638	Queuer	java=18592,xml=46	19581	Queuer	java=19535,xml=46	+943
11059	Reports	java=11026,xml=33	11059	Reports	java=11026,xml=33	
(Module added in r4.0.capstone)			9989	MultiSiteProvider	java=9980,xml=9	+9989
8875	RigManagement	java=8833,xml=42	8907	RigManagement	java=8865,xml=42	+32
6246	DataAccess	java=6075,xml=171	7071	DataAccess	java=6884,xml=187	+809
6852	RigProvider	java=6819,xml=33	6721	RigProvider	java=6688,xml=33	-131
6301	Session	java=6271,xml=30	6410	Session	java=6380,xml=30	+109
2633	servicewrapper	ansic=2532,sh=101	2633	servicewrapper	ansic=2532,sh=101	
2023	Server	java=2014,xml=9	2234	Server	java=2225,xml=9	+211
947	RigOperations	java=938,xml=9	869	RigOperations	java=860,xml=9	-78
742	Logger	java=733,xml=9	742	Logger	java=733,xml=9	
573	Framework	java=496,xml=77	575	Framework	java=496,xml=79	
439	Messenger	java=430,xml=9	439	Messenger	java=430,xml=9	
423	top_dir	xml=423	431	top_dir	xml=431	
408	Configuration	java=400,xml=8	408	Configuration	java=400,xml=8	
146	TaskScheduler	java=137,xml=9	146	TaskScheduler	java=137,xml=9	
Totals grouped by language (dominant language first):			Totals grouped by language (dominant language first):			
java:	153936 (97.62%)		java:	186453 (94.36%)		+32517
ansic:	2532 (1.61%)		ansic:	2532 (1.28%)		
Rig Client:						0
SLOC	Directory	SLOC-by-Language	SLOC	Directory	SLOC-by-Language	

37282	src_au	java=37282	37282	src_au	java=37282	
7290	test	xml=5647,sh=1643	7290	test	xml=5647,sh=1643	
2745	servicewrapper	ansic=2632,sh=113	2745	servicewrapper	ansic=2632,sh=113	
873	META-INF	xml=873	873	META-INF	xml=873	
231	top_dir	xml=231	231	top_dir	xml=231	
75	resources	xml=75	75	resources	xml=75	
0	_axis2	(none)	0	_axis2	(none)	
0	bin	(none)	0	bin	(none)	
0	conf	(none)	0	conf	(none)	
0	dist	(none)	0	dist	(none)	
0	doc	(none)	0	doc	(none)	
0	installer	(none)	0	installer	(none)	
0	lib	(none)	0	lib	(none)	
Totals grouped by language (dominant language first):			Totals grouped by language (dominant language first):			
java:	37282 (76.88%)		java:	37282 (76.88%)		
ansic:	2632 (5.43%)		ansic:	2632 (5.43%)		
sh:	1756 (3.62%)		sh:	1756 (3.62%)		
Web Interface: **NOTE:** The majority of Web Interface changes were in JavaScript and this is not counted in this table.						>30
SLOC	Directory	SLOC-by-Language	SLOC	Directory	SLOC-by-Language	
3230	application	php=3230	3260	application	php=3260	+30
1319	library	php=1319	1319	library	php=1319	
581	institution	php=549,sh=32	581	institution	php=549,sh=32	
222	docs	php=165,sh=57	222	docs	php=165,sh=57	
14	public	php=14	14	public	php=14	
Totals grouped by language (dominant language first):			Totals grouped by language (dominant language first):			
php:	5277 (98.34%)		php:	5307 (98.35%)		+ 30
sh:	89 (1.66%)		sh:	89 (1.65%)		

COMMIT LOGS

The program 'StatSVN' was used to obtain on information from the Subversion commit logs of the Scheduling Server multisite branch.

- Head revision: 3642
- Report Period: 2011-05-10 to 2011-10-27
- Total Files: 622
- Total Lines of Code: 126011
- Developers: 1

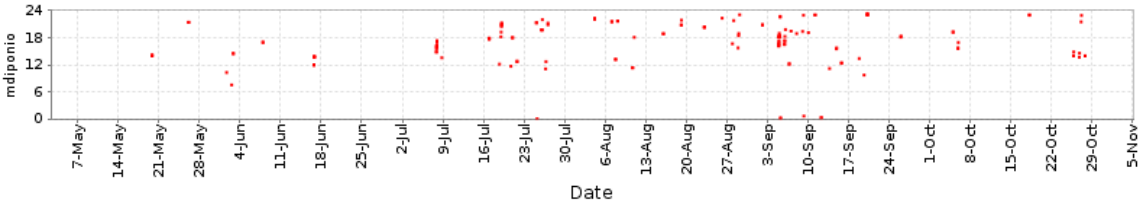


Figure 70 Commit activity of Scheduling Server Multisite branch

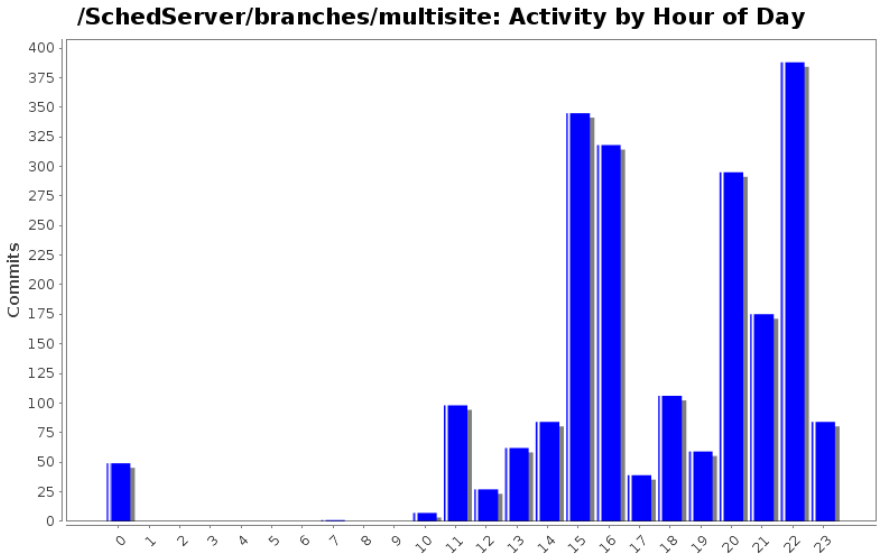


Figure 71 Commits by time of day

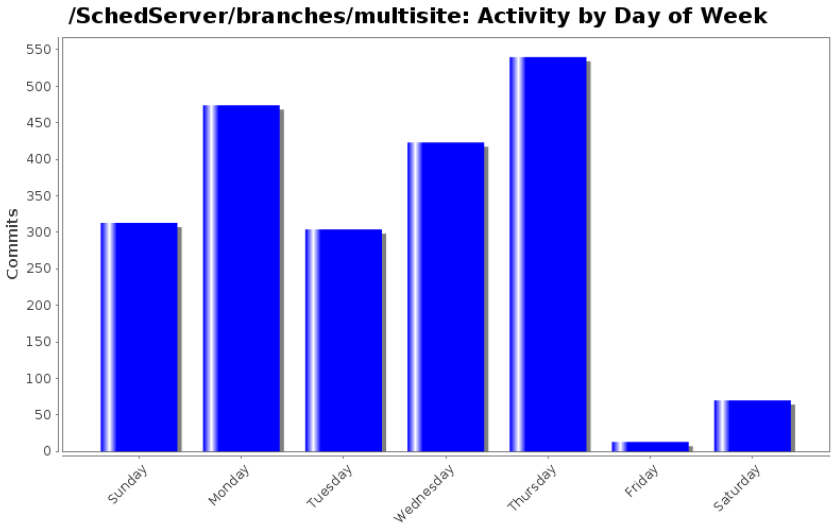


Figure 72 Commits by day of week

A. Multisite SOAP Services WSDL Files

Multisite Service

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="Multisite" targetNamespace="http://remotelabs.eng.uts.edu.au/schedserver/multisite"
xmlns:mut="http://remotelabs.eng.uts.edu.au/schedserver/multisite" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:types>
    <!-- Common MultiSite schema. -->
                <schema xmlns="http://www.w3.org/2001/XMLSchema"
                        xmlns:mut="http://remotelabs.eng.uts.edu.au/schedserver/multisite"
                        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
                        elementFormDefault="unqualified"
                        targetNamespace="http://remotelabs.eng.uts.edu.au/schedserver/multisite">
                        <element name="checkAvailability" type="mut:PermissionIDType" />
                        <element name="checkAvailabilityResponse" type="mut:AvailabilityResponseType" />
                        <element name="addToQueue" type="mut:QueueRequestType" />
                        <element name="addToQueueResponse" type="mut:UserStatusType" />
                        <element name="getUserStatus" type="mut:UserIDType" />
                        <element name="getUserStatusResponse" type="mut:UserStatusType" />
                        <element name="getQueuePosition" type="mut:UserIDType" />
                        <element name="getQueuePositionResponse" type="mut:QueueType" />
                        <element name="getSessionInformation" type="mut:UserIDType" />
                        <element name="getSessionInformationResponse" type="mut:SessionType" />
                        <element name="finishSession" type="mut:UserIDType" />
                        <element name="finishSessionResponse" type="mut:OperationResponseType" />
                        <element name="createBooking" type="mut:CreateBookingType" />
                        <element name="createBookingResponse" type="mut:BookingResponseType" />
                        <element name="cancelBooking" type="mut:BookingIDType" />
                        <element name="cancelBookingResponse" type="mut:OperationResponseType" />
                        <element name="findFreeBookings" type="mut:FindFreeBookingsType" />
                        <element name="findFreeBookingsResponse" type="mut:FindFreeBookingsResponseType" />
                        <complexType name="PermissionIDType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="permissionID" type="string" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="SiteIDType">
                                <sequence>
                                        <element name="siteID" type="string" />
                                </sequence>
                        </complexType>
                        <complexType name="QueueRequestType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="permission" type="mut:PermissionIDType" />
                                                        <element name="user" type="mut:UserIDType" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="UserIDType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="userID" type="string" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="UserStatusType">
                                <sequence>
                                        <element minOccurs="0" name="operation" type="mut:OperationResponseType" />
                                        <element name="inQueue" type="boolean" />
                                        <element name="inSession" type="boolean" />
                                        <element name="inBooking" type="boolean" />
                                        <element minOccurs="0" name="queuedResource" type="mut:ResourceType" />
                                        <element minOccurs="0" name="bookedResource" type="mut:ResourceType" />
                                        <element minOccurs="0" name="session" type="mut:SessionType" />
                                </sequence>
```

```
	</complexType>
	<complexType name="OperationResponseType">
		<sequence>
			<element name="wasSuccessful" type="boolean" />
			<element minOccurs="0" name="reason" type="string" />
		</sequence>
	</complexType>
	<complexType name="AvailabilityResponseType">
		<sequence>
			<element name="viable" type="boolean" />
			<element name="hasFree" type="boolean" />
			<element name="isQueueable" type="boolean" />
			<element name="isBookable" type="boolean" />
			<element name="isCodeAssignable" type="boolean" />
			<element name="queuedResource" type="mut:ResourceType" />
			<element maxOccurs="unbounded" minOccurs="0" name="queueTarget"
				type="mut:QueueTargetType" />
		</sequence>
	</complexType>
	<complexType name="QueueTargetType">
		<sequence>
			<element name="resource" type="mut:ResourceType" />
			<element name="viable" type="boolean" />
			<element name="isFree" type="boolean" />
		</sequence>
	</complexType>
	<complexType name="ResourceType">
		<sequence>
			<element name="type" type="string" />
			<element name="name" type="string" />
		</sequence>
	</complexType>
	<complexType name="SessionType">
		<sequence>
			<element name="isReady" type="boolean" />
			<element name="isCodeAssigned" type="boolean" />
			<element minOccurs="0" name="resource" type="mut:ResourceType" />
			<element minOccurs="0" name="rigType" type="string" />
			<element minOccurs="0" name="rigName" type="string" />
			<element minOccurs="0" name="contactURL" type="string" />
			<element name="duration" type="int"/>
<element name="extensions" type="int" />
<element name="timeLeft" type="int"/>
			<element name="inGrace" type="boolean" />
			<element minOccurs="0" name="warningMessage" type="string" />
		</sequence>
	</complexType>
	<complexType name="QueueType">
		<sequence>
			<element name="inQueue" type="boolean" />
			<element name="inSession" type="boolean" />
			<element name="inBooking" type="boolean" />
			<element minOccurs="0" name="queuedResource" type="mut:ResourceType" />
			<element name="position" type="int" />
			<element name="time" type="int" />
		</sequence>
	</complexType>
	<complexType name="CreateBookingType">
		<complexContent>
			<extension base="mut:SiteIDType">
				<sequence>
					<element name="user" type="mut:UserIDType" />
					<element name="booking" type="mut:BookingType" />
					<element minOccurs="0" name="code" type="base64Binary" />
				</sequence>
			</extension>
		</complexContent>
	</complexType>
	<complexType name="BookingType">
		<sequence>
			<element name="permission" type="mut:PermissionIDType" />
			<element name="start" type="dateTime" />
			<element name="end" type="dateTime" />
		</sequence>
	</complexType>
	<complexType name="BookingResponseType">
		<complexContent>
			<extension base="mut:OperationResponseType">
				<sequence>
```

```
                                                <element minOccurs="0" name="bookingID"
type="mut:BookingIDType" />
                                                <element maxOccurs="unbounded" minOccurs="0" name="bestFits"
                                                        type="mut:TimePeriodType" />
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="BookingIDType">
                        <complexContent>
                                <extension base="mut:SiteIDType">
                                        <sequence>
                                                <element name="id" type="int" />
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="TimePeriodType">
                        <sequence>
                                <element name="start" type="dateTime" />
                                <element name="end" type="dateTime" />
                        </sequence>
                </complexType>
                <complexType name="FindFreeBookingsType">
                        <complexContent>
                                <extension base="mut:SiteIDType">
                                        <sequence>
                                                <element name="permission" type="mut:PermissionIDType" />
                                                <element name="period" type="mut:TimePeriodType" />
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="FindFreeBookingsResponseType">
                        <sequence>
                                <element name="permission" type="mut:PermissionIDType" />
                                <element name="resource" type="mut:ResourceType" />
                                <element maxOccurs="unbounded" minOccurs="0" name="slot"
                                        type="mut:BookingSlotType" />
                        </sequence>
                </complexType>
                <complexType name="BookingSlotType">
                        <complexContent>
                                <extension base="mut:TimePeriodType">
                                        <sequence>
                                                <element name="state">
                                                        <simpleType>
                                                                <restriction base="string">
                                                                        <enumeration value="FREE" />
                                                                        <enumeration value="BOOKED" />
                                                                        <enumeration value="NOPERMISSION" />
                                                                </restriction>
                                                        </simpleType>
                                                </element>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
        </schema>

  </wsdl:types>
  <wsdl:message name="findFreeBookingsResponse">
    <wsdl:part name="parameters" element="mut:findFreeBookingsResponse">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="addToQueueRequest">
    <wsdl:part name="parameters" element="mut:addToQueue">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="getSessionInformationRequest">
    <wsdl:part name="parameters" element="mut:getSessionInformation">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="checkAvailabilityResponse">
    <wsdl:part name="parameters" element="mut:checkAvailabilityResponse">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="getUserStatusRequest">
    <wsdl:part name="parameters" element="mut:getUserStatus">
```

```
  </wsdl:part>
</wsdl:message>
<wsdl:message name="createBookingRequest">
  <wsdl:part name="parameters" element="mut:createBooking">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="getSessionInformationResponse">
  <wsdl:part name="parameters" element="mut:getSessionInformationResponse">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="getQueuePositionRequest">
  <wsdl:part name="parameters" element="mut:getQueuePosition">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="finishSessionResponse">
  <wsdl:part name="parameters" element="mut:finishSessionResponse">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="checkAvailabilityRequest">
  <wsdl:part name="parameters" element="mut:checkAvailability">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="getUserStatusResponse">
  <wsdl:part name="parameters" element="mut:getUserStatusResponse">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="cancelBookingRequest">
  <wsdl:part name="parameters" element="mut:cancelBooking">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="getQueuePositionResponse">
  <wsdl:part name="parameters" element="mut:getQueuePositionResponse">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="findFreeBookingsRequest">
  <wsdl:part name="parameters" element="mut:findFreeBookings">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="finishSessionRequest">
  <wsdl:part name="parameters" element="mut:finishSession">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="cancelBookingResponse">
  <wsdl:part name="parameters" element="mut:cancelBookingResponse">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="createBookingResponse">
  <wsdl:part name="parameters" element="mut:createBookingResponse">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="addToQueueResponse">
  <wsdl:part name="parameters" element="mut:addToQueueResponse">
  </wsdl:part>
</wsdl:message>
<wsdl:portType name="MultiSite">
  <wsdl:operation name="checkAvailability">
    <wsdl:input message="mut:checkAvailabilityRequest">
  </wsdl:input>
    <wsdl:output message="mut:checkAvailabilityResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="addToQueue">
    <wsdl:input message="mut:addToQueueRequest">
  </wsdl:input>
    <wsdl:output message="mut:addToQueueResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getUserStatus">
    <wsdl:input message="mut:getUserStatusRequest">
  </wsdl:input>
    <wsdl:output message="mut:getUserStatusResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getQueuePosition">
    <wsdl:input message="mut:getQueuePositionRequest">
  </wsdl:input>
    <wsdl:output message="mut:getQueuePositionResponse">
  </wsdl:output>
  </wsdl:operation>
```

```
  <wsdl:operation name="getSessionInformation">
    <wsdl:input message="mut:getSessionInformationRequest">
  </wsdl:input>
    <wsdl:output message="mut:getSessionInformationResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="finishSession">
    <wsdl:input message="mut:finishSessionRequest">
  </wsdl:input>
    <wsdl:output message="mut:finishSessionResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="createBooking">
    <wsdl:input message="mut:createBookingRequest">
  </wsdl:input>
    <wsdl:output message="mut:createBookingResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="cancelBooking">
    <wsdl:input message="mut:cancelBookingRequest">
  </wsdl:input>
    <wsdl:output message="mut:cancelBookingResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="findFreeBookings">
    <wsdl:input message="mut:findFreeBookingsRequest">
  </wsdl:input>
    <wsdl:output message="mut:findFreeBookingsResponse">
  </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="MultiSiteSOAP" type="mut:MultiSite">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="checkAvailability">
    <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/NewOperation"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="addToQueue">
    <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/addToQueue"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getUserStatus">
    <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/getUserStatus"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getQueuePosition">
    <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/getQueuePosition"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getSessionInformation">
    <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/getSessionInformation"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="finishSession">
```

```
      <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/finishSession"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="createBooking">
      <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/createBooking"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="cancelBooking">
      <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/cancelBooking"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="findFreeBookings">
      <soap:operation soapAction="http://remotelabs.eng.uts.edu.au/schedserver/multisite/findFreeBookings"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="MultiSite">
    <wsdl:port name="MultiSiteSOAP" binding="mut:MultiSiteSOAP">
      <soap:address location="http://remotelabs.eng.uts.edu.au:8080/SchedulingServer-MultiSite/services/MultiSite"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

MULTISITE CALLBACK SERVICE

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions
        xmlns:mutcb="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteCallback/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="MultiSiteCallback"
        targetNamespace="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteCallback/">
        <wsdl:types>
        <!-- Common MultiSite schema. -->
                <schema xmlns="http://www.w3.org/2001/XMLSchema"
                        xmlns:mut="http://remotelabs.eng.uts.edu.au/schedserver/multisite"
                        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
                        elementFormDefault="unqualified"
                        targetNamespace="http://remotelabs.eng.uts.edu.au/schedserver/multisite">
                        <element name="checkAvailability" type="mut:PermissionIDType" />
                        <element name="checkAvailabilityResponse" type="mut:AvailabilityResponseType" />
                        <element name="addToQueue" type="mut:QueueRequestType" />
                        <element name="addToQueueResponse" type="mut:UserStatusType" />
                        <element name="getUserStatus" type="mut:UserIDType" />
                        <element name="getUserStatusResponse" type="mut:UserStatusType" />
                        <element name="getQueuePosition" type="mut:UserIDType" />
                        <element name="getQueuePositionResponse" type="mut:QueueType" />
                        <element name="getSessionInformation" type="mut:UserIDType" />
                        <element name="getSessionInformationResponse" type="mut:SessionType" />
                        <element name="finishSession" type="mut:UserIDType" />
                        <element name="finishSessionResponse" type="mut:OperationResponseType" />
                        <element name="createBooking" type="mut:CreateBookingType" />
                        <element name="createBookingResponse" type="mut:BookingResponseType" />
                        <element name="cancelBooking" type="mut:BookingIDType" />
                        <element name="cancelBookingResponse" type="mut:OperationResponseType" />
                        <element name="findFreeBookings" type="mut:FindFreeBookingsType" />
                        <element name="findFreeBookingsResponse" type="mut:FindFreeBookingsResponseType" />
                        <complexType name="PermissionIDType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="permissionID" type="string" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="SiteIDType">
                                <sequence>
                                        <element name="siteID" type="string" />
                                </sequence>
                        </complexType>
                        <complexType name="QueueRequestType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="permission" type="mut:PermissionIDType" />
                                                        <element name="user" type="mut:UserIDType" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="UserIDType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="userID" type="string" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="UserStatusType">
                                <sequence>
                                        <element minOccurs="0" name="operation" type="mut:OperationResponseType" />
                                        <element name="inQueue" type="boolean" />
                                        <element name="inSession" type="boolean" />
                                        <element name="inBooking" type="boolean" />
                                        <element minOccurs="0" name="queuedResource" type="mut:ResourceType" />
                                        <element minOccurs="0" name="bookedResource" type="mut:ResourceType" />
                                        <element minOccurs="0" name="session" type="mut:SessionType" />
                                </sequence>
                        </complexType>
```

```
	<complexType name="OperationResponseType">
		<sequence>
			<element name="wasSuccessful" type="boolean" />
			<element minOccurs="0" name="reason" type="string" />
		</sequence>
	</complexType>
	<complexType name="AvailabilityResponseType">
		<sequence>
			<element name="viable" type="boolean" />
			<element name="hasFree" type="boolean" />
			<element name="isQueueable" type="boolean" />
			<element name="isBookable" type="boolean" />
			<element name="isCodeAssignable" type="boolean" />
			<element name="queuedResource" type="mut:ResourceType" />
			<element maxOccurs="unbounded" minOccurs="0" name="queueTarget"
				type="mut:QueueTargetType" />
		</sequence>
	</complexType>
	<complexType name="QueueTargetType">
		<sequence>
			<element name="resource" type="mut:ResourceType" />
			<element name="viable" type="boolean" />
			<element name="isFree" type="boolean" />
		</sequence>
	</complexType>
	<complexType name="ResourceType">
		<sequence>
			<element name="type" type="string" />
			<element name="name" type="string" />
		</sequence>
	</complexType>
	<complexType name="SessionType">
		<sequence>
			<element name="isReady" type="boolean" />
			<element name="isCodeAssigned" type="boolean" />
			<element minOccurs="0" name="resource" type="mut:ResourceType" />
			<element minOccurs="0" name="rigType" type="string" />
			<element minOccurs="0" name="rigName" type="string" />
			<element minOccurs="0" name="contactURL" type="string" />
			<element name="duration" type="int"/>
<element name="extensions" type="int" />
<element name="timeLeft" type="int"/>
			<element name="inGrace" type="boolean" />
			<element minOccurs="0" name="warningMessage" type="string" />
		</sequence>
	</complexType>
	<complexType name="QueueType">
		<sequence>
			<element name="inQueue" type="boolean" />
			<element name="inSession" type="boolean" />
			<element name="inBooking" type="boolean" />
			<element minOccurs="0" name="queuedResource" type="mut:ResourceType" />
			<element name="position" type="int" />
			<element name="time" type="int" />
		</sequence>
	</complexType>
	<complexType name="CreateBookingType">
		<complexContent>
			<extension base="mut:SiteIDType">
				<sequence>
					<element name="user" type="mut:UserIDType" />
					<element name="booking" type="mut:BookingType" />
					<element minOccurs="0" name="code" type="base64Binary" />
				</sequence>
			</extension>
		</complexContent>
	</complexType>
	<complexType name="BookingType">
		<sequence>
			<element name="permission" type="mut:PermissionIDType" />
			<element name="start" type="dateTime" />
			<element name="end" type="dateTime" />
		</sequence>
	</complexType>
	<complexType name="BookingResponseType">
		<complexContent>
			<extension base="mut:OperationResponseType">
				<sequence>
```

```
                                        <element minOccurs="0" name="bookingID"
type="mut:BookingIDType" />
                                        <element maxOccurs="unbounded" minOccurs="0" name="bestFits"
                                                type="mut:TimePeriodType" />
                                </sequence>
                        </extension>
                </complexContent>
        </complexType>
        <complexType name="BookingIDType">
                <complexContent>
                        <extension base="mut:SiteIDType">
                                <sequence>
                                        <element name="id" type="int" />
                                </sequence>
                        </extension>
                </complexContent>
        </complexType>
        <complexType name="TimePeriodType">
                <sequence>
                        <element name="start" type="dateTime" />
                        <element name="end" type="dateTime" />
                </sequence>
        </complexType>
        <complexType name="FindFreeBookingsType">
                <complexContent>
                        <extension base="mut:SiteIDType">
                                <sequence>
                                        <element name="permission" type="mut:PermissionIDType" />
                                        <element name="period" type="mut:TimePeriodType" />
                                </sequence>
                        </extension>
                </complexContent>
        </complexType>
        <complexType name="FindFreeBookingsResponseType">
                <sequence>
                        <element name="permission" type="mut:PermissionIDType" />
                        <element name="resource" type="mut:ResourceType" />
                        <element maxOccurs="unbounded" minOccurs="0" name="slot"
                                type="mut:BookingSlotType" />
                </sequence>
        </complexType>
        <complexType name="BookingSlotType">
                <complexContent>
                        <extension base="mut:TimePeriodType">
                                <sequence>
                                        <element name="state">
                                                <simpleType>
                                                        <restriction base="string">
                                                                <enumeration value="FREE" />
                                                                <enumeration value="BOOKED" />
                                                                <enumeration value="NOPERMISSION" />
                                                        </restriction>
                                                </simpleType>
                                        </element>
                                </sequence>
                        </extension>
                </complexContent>
        </complexType>
</schema>

<xsd:schema
        targetNamespace="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteCallback/"
        xmlns:Q1="http://remotelabs.eng.uts.edu.au/schedserver/multisite">
        <xsd:import namespace="http://remotelabs.eng.uts.edu.au/schedserver/multisite" />
        <xsd:element name="bookingCancelled" type="mutcb:BookingCancelType">
        </xsd:element>
        <xsd:element name="bookingCancelledResponse" type="Q1:OperationResponseType">
        </xsd:element>
        <xsd:element name="sessionStarted" type="mutcb:UserSessionType">
        </xsd:element>
        <xsd:element name="sessionStartedResponse" type="Q1:OperationResponseType">
        </xsd:element>
        <xsd:element name="sessionFinished" type="Q1:UserIDType">
        </xsd:element>
        <xsd:element name="sessionFinishedResponse"
                type="Q1:OperationResponseType">
        </xsd:element>
        <xsd:element name="sessionUpdate" type="mutcb:UserSessionType">
        </xsd:element>
```

```
                <xsd:element name="sessionUpdateResponse" type="Q1:OperationResponseType">
                </xsd:element>

                <xsd:complexType name="BookingCancelType">
                        <xsd:complexContent>
                                <xsd:extension base="Q1:SiteIDType">
                                        <xsd:sequence>
                                                <xsd:element name="user" type="Q1:UserIDType">
                                                </xsd:element>
                                                <xsd:element name="booking" type="Q1:BookingType">
                                                </xsd:element>
                                        </xsd:sequence>
                                </xsd:extension>
                        </xsd:complexContent>
                </xsd:complexType>

                <xsd:complexType name="UserSessionType">
                        <xsd:complexContent>
                                <xsd:extension base="Q1:UserIDType">
                                        <xsd:sequence>
                                                <xsd:element name="permission"
                                                        type="Q1:PermissionIDType">
                                                </xsd:element>
                                                <xsd:element name="session"
                                                        type="Q1:SessionType">
                                                </xsd:element>
                                        </xsd:sequence>
                                </xsd:extension>
                        </xsd:complexContent>
                </xsd:complexType>
        </xsd:schema>
</wsdl:types>
<wsdl:message name="bookingCancelledRequest">
        <wsdl:part element="mutcb:bookingCancelled" name="parameters" />
</wsdl:message>
<wsdl:message name="bookingCancelledResponse">
        <wsdl:part element="mutcb:bookingCancelledResponse" name="parameters" />
</wsdl:message>
<wsdl:message name="sessionStartedRequest">
        <wsdl:part name="parameters" element="mutcb:sessionStarted"></wsdl:part>
</wsdl:message>
<wsdl:message name="sessionStartedResponse">
        <wsdl:part name="parameters" element="mutcb:sessionStartedResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="sessionFinishedRequest">
        <wsdl:part name="parameters" element="mutcb:sessionFinished"></wsdl:part>
</wsdl:message>
<wsdl:message name="sessionFinishedResponse">
        <wsdl:part name="parameters" element="mutcb:sessionFinishedResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="sessionUpdateRequest">
        <wsdl:part name="parameters" element="mutcb:sessionUpdate"></wsdl:part>
</wsdl:message>
<wsdl:message name="sessionUpdateResponse">
        <wsdl:part name="parameters" element="mutcb:sessionUpdateResponse"></wsdl:part>
</wsdl:message>
<wsdl:portType name="MultiSiteCallback">
        <wsdl:operation name="bookingCancelled">
                <wsdl:input message="mutcb:bookingCancelledRequest" />
                <wsdl:output message="mutcb:bookingCancelledResponse" />
        </wsdl:operation>
        <wsdl:operation name="sessionStarted">
                <wsdl:input message="mutcb:sessionStartedRequest"></wsdl:input>
                <wsdl:output message="mutcb:sessionStartedResponse"></wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="sessionFinished">
                <wsdl:input message="mutcb:sessionFinishedRequest"></wsdl:input>
                <wsdl:output message="mutcb:sessionFinishedResponse"></wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="sessionUpdate">
                <wsdl:input message="mutcb:sessionUpdateRequest"></wsdl:input>
                <wsdl:output message="mutcb:sessionUpdateResponse"></wsdl:output>
        </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="MultiSiteCallbackSOAP" type="mutcb:MultiSiteCallback">
        <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="bookingCancelled">
                <soap:operation
```

```
        soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteCallback/bookingCancelled" />
                        <wsdl:input>
                                <soap:body use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                                <soap:body use="literal" />
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="sessionStarted">
                        <soap:operation

        soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteCallback/sessionStarted" />
                        <wsdl:input>
                                <soap:body use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                                <soap:body use="literal" />
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="sessionFinished">
                        <soap:operation

        soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteCallback/sessionFinished" />
                        <wsdl:input>
                                <soap:body use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                                <soap:body use="literal" />
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="sessionUpdate">
                        <soap:operation

        soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteCallback/sessionUpdate" />
                        <wsdl:input>
                                <soap:body use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                                <soap:body use="literal" />
                        </wsdl:output>
                </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="MultiSiteCallback">
                <wsdl:port binding="mutcb:MultiSiteCallbackSOAP" name="MultiSiteCallbackSOAP">
                        <soap:address
                                location="http://remotelabs.eng.uts.edu.au:8080/SchedulingServer-
MultiSite/services/MultiSiteCallback" />
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        name="MultiSiteFederation"
        targetNamespace="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/"
        xmlns:p="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:mutfd="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/">
        <wsdl:types>
           <!-- Common MultiSite schema. -->
                <schema xmlns="http://www.w3.org/2001/XMLSchema"
                        xmlns:mut="http://remotelabs.eng.uts.edu.au/schedserver/multisite"
                        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
                        elementFormDefault="unqualified"
                        targetNamespace="http://remotelabs.eng.uts.edu.au/schedserver/multisite">
                        <element name="checkAvailability" type="mut:PermissionIDType" />
                        <element name="checkAvailabilityResponse" type="mut:AvailabilityResponseType" />
                        <element name="addToQueue" type="mut:QueueRequestType" />
                        <element name="addToQueueResponse" type="mut:UserStatusType" />
                        <element name="getUserStatus" type="mut:UserIDType" />
                        <element name="getUserStatusResponse" type="mut:UserStatusType" />
                        <element name="getQueuePosition" type="mut:UserIDType" />
                        <element name="getQueuePositionResponse" type="mut:QueueType" />
                        <element name="getSessionInformation" type="mut:UserIDType" />
                        <element name="getSessionInformationResponse" type="mut:SessionType" />
                        <element name="finishSession" type="mut:UserIDType" />
                        <element name="finishSessionResponse" type="mut:OperationResponseType" />
                        <element name="createBooking" type="mut:CreateBookingType" />
                        <element name="createBookingResponse" type="mut:BookingResponseType" />
                        <element name="cancelBooking" type="mut:BookingIDType" />
                        <element name="cancelBookingResponse" type="mut:OperationResponseType" />
                        <element name="findFreeBookings" type="mut:FindFreeBookingsType" />
                        <element name="findFreeBookingsResponse" type="mut:FindFreeBookingsResponseType" />
                        <complexType name="PermissionIDType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="permissionID" type="string" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="SiteIDType">
                                <sequence>
                                        <element name="siteID" type="string" />
                                </sequence>
                        </complexType>
                        <complexType name="QueueRequestType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="permission" type="mut:PermissionIDType" />
                                                        <element name="user" type="mut:UserIDType" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="UserIDType">
                                <complexContent>
                                        <extension base="mut:SiteIDType">
                                                <sequence>
                                                        <element name="userID" type="string" />
                                                </sequence>
                                        </extension>
                                </complexContent>
                        </complexType>
                        <complexType name="UserStatusType">
                                <sequence>
                                        <element minOccurs="0" name="operation" type="mut:OperationResponseType" />
                                        <element name="inQueue" type="boolean" />
                                        <element name="inSession" type="boolean" />
                                        <element name="inBooking" type="boolean" />
                                        <element minOccurs="0" name="queuedResource" type="mut:ResourceType" />
                                        <element minOccurs="0" name="bookedResource" type="mut:ResourceType" />
                                        <element minOccurs="0" name="session" type="mut:SessionType" />
                                </sequence>
```

```
	</complexType>
	<complexType name="OperationResponseType">
		<sequence>
			<element name="wasSuccessful" type="boolean" />
			<element minOccurs="0" name="reason" type="string" />
		</sequence>
	</complexType>
	<complexType name="AvailabilityResponseType">
		<sequence>
			<element name="viable" type="boolean" />
			<element name="hasFree" type="boolean" />
			<element name="isQueueable" type="boolean" />
			<element name="isBookable" type="boolean" />
			<element name="isCodeAssignable" type="boolean" />
			<element name="queuedResource" type="mut:ResourceType" />
			<element maxOccurs="unbounded" minOccurs="0" name="queueTarget"
				type="mut:QueueTargetType" />
		</sequence>
	</complexType>
	<complexType name="QueueTargetType">
		<sequence>
			<element name="resource" type="mut:ResourceType" />
			<element name="viable" type="boolean" />
			<element name="isFree" type="boolean" />
		</sequence>
	</complexType>
	<complexType name="ResourceType">
		<sequence>
			<element name="type" type="string" />
			<element name="name" type="string" />
		</sequence>
	</complexType>
	<complexType name="SessionType">
		<sequence>
			<element name="isReady" type="boolean" />
			<element name="isCodeAssigned" type="boolean" />
			<element minOccurs="0" name="resource" type="mut:ResourceType" />
			<element minOccurs="0" name="rigType" type="string" />
			<element minOccurs="0" name="rigName" type="string" />
			<element minOccurs="0" name="contactURL" type="string" />
			<element name="duration" type="int"/>
<element name="extensions" type="int" />
<element name="timeLeft" type="int"/>
			<element name="inGrace" type="boolean" />
			<element minOccurs="0" name="warningMessage" type="string" />
		</sequence>
	</complexType>
	<complexType name="QueueType">
		<sequence>
			<element name="inQueue" type="boolean" />
			<element name="inSession" type="boolean" />
			<element name="inBooking" type="boolean" />
			<element minOccurs="0" name="queuedResource" type="mut:ResourceType" />
			<element name="position" type="int" />
			<element name="time" type="int" />
		</sequence>
	</complexType>
	<complexType name="CreateBookingType">
		<complexContent>
			<extension base="mut:SiteIDType">
				<sequence>
					<element name="user" type="mut:UserIDType" />
					<element name="booking" type="mut:BookingType" />
					<element minOccurs="0" name="code" type="base64Binary" />
				</sequence>
			</extension>
		</complexContent>
	</complexType>
	<complexType name="BookingType">
		<sequence>
			<element name="permission" type="mut:PermissionIDType" />
			<element name="start" type="dateTime" />
			<element name="end" type="dateTime" />
		</sequence>
	</complexType>
	<complexType name="BookingResponseType">
		<complexContent>
			<extension base="mut:OperationResponseType">
				<sequence>
```

```
                                                <element minOccurs="0" name="bookingID"
type="mut:BookingIDType" />
                                                <element maxOccurs="unbounded" minOccurs="0" name="bestFits"
                                                        type="mut:TimePeriodType" />
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="BookingIDType">
                        <complexContent>
                                <extension base="mut:SiteIDType">
                                        <sequence>
                                                <element name="id" type="int" />
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="TimePeriodType">
                        <sequence>
                                <element name="start" type="dateTime" />
                                <element name="end" type="dateTime" />
                        </sequence>
                </complexType>
                <complexType name="FindFreeBookingsType">
                        <complexContent>
                                <extension base="mut:SiteIDType">
                                        <sequence>
                                                <element name="permission" type="mut:PermissionIDType" />
                                                <element name="period" type="mut:TimePeriodType" />
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
                <complexType name="FindFreeBookingsResponseType">
                        <sequence>
                                <element name="permission" type="mut:PermissionIDType" />
                                <element name="resource" type="mut:ResourceType" />
                                <element maxOccurs="unbounded" minOccurs="0" name="slot"
                                        type="mut:BookingSlotType" />
                        </sequence>
                </complexType>
                <complexType name="BookingSlotType">
                        <complexContent>
                                <extension base="mut:TimePeriodType">
                                        <sequence>
                                                <element name="state">
                                                        <simpleType>
                                                                <restriction base="string">
                                                                        <enumeration value="FREE" />
                                                                        <enumeration value="BOOKED" />
                                                                        <enumeration value="NOPERMISSION" />
                                                                </restriction>
                                                        </simpleType>
                                                </element>
                                        </sequence>
                                </extension>
                        </complexContent>
                </complexType>
        </schema>


        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/"
xmlns:mut="http://remotelabs.eng.uts.edu.au/schedserver/multisite">
                <xsd:import namespace="http://remotelabs.eng.uts.edu.au/schedserver/multisite" />
                <xsd:element name="siteReconnect" type="mutfd:SiteType">
                </xsd:element>
                <xsd:element name="siteReconnectResponse"
                        type="mut:OperationResponseType">
                </xsd:element>
                <xsd:element name="siteStatus" type="mut:SiteIDType">
                </xsd:element>
                <xsd:element name="siteStatusResponse"
                        type="mut:OperationResponseType">
                </xsd:element>
                <xsd:element name="siteShutdown"
                        type="mutfd:SiteShutdownType">
                </xsd:element>
                <xsd:element name="siteShutdownResponse"
```

```
                type="mut:OperationResponseType">
        </xsd:element>
        <xsd:element name="initiateSite" type="mutfd:SiteType">
        </xsd:element>
        <xsd:element name="initiateSiteResponse"
                type="mutfd:InitiateSiteResponseType">
        </xsd:element>
        <xsd:element name="discoverResources"
                type="mut:SiteIDType">
        </xsd:element>
        <xsd:element name="discoverResourcesResponse"
                type="mutfd:RequestableResourceListType">
        </xsd:element>
        <xsd:element name="requestResource"
                type="mutfd:ResourceRequestType">
        </xsd:element>
        <xsd:element name="requestResourceResponse"
                type="mutfd:RequestResourceResponseType">
        </xsd:element>
        <xsd:element name="notifyAccept"
                type="mut:PermissionIDType">
        </xsd:element>
        <xsd:element name="notifyAcceptResponse"
                type="mut:OperationResponseType">
        </xsd:element>
        <xsd:element name="notifyModify"
                type="mutfd:ResourceRequestType">
        </xsd:element>
        <xsd:element name="notifyModifyResponse"
                type="mut:OperationResponseType">
        </xsd:element>
        <xsd:element name="notifyCancel"
                type="mut:PermissionIDType">
        </xsd:element>
        <xsd:element name="notifyCancelResponse"
                type="mut:OperationResponseType">
        </xsd:element>
        <xsd:element name="getRequests" type="mut:SiteIDType">
        </xsd:element>
        <xsd:element name="getRequestsResponse"
                type="mutfd:ResourceRequestListType">
        </xsd:element>

        <xsd:complexType name="SiteType">
                <xsd:complexContent>
                        <xsd:extension base="mut:SiteIDType">
                                <xsd:sequence>
                                        <xsd:element name="name"
                                                type="xsd:string">
                                        </xsd:element>
            <xsd:element name="namespace" type="xsd:string">
                                        </xsd:element>
            <xsd:element name="address"
                                                type="xsd:string">
                                        </xsd:element>
                                </xsd:sequence>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>

        <xsd:complexType name="SiteShutdownType">
                <xsd:complexContent>
                        <xsd:extension base="mut:SiteIDType">
                                <xsd:sequence>
                                        <xsd:element name="reason"
                                                type="xsd:string">
                                        </xsd:element>
                                </xsd:sequence>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>

        <xsd:complexType name="RequestableResourceListType">
                <xsd:sequence>
                        <xsd:element name="resources"
                                type="mutfd:RequestableResourceType" maxOccurs="unbounded"
minOccurs="0">
                        </xsd:element>
                </xsd:sequence>
```

```
</xsd:complexType>

<xsd:complexType name="RequestableResourceType">
        <xsd:sequence>
                <xsd:element name="resource"
                        type="mut:ResourceType">
                </xsd:element>
                <xsd:element name="start" type="xsd:dateTime"></xsd:element>
                <xsd:element name="end" type="xsd:dateTime"></xsd:element>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="RequestResourceResponseType">
        <xsd:complexContent>
                <xsd:extension base="mut:OperationResponseType">
                        <xsd:sequence>
   <xsd:element name="permissionID" type="xsd:string" maxOccurs="1" minOccurs="0">
                                </xsd:element>
   <xsd:element name="requiresRedirect"
                                        type="xsd:boolean" maxOccurs="1" minOccurs="0">
                                </xsd:element>
                                <xsd:element name="site"
                                        type="mutfd:SiteType" maxOccurs="1" minOccurs="0">
                                </xsd:element>
                        </xsd:sequence>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ResourceRequestType">
        <xsd:complexContent>
                <xsd:extension base="mut:SiteIDType">
                        <xsd:sequence>
                                <xsd:element name="permissionID"
                                        type="xsd:string" maxOccurs="1" minOccurs="0">
                                </xsd:element>
                                <xsd:element name="resource"
                                        type="mut:ResourceType">
                                </xsd:element>
                                <xsd:element name="start"
                                        type="xsd:dateTime">
                                </xsd:element>
                                <xsd:element name="expiry"
                                        type="xsd:dateTime">
                                </xsd:element>
                                <xsd:element name="name"
                                        type="xsd:string">
                                </xsd:element>
                                <xsd:element name="sessionDuration"
                                        type="xsd:int">
                                </xsd:element>
                                <xsd:element name="allowedExtensions"
                                        type="xsd:int">
                                </xsd:element>
                                <xsd:element name="extensionDuration"
                                        type="xsd:int">
                                </xsd:element>
                                <xsd:element name="isQueuable"
                                        type="xsd:boolean">
                                </xsd:element>
                                <xsd:element name="isBookable"
                                        type="xsd:boolean">
                                </xsd:element>
                                <xsd:element name="maxBookings"
                                        type="xsd:string" maxOccurs="1" minOccurs="0">
                                </xsd:element>
                                <xsd:element name="note" type="xsd:string"
                                        maxOccurs="1" minOccurs="0">
                                </xsd:element>
                        </xsd:sequence>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ResourceRequestListType">
        <xsd:sequence>
                <xsd:element name="list"
                        type="mutfd:ResourceRequestType" maxOccurs="unbounded" minOccurs="0">
                </xsd:element>
```

```
                        </xsd:sequence>
                </xsd:complexType>

                <xsd:complexType name="InitiateSiteResponseType">
                        <xsd:complexContent>
                                <xsd:extension base="mut:OperationResponseType">
                                        <xsd:sequence>
                                                <xsd:element name="site"
                                                        type="mutfd:SiteType" maxOccurs="1" minOccurs="0">
                                                </xsd:element>
                                        </xsd:sequence>
                                </xsd:extension>
                        </xsd:complexContent>
                </xsd:complexType>
        </xsd:schema>
</wsdl:types>
<wsdl:message name="siteReconnectRequest">
        <wsdl:part name="parameters" element="mutfd:siteReconnect" />
</wsdl:message>
<wsdl:message name="siteReconnectResponse">
        <wsdl:part name="parameters" element="mutfd:siteReconnectResponse" />
</wsdl:message>
<wsdl:message name="siteStatusRequest">
        <wsdl:part name="parameters" element="mutfd:siteStatus" />
</wsdl:message>
<wsdl:message name="siteStatusResponse">
        <wsdl:part name="parameters" element="mutfd:siteStatusResponse" />
</wsdl:message>
<wsdl:message name="siteShutdownRequest">
        <wsdl:part name="parameters" element="mutfd:siteShutdown" />
</wsdl:message>
<wsdl:message name="siteShutdownResponse">
        <wsdl:part name="parameters" element="mutfd:siteShutdownResponse" />
</wsdl:message>
<wsdl:message name="initiateSiteRequest">
        <wsdl:part name="parameters" element="mutfd:initiateSite" />
</wsdl:message>
<wsdl:message name="initiateSiteResponse">
        <wsdl:part name="parameters" element="mutfd:initiateSiteResponse" />
</wsdl:message>
<wsdl:message name="discoverResourcesRequest">
        <wsdl:part name="parameters" element="mutfd:discoverResources" />
</wsdl:message>
<wsdl:message name="discoverResourcesResponse">
        <wsdl:part name="parameters" element="mutfd:discoverResourcesResponse" />
</wsdl:message>
<wsdl:message name="requestResourceRequest">
        <wsdl:part name="parameters" element="mutfd:requestResource" />
</wsdl:message>
<wsdl:message name="requestResourceResponse">
        <wsdl:part name="parameters" element="mutfd:requestResourceResponse" />
</wsdl:message>
<wsdl:message name="notifyAcceptRequest">
        <wsdl:part name="parameters" element="mutfd:notifyAccept" />
</wsdl:message>
<wsdl:message name="notifyAcceptResponse">
        <wsdl:part name="parameters" element="mutfd:notifyAcceptResponse" />
</wsdl:message>
<wsdl:message name="notifyModifyRequest">
        <wsdl:part name="parameters" element="mutfd:notifyModify" />
</wsdl:message>
<wsdl:message name="notifyModifyResponse">
        <wsdl:part name="parameters" element="mutfd:notifyModifyResponse" />
</wsdl:message>
<wsdl:message name="notifyCancelRequest">
        <wsdl:part name="parameters" element="mutfd:notifyCancel" />
</wsdl:message>
<wsdl:message name="notifyCancelResponse">
        <wsdl:part name="parameters" element="mutfd:notifyCancelResponse" />
</wsdl:message>
<wsdl:message name="getRequestsRequest">
        <wsdl:part name="parameters" element="mutfd:getRequests" />
</wsdl:message>
<wsdl:message name="getRequestsResponse">
        <wsdl:part name="parameters" element="mutfd:getRequestsResponse" />
</wsdl:message>
<wsdl:portType name="MultiSiteFederation">
        <wsdl:operation name="siteReconnect">
                <wsdl:input message="mutfd:siteReconnectRequest" />
```

```
                <wsdl:output message="mutfd:siteReconnectResponse" />
        </wsdl:operation>
        <wsdl:operation name="siteStatus">
                <wsdl:input message="mutfd:siteStatusRequest" />
                <wsdl:output message="mutfd:siteStatusResponse" />
        </wsdl:operation>
        <wsdl:operation name="siteShutdown">
                <wsdl:input message="mutfd:siteShutdownRequest" />
                <wsdl:output message="mutfd:siteShutdownResponse" />
        </wsdl:operation>
        <wsdl:operation name="initiateSite">
                <wsdl:input message="mutfd:initiateSiteRequest" />
                <wsdl:output message="mutfd:initiateSiteResponse" />
        </wsdl:operation>
        <wsdl:operation name="discoverResources">
                <wsdl:input message="mutfd:discoverResourcesRequest" />
                <wsdl:output message="mutfd:discoverResourcesResponse" />
        </wsdl:operation>
        <wsdl:operation name="requestResource">
                <wsdl:input message="mutfd:requestResourceRequest" />
                <wsdl:output message="mutfd:requestResourceResponse" />
        </wsdl:operation>
        <wsdl:operation name="notifyAccept">
                <wsdl:input message="mutfd:notifyAcceptRequest" />
                <wsdl:output message="mutfd:notifyAcceptResponse" />
        </wsdl:operation>
        <wsdl:operation name="notifyModify">
                <wsdl:input message="mutfd:notifyModifyRequest" />
                <wsdl:output message="mutfd:notifyModifyResponse" />
        </wsdl:operation>
        <wsdl:operation name="notifyCancel">
                <wsdl:input message="mutfd:notifyCancelRequest" />
                <wsdl:output message="mutfd:notifyCancelResponse" />
        </wsdl:operation>
        <wsdl:operation name="getRequests">
                <wsdl:input message="mutfd:getRequestsRequest" />
                <wsdl:output message="mutfd:getRequestsResponse" />
        </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="MultiSiteFederationSOAP"
        type="mutfd:MultiSiteFederation">
        <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="siteReconnect">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/scheduling/MultiSiteFederation/NewOperation" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="siteStatus">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/siteStatus" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="siteShutdown">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/siteShutdown" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="initiateSite">
                <soap:operation
```

```
soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/initiateSite" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="discoverResources">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/discoverResources" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="requestResource">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/requestResource" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="notifyAccept">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/notifyAccept" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="notifyModify">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/notifyModify" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="notifyCancel">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/notifyCancel" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getRequests">
                <soap:operation

soapAction="http://remotelabs.eng.uts.edu.au/schedserver/MultiSiteFederation/getRequests" />
                <wsdl:input>
                        <soap:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                        <soap:body use="literal" />
                </wsdl:output>
        </wsdl:operation>
</wsdl:binding>
<wsdl:service name="MultiSiteFederation">
        <wsdl:port binding="mutfd:MultiSiteFederationSOAP" name="MultiSiteFederationSOAP">
                <soap:address
```

```
                                location="http://remotelabs.eng.uts.edu.au:8080/SchedulingServer-
MultiSite/services/MultiSiteFederation" />
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

B. SCHEDULING SERVER DATABASE SCHEMA (MYSQL)

```
create table academic_permission (
    id integer not null auto_increment unique,
    can_control bit not null,
    can_generate_reports bit not null,
    can_kick bit not null,
    can_modify bit not null,
    can_view bit not null,
    user_id bigint not null,
    user_class_id bigint not null,
    primary key (id)
) ENGINE=InnoDB;

create table bookings (
    id bigint not null auto_increment unique,
    active bit not null,
    cancel_reason varchar(1024),
    code_reference varchar(1024),
    duration integer not null,
    end_time datetime not null,
    provider_id integer,
    resource_type varchar(10) not null,
    start_time datetime not null,
    user_name varchar(50) not null,
    user_namespace varchar(50) not null,
    request_caps_id bigint,
    resource_permission_id bigint not null,
    rig_id bigint,
    rig_type_id bigint,
    session_id bigint,
    user_id bigint not null,
    primary key (id)
) ENGINE=InnoDB;

create table config (
    id integer not null auto_increment unique,
    config_key varchar(200) not null,
    value varchar(1024) not null,
    primary key (id)
) ENGINE=InnoDB;

create table matching_capabilities (
    request_capabilities bigint not null,
    rig_capabilities bigint not null,
    primary key (request_capabilities, rig_capabilities),
    unique (rig_capabilities, request_capabilities)
) ENGINE=InnoDB;

create table remote_permission (
    id bigint not null auto_increment unique,
    active bit not null,
    guid varchar(255) not null unique,
    pending bit not null,
    rejected bit not null,
    request_time datetime not null,
    resource_permission_id bigint not null,
    remote_site_id bigint not null,
    primary key (id),
    unique (resource_permission_id)
) ENGINE=InnoDB;

create table remote_permission_log (
    id bigint not null auto_increment unique,
    event varchar(255) not null,
    reason varchar(255),
    site tinyblob,
    time datetime not null,
    remote_permission_id bigint not null,
    primary key (id)
) ENGINE=InnoDB;

create table remote_site (
    id bigint not null auto_increment unique,
    is_authorizer bit not null,
    guid varchar(255) not null unique,
    last_push datetime not null,
    name varchar(255) not null unique,
    offline_reason varchar(255),
```

```
    online bit not null,
    is_redirectee bit not null,
    is_requestor bit not null,
    service_address varchar(1024) not null,
    user_namespace varchar(255) not null unique,
    is_viewer bit not null,
    primary key (id)
) ENGINE=InnoDB;

create table request_capabilities (
    id bigint not null auto_increment unique,
    capabilities varchar(255) not null,
    primary key (id),
    unique (capabilities)
) ENGINE=InnoDB;

create table requestable_permission_period (
    id bigint not null auto_increment,
    active bit not null,
    end datetime not null,
    start datetime not null,
    type varchar(255) not null,
    request_capabilities_id bigint,
    rig_id bigint,
    rig_type_id bigint,
    primary key (id)
) ENGINE=InnoDB;

create table resource_permission (
    id bigint not null auto_increment unique,
    use_activity_detection bit not null,
    allowed_extensions smallint not null,
    display_name varchar(255),
    expiry_time datetime not null,
    extension_duration integer not null,
    maximum_bookings int default '0' not null,
    queue_activity_timeout integer not null,
    is_remote bit not null,
    session_activity_timeout integer not null,
    session_duration integer not null,
    start_time datetime not null,
    type varchar(10) not null,
    request_caps_id bigint,
    name_id bigint,
    type_id bigint,
    user_class_id bigint,
    primary key (id)
) ENGINE=InnoDB;

create table rig (
    id bigint not null auto_increment unique,
    active bit not null,
    contact_url varchar(1024),
    in_session bit not null,
    last_update_timestamp datetime not null,
    managed bit not null,
    meta varchar(255),
    name varchar(50) not null unique,
    offline_reason varchar(255),
    online bit not null,
    caps_id bigint,
    type_id bigint not null,
    session_id bigint,
    site_id bigint,
    primary key (id),
    unique (name)
) ENGINE=InnoDB;

create table rig_capabilities (
    id bigint not null auto_increment unique,
    capabilities varchar(255) not null,
    primary key (id),
    unique (capabilities)
) ENGINE=InnoDB;

create table rig_log (
    id bigint not null auto_increment unique,
    new_state varchar(20) not null,
    old_state varchar(20) not null,
```

```
    reason varchar(255) not null,
    timestamp datetime not null,
    rig_id bigint not null,
    primary key (id)
) ENGINE=InnoDB;

create table rig_offline_schedule (
    id bigint not null auto_increment unique,
    active bit not null,
    end_time datetime not null,
    reason varchar(255) not null,
    start_time datetime not null,
    rig_id bigint not null,
    primary key (id)
) ENGINE=InnoDB;

create table rig_type (
    id bigint not null auto_increment unique,
    code_assignable bit not null,
    logoff_grace_duration integer not null,
    managed bit not null,
    meta varchar(255),
    name varchar(50) not null,
    set_up_time int default '0' not null,
    tear_down_time int default '0' not null,
    site_id bigint,
    primary key (id),
    unique (name)
) ENGINE=InnoDB;

create table rig_type_information (
    id integer not null auto_increment unique,
    description varchar(2055) not null,
    institution varchar(255) not null,
    short_description varchar(2055),
    usage_description varchar(2055),
    rig_type_id bigint,
    primary key (id),
    unique (rig_type_id)
) ENGINE=InnoDB;

create table rig_type_media (
    id integer not null auto_increment unique,
    file_name varchar(255) not null,
    mime varchar(50) not null,
    rig_type_id bigint not null,
    primary key (id)
) ENGINE=InnoDB;

create table session (
    id bigint not null auto_increment unique,
    active bit not null,
    activity_last_updated datetime not null,
    assigned_rig_name varchar(50),
    assignment_time datetime,
    code_reference varchar(1024),
    duration integer not null,
    extensions smallint not null,
    in_grace bit,
    priority smallint not null,
    ready bit,
    removal_reason varchar(1024),
    removal_time datetime,
    request_time datetime not null,
    requested_resource_id bigint,
    requested_resource_name varchar(1024) not null,
    resource_type varchar(10) not null,
    user_name varchar(50) not null,
    user_namespace varchar(50) not null,
    resource_permission_id bigint,
    assigned_rig_id bigint,
    user_id bigint,
    primary key (id)
) ENGINE=InnoDB;

create table user_association (
    user_class_id bigint not null,
    users_id bigint not null,
    primary key (user_class_id, users_id)
```

```
) ENGINE=InnoDB;

create table user_class (
    id bigint not null auto_increment unique,
    active bit not null,
    bookable bit not null,
    kickable bit not null,
    name varchar(50) not null unique,
    priority smallint not null,
    queuable bit not null,
    time_horizon int default '0' not null,
    users_lockable bit not null,
    primary key (id),
    unique (name)
) ENGINE=InnoDB;

create table user_lock (
    id bigint not null auto_increment unique,
    is_locked bit not null,
    lock_key varchar(50) not null,
    resource_permission_id bigint not null,
    user_id bigint not null,
    primary key (id)
) ENGINE=InnoDB;

create table users (
    id bigint not null auto_increment unique,
    email varchar(100),
    first_name varchar(50),
    last_name varchar(50),
    name varchar(50) not null,
    namespace varchar(50) not null,
    persona varchar(8) not null,
    primary key (id),
    unique (name, namespace)
) ENGINE=InnoDB;

alter table academic_permission
    add index FK2DC5C40760957DDC (user_class_id),
    add constraint FK2DC5C40760957DDC
    foreign key (user_class_id)
    references user_class (id);

alter table academic_permission
    add index FK2DC5C407DC2CA001 (user_id),
    add constraint FK2DC5C407DC2CA001
    foreign key (user_id)
    references users (id);

alter table bookings
    add index FK7786033A4BD618F4 (rig_type_id),
    add constraint FK7786033A4BD618F4
    foreign key (rig_type_id)
    references rig_type (id);

alter table bookings
    add index FK7786033AC793D867 (request_caps_id),
    add constraint FK7786033AC793D867
    foreign key (request_caps_id)
    references request_capabilities (id);

alter table bookings
    add index FK7786033A8DD270B3 (rig_id),
    add constraint FK7786033A8DD270B3
    foreign key (rig_id)
    references rig (id);

alter table bookings
    add index FK7786033ADC2CA001 (user_id),
    add constraint FK7786033ADC2CA001
    foreign key (user_id)
    references users (id);

alter table bookings
    add index FK7786033A5C779673 (session_id),
    add constraint FK7786033A5C779673
    foreign key (session_id)
    references session (id);
```

```
alter table bookings
    add index FK7786033AAACFAD7E (resource_permission_id),
    add constraint FK7786033AAACFAD7E
    foreign key (resource_permission_id)
    references resource_permission (id);

alter table matching_capabilities
    add index FK475FF0B8FCA6AAE4 (rig_capabilities),
    add constraint FK475FF0B8FCA6AAE4
    foreign key (rig_capabilities)
    references rig_capabilities (id);

alter table matching_capabilities
    add index FK475FF0B84E80A644 (request_capabilities),
    add constraint FK475FF0B84E80A644
    foreign key (request_capabilities)
    references request_capabilities (id);

alter table remote_permission
    add index FK5A415B68E7677AE (remote_site_id),
    add constraint FK5A415B68E7677AE
    foreign key (remote_site_id)
    references remote_site (id);

alter table remote_permission
    add index FK5A415B68AACFAD7E (resource_permission_id),
    add constraint FK5A415B68AACFAD7E
    foreign key (resource_permission_id)
    references resource_permission (id);

alter table remote_permission_log
    add index FK5A0239ED75DABBAE (remote_permission_id),
    add constraint FK5A0239ED75DABBAE
    foreign key (remote_permission_id)
    references remote_permission (id);

alter table requestable_permission_period
    add index FK4966FFDB647E4772 (request_capabilities_id),
    add constraint FK4966FFDB647E4772
    foreign key (request_capabilities_id)
    references request_capabilities (id);

alter table requestable_permission_period
    add index FK4966FFDB4BD618F4 (rig_type_id),
    add constraint FK4966FFDB4BD618F4
    foreign key (rig_type_id)
    references rig_type (id);

alter table requestable_permission_period
    add index FK4966FFDB8DD270B3 (rig_id),
    add constraint FK4966FFDB8DD270B3
    foreign key (rig_id)
    references rig (id);

alter table resource_permission
    add index FK3BCCE2A060957DDC (user_class_id),
    add constraint FK3BCCE2A060957DDC
    foreign key (user_class_id)
    references user_class (id);

alter table resource_permission
    add index FK3BCCE2A0C793D867 (request_caps_id),
    add constraint FK3BCCE2A0C793D867
    foreign key (request_caps_id)
    references request_capabilities (id);

alter table resource_permission
    add index FK3BCCE2A097C6C923 (type_id),
    add constraint FK3BCCE2A097C6C923
    foreign key (type_id)
    references rig_type (id);

alter table resource_permission
    add index FK3BCCE2A02BF4F958 (name_id),
    add constraint FK3BCCE2A02BF4F958
    foreign key (name_id)
    references rig (id);

alter table rig
```

```
    add index FK1B910C571AF98 (caps_id),
    add constraint FK1B910C571AF98
    foreign key (caps_id)
    references rig_capabilities (id);

alter table rig
    add index FK1B910CD5D7EA7 (site_id),
    add constraint FK1B910CD5D7EA7
    foreign key (site_id)
    references remote_site (id);

alter table rig
    add index FK1B91097C6C923 (type_id),
    add constraint FK1B91097C6C923
    foreign key (type_id)
    references rig_type (id);

alter table rig
    add index FK1B9105C779673 (session_id),
    add constraint FK1B9105C779673
    foreign key (session_id)
    references session (id);

alter table rig_log
    add index FK478B83958DD270B3 (rig_id),
    add constraint FK478B83958DD270B3
    foreign key (rig_id)
    references rig (id);

alter table rig_offline_schedule
    add index FK1EE690C28DD270B3 (rig_id),
    add constraint FK1EE690C28DD270B3
    foreign key (rig_id)
    references rig (id);

alter table rig_type
    add index FKA9E8B909CD5D7EA7 (site_id),
    add constraint FKA9E8B909CD5D7EA7
    foreign key (site_id)
    references remote_site (id);

alter table rig_type_information
    add index FK707946F64BD618F4 (rig_type_id),
    add constraint FK707946F64BD618F4
    foreign key (rig_type_id)
    references rig_type (id);

alter table rig_type_media
    add index FK811DE12E4BD618F4 (rig_type_id),
    add constraint FK811DE12E4BD618F4
    foreign key (rig_type_id)
    references rig_type (id);

alter table session
    add index FK76508296DC2CA001 (user_id),
    add constraint FK76508296DC2CA001
    foreign key (user_id)
    references users (id);

alter table session
    add index FK76508296F147DC44 (assigned_rig_id),
    add constraint FK76508296F147DC44
    foreign key (assigned_rig_id)
    references rig (id);

alter table session
    add index FK76508296AACFAD7E (resource_permission_id),
    add constraint FK76508296AACFAD7E
    foreign key (resource_permission_id)
    references resource_permission (id);

alter table user_association
    add index FK9DC9CE0D60957DDC (user_class_id),
    add constraint FK9DC9CE0D60957DDC
    foreign key (user_class_id)
    references user_class (id);

alter table user_association
    add index FK9DC9CE0DD51D78A4 (users_id),
```

```
    add constraint FK9DC9CE0DD51D78A4
    foreign key (users_id)
    references users (id);

alter table user_lock
    add index FK1439391FDC2CA001 (user_id),
    add constraint FK1439391FDC2CA001
    foreign key (user_id)
    references users (id);

alter table user_lock
    add index FK1439391FAACFAD7E (resource_permission_id),
    add constraint FK1439391FAACFAD7E
    foreign key (resource_permission_id)
    references resource_permission (id);
```