Master’s Thesis
submitted in partial fulfilment of the
requirements for the course “Applied Computer Science”

Framework to share group information using heterogeneous group management systems in shibboleth

Sanjeev Laha

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG)

Bachelor’s and Master’s Theses of the Center for Computational Sciences at the Georg-August-Universität Göttingen

01. June 2015
Georg-August-Universität Göttingen
Institute of Computer Science
Goldschmidtstraße 7
37077 Göttingen
Germany

📞 +49 (551) 39-172000
✉️ +49 (551) 39-14403
✉️ office@informatik.uni-goettingen.de
🌐 www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Ramin Yahyapour
Second Supervisor: Dr. Sven Bingert
I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 01. June 2015
Framework to share group information using heterogeneous group management systems in shibboleth

Sanjeev Laha

01 June 2015

Supervisor
Prof. Dr. Ramin Yahyapour
Department for Practical Informatics
Georg-August-Universität Göttingen

Second supervisor
Dr. Sven Bingert
Gesellschaft für wissenschaftliche Datenverarbeitung mbH
Göttingen
Abstract

Federated identity management using shibboleth is quite popular in the field of identity management, especially among federated education and research institutes. Authentication and authorization are two important aspects of service delegation. While both authentication and authorization are set by an administrator, there is very less scope for self authorization of services by users themselves. This thesis provides an architecture and solution for implementing a system using which group administrators can authorize group members to use a particular service. The group information system will work in concurrent with shibboleth. Group information along with identity information will be shared by shibboleth identity provider with shibboleth service provider. Group information will be sourced from multiple heterogeneous group management systems held at the organization of shibboleth identity provider.
# Contents

1. **Introduction** .............................. 1
   1.1. **Motivation** .............................. 2
   1.2. **Overview** .............................. 4

2. **Fundamentals** .............................. 5
   2.1. **Shibboleth** .............................. 5
       2.1.1. Shibboleth Service Provider ......... 6
       2.1.2. Shibboleth Identity Provider ....... 6
       2.1.3. WAYF ................................ 8
       2.1.4. Flow of data and control in Shibboleth 8
   2.2. **Mailman** ................................ 9
   2.3. **UserFrosting** .......................... 11
   2.4. **LDAP** .................................. 11
   2.5. **Active Directory** ...................... 12

3. **Heterogeneous group management** .......... 14
   3.1. **Heterogeneous group sharing and advantages** 14
       3.1.1. Group sharing .......................... 14
       3.1.2. Heterogeneous group sharing ....... 16
       3.1.3. Usage at GWDG ......................... 17
   3.2. **Requirements Analysis** ................. 17
       3.2.1. Business processes .................... 18
       3.2.2. IT processes supporting business processes 19
       3.2.3. Activities within each IT process .... 19
       3.2.4. Functions within each activity ....... 19
       3.2.5. Use cases for the functions .......... 20

4. **Implementation** .............................. 22
   4.1. **Core elements and architecture** .......... 22
   4.2. **Group information providers UI** ........... 24
       4.2.1. Mailman ............................... 24
       4.2.2. UserFrosting .......................... 25
       4.2.3. Light Weight Directory Access Protocol 25
       4.2.4. Active Directory ...................... 26
   4.3. **API development** ........................ 27
       4.3.1. Basics ................................. 27
# List of Figures

1.1. Federated identity management example ........................................ 2  
1.2. Federated Identity Management with group information sharing .......... 3  

2.1. Shibboleth Architecture .............................................................. 5  
2.2. Shibboleth control flow .............................................................. 8  
2.3. Mailman Umbrella list ............................................................... 10  
2.4. LDAP directory tree ................................................................. 11  
2.5. Active directory domain forest .................................................... 12  

3.1. Benefits of sharing group information ........................................... 15  
3.2. Heterogeneous group management system ..................................... 16  
3.3. Functional Requirement Analysis ................................................... 18  
3.4. Use case for functions .................................................................. 20  

4.1. Modules used in group sharing framework ....................................... 22  
4.2. Flow chart for group management ................................................... 38  
4.3. Flow chart for access management .................................................. 40  

5.1. Shibboleth IdP login response time ................................................. 43  
5.2. Shibboleth SP attribute synchronizer response time ......................... 44  

7.1. Grouper based federated group sharing ........................................... 46  

List of Tables

4.1. Identity provider extension parts ........................................ 27
A.1. System Requirements for Shibboleth Installation ................ 53
A.2. Shibboleth IdP configuration changes .............................. 55
A.3. Shibboleth IdP configuration changes .............................. 57
### Glossary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>AD</td>
<td>Active Directory is a directory service provided by Microsoft in its operating systems.</td>
</tr>
<tr>
<td>Ajax</td>
<td>Asynchronous JavaScript and XML is a set of techniques used at client-side to create asynchronous communication with web server.</td>
</tr>
<tr>
<td>API</td>
<td>Application program interface is used for integrating new features into existing software.</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface is a standard used for generating dynamic web content and web applications</td>
</tr>
<tr>
<td>CIDR</td>
<td>Classless Inter-Domain Routing</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma separated value file is used for storing delimited data</td>
</tr>
<tr>
<td>GWDDG</td>
<td>Gesellschaft für wissenschaftliche Datenverarbeitung mbH is an IT service provider based in Goettingen for Georg-August-Universitaet Goettingen and Max Planck Society.</td>
</tr>
<tr>
<td>Handler</td>
<td>Shibboleth service provider handler is an extension to provide additional features to the service provider.</td>
</tr>
<tr>
<td>IdP</td>
<td>Identity Provider in shibboleth</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive is a package file format used for distribution of aggregated Java class files and any other dependent metadata and resources.</td>
</tr>
<tr>
<td>JSch</td>
<td>Java Secure Channel is a java implementation of SSH2 protocol which allows to connect securely java programs with SSH based servers.</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol is an internet protocol used by online application and softwares to look for information stored in a server.</td>
</tr>
<tr>
<td>SLO</td>
<td>Single Logout is an action to cleanly exit from multiple sessions in a single sign-on session.</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol is a messaging protocol which helps in communication between heterogeneous operating systems using HTTP and XML protocols.</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider in shibboleth.</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network. Source: <a href="https://tools.ietf.org/html/rfc4253">https://tools.ietf.org/html/rfc4253</a></td>
</tr>
<tr>
<td>SVN</td>
<td>Sub-Version is a popular version control system used for collaborative project development.</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface.</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform resource identifier (URI) is a character string used to identify a resource’s name.</td>
</tr>
<tr>
<td>URL</td>
<td>Universal Resource Locator.</td>
</tr>
<tr>
<td>UUID</td>
<td>Universally unique identifier is a 128 bit value often displayed as hexadecimal text with hyphen characters in middle. It is used in software systems for uniquely identifying information.</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language (rfc4825)</td>
</tr>
</tbody>
</table>
1. Introduction

Digital identity management is an important aspect for online services as it provides authentication and authorization to the users, to be able to use the services provided by organizations. With growing number of online services, users are expected to maintain different digital identification information, which means different username and password for each independent account. Majority of users use the same username and password for login with every service they are using online[1], which makes the process unsafe because a malicious service provider can misuse the stored password of the user to access other services, or an identity leak will render more number of accounts vulnerable.

Digital identity management is a very important aspect for both service provider and the service users. Federated identity management helps both users and service providers to minimize the risk and burden of maintaining redundant identities[2]. The core functionality of a federated identity management system is to share authentication information among federated organizations[3] and allow users from different organizations to access the services without the need to have dedicated user accounts. Users authenticate themselves with their home organization which in turn sends an authorization token to the service provider to allow or deny the access to a service. Authorization is an equally important aspect for federated identity management solution.

“The development of Web-based federated identity solutions has advanced more rapidly compared to Web-based privilege management mechanisms. The result is a wide gap between the federated identity and privilege management mechanisms and calls for an integrated approach to provide a comprehensive access management solution.”[4]. While most of the federated identity management solutions can offer authentication and authorization by utilizing the same back end provider like LDAP[5] or Active Directory[6] (AD), but in many cases it could be possible to have a dedicated group management system in the organization. For instance an organization might have the requirement to distinguish the systems storing authentication information and authorization information (especially for user groups). The modern federated identity management systems provide very less or no support to connect any existing group management system to a federated identity management system.
1.1. Motivation

The main goal of this research paper is to develop a solution which could be utilized to connect different available group management systems with the federated identity management system shibboleth[7], in order to be able to share group information. This section contains discussion about the usage of group information sharing in the federated organization environment and the benefits which could be leveraged through such a system.

Figure 1.1 shows an example of two federated organizations connected through shibboleth. As shibboleth is mostly used by educational institutes let’s assume organization A in fig 1.1 is the home university of user A. User A is enrolled in many courses at his home university. User A is further interested to enrol in a course offered by organization B. Normally to get the access to the course at organization B user A will be asked to make a request for a new account at organization B.

Here comes shibboleth into play by replacing the need to create new user account for user A. Instead, organization A and organization B sign a contract for existence of a federation between them. With the federation in place the login page of organization B is simultaneously updated with a link to support login via organization A. The entire users from organization A including user A can access this link which redirect them to their home organization for authentication. After successful authentication organization A sends a valid authentication token with additional attributes understandable by organization B’s web service. In the next step organization B creates an account for user A by processing the authentication token and attributes returned by organization A. This step helps the service provider identify returning users and also stores their preferences and other information related to the service offered. In this example scenario, user A can now ask the course coordinator at organization B to give him access
Chapter 1. Introduction

to the course as his account is active in organization B. Due to federation in place user A is able to access the service and the course within minutes which might have taken a couple of days to be addressed earlier.

Organization A also contains group information of the users (fig 1.1). Let’s assume user A is the administrator of a course group in his university and this information is stored in the group information storage system at organization A. Further, user A has been privileged to administer a course in organization B. User A wants to provide access of this new course to the group members of user A from organization A. To achieve this scenario, user A will request all the group members to login into organization B’s service in order to create the initial account with the help of shibboleth. In the next step user A will search and add individual users to the new course group in organization B. This process of group creation could be very tedious if there are 100’s of users who belong to a group and given the fact that the group already exist in another federated organization within the federation. Moreover, user A has to wait until a user creates his account at organization B with the help of shibboleth to add the new user to the course group.

Figure 1.2.: Federated Identity Management with group information sharing

The main goal of this research paper is to identify a solution for sharing group information using shibboleth as the mediator and many different group management systems as the source of group information. Also, the goal is to identify the key use cases in which a group information sharing might be useful specially in a federation of universities, scientific institutes and IT service providers like Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG). As described in fig 1.2 the motivation of this research paper is to connect the heterogeneous group management systems between different federated organizations which will create a link between the systems to allow exchange of group information. Users will be able to import already existing groups from their home organization to minimize redundant group creation.
and provide new users belonging to the group with instant access to shared services. Further, the solution will help in real time synchronization of current group members, providing or revoking proper access rights to a group user. There will be discussion on the best practices for single log out procedure in shibboleth’s environment to safely end multiple signed in sessions by the user created by accessing the shared service.

1.2. Overview

The following chapters will discuss about the path towards identification of key requirements for development and implementation of the solution for group information sharing. Chapter 2 contains information related to shibboleth, its overall architecture and information about plug-in development for shibboleth. Also, there will be discussion about some group management systems as use cases which will act as a source of group information.

Chapter 3 will concentrate on the requirements for devising the solution, the architecture of the solution and what steps will be required to achieve the final solution. It will also provide the advantage of a group management system in a federated environment. Further, I will describe the use cases for testing the developed solution. Chapter 4 will describe the detailed implementation process, general steps for connecting any group management system with shibboleth, group attribute sharing, group information synchronization and group information storage at service provider’s unit. Chapter 5 provides some test results on the performance of the designed modules. Chapter 6 describes the time frame and steps towards the organizational deployment of the designed solution. Chapter 7 contains information about similar projects and solutions. Finally, chapter 8 contains the conclusion of the thesis work.
2. Fundamentals

2.1. Shibboleth

Shibboleth is an open-source implementation developed by internet2 which provides authentication and authorization services to federated organizations using Security Assertion Markup Language (SAML) protocol. This section is about the architecture of shibboleth. The two main components of shibboleth involved in web based single sign-on and attribute exchange are the identity provider (IdP) and the service provider (SP). Both components comprise of many different core sub-components as described in fig 2.1 and further explained in following sub sections. The architecture explained in following sub-sections has been derived from the official shibboleth architecture documentation.

![Shibboleth Architecture](https://www.switch.ch/iai/demos/expert/)

Figure 2.1.: Shibboleth Architecture

1http://www.internet2.edu/
2.1.1. Shibboleth Service Provider

The shibboleth service provider (SP) resides at the organization hosting the service. Service provider’s primary task is to protect unauthorized access to a service, specifically an URL and its contents in the web server. Service provider resides in the same server as the HTTP server (preferably Apache) hosting the service. All provided technical details, discussions and development work has been performed using shibboleth SP version 2.5.4.

Service Provider consists of the following main components:

1. **Assertion Consumer Service**: It is an URL/URLs (HTTP resource) hosted by the Apache server which acts as receptor of HTML form submissions and eventually redirects the user to a requested resource. The secure context is established through new user session which is further handled by shibboleth service provider software.

2. **Session Initiator**: Used for generating authentication request between a user’s browser connecting to shibboleth service provider and extending the request to shibboleth identity provider.

3. **DS (Discovery Service)**: Primary task of discovery service is to facilitate automatic/manual selection of shibboleth identity provider which is serving a single shibboleth service provider.

4. **SAML POST**: Part of assertion consumer service that handles the received SAML assertion from shibboleth identity provider.

5. **Shib Module**: It is a module build to facilitate shibboleth service provider integration with Apache or any other HTTP server.

6. **Shib Daemon**: Dedicated service running in the background in an operating system to invoke shibboleth service provider related calls and services.

7. **Web Resource**: The web resource which resides in HTTP server and protected by shibboleth service provider.

2.1.2. Shibboleth Identity Provider

The shibboleth identity provider is the entity responsible for authenticating users and release of attributes. The authentication information and attributes are together bundled in a SAML assertion which is sent to the service provider. All provided technical

---

2https://webauth.service.ohio-state.edu/~shibboleth/shire.html
details, discussions and development work has been performed using shibboleth IdP version 2.4.3. The components involved in the authentication and authorization process by shibboleth identity provider are:

1. **Authentication Engine/Authority:** It is a SAML based service responsible for authenticating the user’s credentials and issuing an authentication assertion for users to be able to access the respective shibboleth service provider. Authentication engine does not directly authenticate the user but uses the defined authentication handler for the given user. The authentication handler information is set by an identity provider administrator during installation. For more information please refer to installation guide in appendix A.

2. **Attribute Authority:** SAML based service used for processing attribute query requests and issuing attribute assertions to service providers with mutual authentication. Further task include, filtering of attributes before release based on service provider specific filtering policy within the federation.

3. **Attribute DB:** It is the database or application used in the backend for storing attributes belonging to users. Multiple backend storage systems are already supported by shibboleth identity provider. The most popular are LDAP and Windows Active Directory. Other extension based backend support could be provided for MySQL, MongoDB[10], etc. Generally speaking, any attribute providing system can be ported into shibboleth identity provider provided that a java connector API exists for the specific application.

4. **Authentication DB:** It is the database used for storage of authentication information of users. Shibboleth identity provider provides active support for authentication via LDAP server for username/password based authentication. Third party extensions are required to provide authentication support for MySQL. Some other third party authentication extension could be obtained from [contribution](https://wiki.shibboleth.net/contribution) page of shibboleth.

5. **Single Sign-On (SSO) service/profile:** It is an HTTP resource contained within shibboleth identity provider that receives and processes authentication requests sent from user’s browser via shibboleth service provider. Authentication process is initiated by SSO service, further, it helps in issuing proper HTTP responses to user’s browser after interacting with authentication authority.

6. **Username Password Authentication:** It is one of the many authentication handler types available in shibboleth which displays the authentication form after redirecting to identity provider from service provider, takes the user’s credentials for authentication and sends the data to the authentication authority via HTTP POST. Other available authentication handlers are IP address based authentication, Remote authentication, etc. Any other authentication handler other then Username Password authentication is out of scope of this thesis paper. Interested
2.1.3. WAYF

“Where Are You From” is an optional service in shibboleth which sits in between shibboleth service provider and identity provider. Its task is to discover correct identity provider for incoming requests from service provider. Mostly, shibboleth service provider contains an in-built WAYF service. In our case WAYF is already implemented in shibboleth service provider and a more extended version of it is available as shibboleth discovery service. Going deeper into this topic is not required as it is not relevant for this thesis work. Interested people can read more about it at https://wiki.shibboleth.net/confluence/display/SHIB2/DiscoveryService, the wiki at official page for shibboleth documentation.

2.1.4. Flow of data and control in Shibboleth

Figure 2.2.: Shibboleth control flow
Figure 2.2 shows the complete flow of data and control between the various parts of Shibboleth as discussed in previous sections. Following is the explanation of Shibboleth control and flow where the numbers correspond to the process number in fig 2.2.

1. User accesses the service through a web browser, the service which is protected by Shibboleth service provider checks if the user is authenticated to access the service.

2. Session initiator creates authentication request and sends it to the browser with information about how to reach Shibboleth identity provider (URL).

3. Browser sends the request to Shibboleth identity provider using HTTP POST. Identity provider checks if user is authenticated or not, SSO service will take appropriate steps for selection of proper authentication handler.

4. Web browser gets redirected to username/password login handler (set by administrator). The selected authentication handler is username/password authentication handler. Appropriate web form is displayed to the user for authentication.

5. User enters authentication information (username/password) and presses submit. The browser sends HTTP POST with the authentication data to the authentication engine. After successful authentication, attribute authority proceeds with attribute resolution and filtering. A SAML assertion is prepared containing an authentication statement and an attribute statement containing user’s attributes.

6. The SAML assertion is sent as HTTP POST from Shibboleth identity provider to Shibboleth service provider. Service provider processes the SAML assertion, releases attributes after service provider side attribute filtering and redirects the control towards the requested resource by the user at the beginning.

7. After redirect, Shibboleth service provider checks again whether the user is authenticated, and finds the user as authenticated this time. Finally, user is provided access to the resource. The web page related to the service requested is displayed in user’s browser.

### 2.2. Mailman

This section contains brief overview about GNU mailman as it is being used as one of the use case for providing group information. GNU mailman is a free software used for electronically managing mailing lists. Generally it is used for managing electronic mail discussions and e-newsletter lists. Mailman provides ease of manageability due to its integration with web. Users can easily manage their online subscription accounts and list administrators can manage their lists through online portal.
Chapter 2. Fundamentals

Mailman can be used for membership management, a list administrator creates a public or private list and either adds users manually to the list or users can request for membership into public list. These group of list act as a well managed user pool. Mailman also supports hierarchical list creation. The parent list comprises sub list which in succession contains more sub lists or list members. These type of hierarchical lists are known as Umbrella list in mailman.

Umbrella lists are quite important within organizations as well as educational institutions as most of the departments form a hierarchy of sub-departments and members.

Figure 2.3.: Mailman Umbrella list

Figure 2.3 displays a simple example of mailman umbrella list. Any email sent to the parent list “University of Göttingen” will eventually be distributed to every subsequent list and its members. In the same way any email sent to the list “Practical Informatics” will be delivered to all of its subsequent lists and its members. This hierarchy of mailing list comprises group membership information which will be leveraged while including mailman as an use case for group information provider with shibboleth. Currently mailman could be installed only on Unix operating system and doesn’t supports windows operating system.
2.3. UserFrosting

UserFrosting\(^3\) is an opensource user management system developed by Alexander Weissman\(^4\). The web based software is written in PHP\(^5\) and uses MySQL\(^6\) database for storage of data. It is a freely available online for download and use, it is based on opensource licensing, installation and download instructions are available at the following link [https://github.com/alexweissman/UserFrosting/]. UserFrosting will support the concept of group information sharing from MySQL database. MySQL’s user interfaces is designed mostly for running sql queries. Therefore, a web based or standalone software with MySQL database in the backend is required to be used as a group information provider. The opensource licensing of UserFrosting gives the right to modify and customize the code as per requirement to fetch group information.

UserFrosting provides a backend Ajax\(^12\) API for querying data from the server. Ajax API allows a pathway to contact web server for retrieving user data. A simple HTTP GET or HTTP POST request is initiated from client’s system at the backend to contact a predefined location at the web server. This predefined location is served by a PHP script which checks the parameters received via HTTP request and processes them to send a respond to the client. The data contained in the response is shown to the user. This method will help shibboleth IdP to retrieve data from UserFrosting’s MySQL database.

2.4. LDAP

![LDAP directory tree](http://www.openldap.org/doc/admin22/intro.html)

Figure 2.4.: LDAP directory tree

\(^3\)http://www.userfrosting.com/

\(^4\)http://alexanderweissman.com

\(^5\)http://php.net/

\(^6\)https://www.mysql.com/
Chapter 2. Fundamentals

LDAP is a popular directory service protocol used for information sharing in intranet and internet applications. LDAP could be used for sharing users, systems, services and applications information. Data inside LDAP is arranged in a hierarchical format as shown in the example in fig 2.4.

From the root node, the directory is separated among different internet domains. Individual domains can contain different organizations as child element in the hierarchy. Organizational nodes can have different organizational unit like people, servers, etc. as child nodes. Organizational units contains individual node for individual entity, for instance people organizational unit can have persons. Each person possesses all the relevant attributes like, uid for identification, name, address, etc. A classic example is a hierarchy of email list in an organization. Identity validation through LDAP is the default process used in shibboleth IdP to authenticate users. Moreover LDAP can provide additional attributes related to users like name, email etc. to shibboleth service provider. LDAP can be used for creating group of users as well. This group information can be shared in the form of attributes with shibboleth service provider.

2.5. Active Directory

Active directory is a directory service developed solely for Windows operating system. Active directory stores the data in a similar fashion as it is stored in LDAP. There exists a hierarchical data structure storing domain, organization, organizational units and users. Users information is accompanied by different attributes related to users information. It is also possible to store group information in active directory like email hierarchy. Users can create new groups in active directory and add or remove members to the group. This group information and membership related information could be later shared with shibboleth service provider. Figure 2.5 contains a typical example from LDAP showing a domain forest.

![Active directory domain forest](image-url)

Figure 2.5.: Active directory domain forest
A domain forest is a hierarchy of domain names. In the above example *gwdg.de is the parent domain and other domains follow with added prefix to parent domain. Let’s take an example of a session cookie. A cookie made for *gwdg.de will give a user access to all the sub-domains. If a user is given cookie for domain email.gwdg.de then access is given to only sub-domains of email.gwdg.de. This is an example of authorization. Replacing the domains with groups will help in group based authorization. This authorization information could be shared with shibboleth service provider for authorization of services.
3. Heterogeneous group management

This chapter covers the importance of heterogeneous group management system in an organization. Further, it covers the requirement analysis towards planning and development of such a system.

3.1. Heterogeneous group sharing and advantages

3.1.1. Group sharing

Federated identity management and privilege management are supported by most of the modern federated management system. Shibboleth can use LDAP server in the backend for providing authentication and authorization information simultaneously. The main task of shibboleth is to perform authentication while the authorization information is shared with the service provider in the form of attributes[13]. Service provider is responsible for further authorization of services based on the attributes received. As stated earlier, most organizations have multiple systems which could contain valuable group related information, but these systems are not directly connected to some already supported backend data provider in shibboleth like LDAP server for authorization purpose. When federated organizations need to share the group information retrieved from these systems they will be forced to synchronize the group information into an existing shibboleth attribute provider service like LDAP or MySQL server. Or else the group information provider system needs to establish a direct contact with the service provider to provide data which will impose added policy making, planning and development. Sharing the attributes through shibboleth is easier and recommended as, shibboleth already takes care of authentication, and its also possible to release attributes in the same channel.

Ability to share group information using heterogeneous group information provider systems is advantageous in a multifaceted organization. Before going deeper into the advantages of heterogeneous group management system lets discuss the advantages of group sharing in a federated environment. At the beginning of this thesis report there has been discussion on the possibility to provide access to a group of users from external organization. This task involves importing group information from external federated organization for providing access instead of re-creating the group of users.
Chapter 3. Heterogeneous group management

The biggest benefit leveraged by importing group information is mitigation of redundant group creation. This is quite identical to federated identity management where redundant user account creation is minimized. The advantage of importing group information instead of recreation can be easily realized in a situation where the group contains hundreds of members. Additionally, users are authorized to existing groups immediately after their first login based on the shared group information attributes coming from their home organization.

![Figure 3.1.: Benefits of sharing group information](image)

Importing group information in a federated environment adds benefit for the administrators as they are not burdened to create individual groups and add members into it. Time is saved in group creation and users receive access in a more efficient and faster way. Additional information could be carried with the general group information for more varied services. Group information could carry attributes related to preference of users within the group. Considering the use case for sharing group information for courses, roles of users within the group could be shared, for instance if a user is a teaching assistant, a tutor or a student, thus organization B (fig 3.1) will have a course with users from organization A and having similar roles for user A (tutor) and user B (student).

There is an obvious benefit of importing group information in scenarios where frequent group creation takes place and the groups being created are either derived or are duplicate of existing groups. If a group tag is attached to the group information attribute then a search facility could be availed to search similar or like minded groups. For instance a group of physicists interested in astronomy could have the following tags, “physics”, “astrology”, “extra terrestrial objects”, etc. Searching on this tags can fetch groups and there working.
3.1.2. Heterogeneous group sharing

Figure 3.2.: Heterogeneous group management system

Heterogeneous group management systems inherits the benefits of group sharing in federated environment as discussed above. Additionally, heterogeneous group management systems minimize the boundary of including any available group provider in the market, for using it to provide valuable information regarding group members and additional attributes to support the group functionality. Users are not bounded to use only supported (by shibboleth) group management system like LDAP. There is possibility to leverage group information from enterprise applications like human resource management application or membership management application and more.

Without the presence of a heterogeneous group sharing framework users will be forced to synchronize group information to the only available group management application supported by the federated identity management solution. In case of shibboleth it could be LDAP or Active Directory.

As shown in fig 3.2, organization A consists of 3 separate group information storage system and all can contribute individually the group information for a user. Later part of this research paper shows that it’s possible to fetch information from some of the group management systems using shibboleth’s attribute manager in identity provider. Also group information from different systems could be either merged together and sent as single identity, or it could be sent as individual attribute for each group management system.
3.1.3. Usage at GWDG

Heterogeneous group management system could help institutions to import groups while using services provided by GWDG. GWDG is the IT service provider for Georg-August-Universität Göttingen and the Max Planck Society in Germany. Following are some use cases where group creation through importing might be required to share a service among a group of users:

1. **File sharing:** It is quite common to share files with a group of users for academic work or organizational work. Online storage and file sharing mediums allow you to create groups of users collaborating on a single folder. Users belonging to such a group could be imported from the home organization of the users.

2. **Virtual machine sharing:** Group of users working on a project might require to use the same Unix system. Or they might want to share the available computing power, memory and storage area. In such a situation group access could be provided by importing the group related information to create automatic Unix accounts for the group members.

3. **Access to managed services:** GWDG provides many managed services to the users like Wiki, Git or SVN. These services could involve collaborative work among a group of users. Access to the managed services could be authorized using group information imported from shibboleth identity provider. A read access to these services is usually provided to all users (except for private service groups). Administrative rights could be delegated using imported group information from identity provider.

4. **Access to paid software:** GWDG hosts many paid software for which special access permission is required. A group of users requiring access to a software like Microsoft Office 365 could be delegated using imported group information to external users within federated organizations.

3.2. Requirements Analysis

This section contains discussions regarding the requirements analysis for designing the heterogeneous group management system. The steps are part of IBM’s suggested pattern for requirements analysis.

---

1 Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen: http://www.gwdg.de/
Chapter 3. Heterogeneous group management

Figure 3.3.: Functional Requirement Analysis

Figure 3.3 shows a chart of the functional requirement analysis for the development of the synchronization of heterogeneous group management system in Shibboleth. The core functionalities of the requirement analysis have been discussed in following sections.

3.2.1. Business processes

The major requirement of the business process is to allow sharing and synchronization of group information from heterogeneous group management system in a federated environment using Shibboleth. It should provide the ability to connect Shibboleth identity provider with different heterogeneous group information storage systems being used by an organization. The connection will allow sharing group information and related attributes with service providers from multiple group management systems to avoid redundant group creation and for extra services based on group related attributes. The solution will allow users to self authorize other users in the form of groups to access dedicated services.
3.2.2. IT processes supporting business processes

The core IT processes supporting the business process are:

1. **User management**: User management in the context of this thesis work is restricted to managing existing users within a group and doesn't involve new user creation or deletion.

2. **Access management**: Access management is a core functionality as the groups are being shared for authorization of dedicated services. Only members who are part of a predefined group will get access to the service. Access management will be performed by group administrator and supported by additional modules.

3.2.3. Activities within each IT process

The IT processes comprises the following activities:

1. **Group related activity**: The primary activity of user management process is to provide group management related activities. This activities could belong to either shibboleth identity provider or shibboleth service provider. All the functions belonging to this activity will support group administration and group sharing in shibboleth.

2. **Access regulation activity**: Access regulation activity supports the access management IT process. The core task of this activity lies in authorization mechanisms and periodically update group members information for real time authorization.

3.2.4. Functions within each activity

Group related activity can be subdivided for identity provider and service provider. The core functions of identity provider are create, update or delete a group, view members of a group and change the privacy setting of the group. Changing the privacy setting will control sharing of group information outside the organization.

Further group related activity in identity provider has functions to add or delete users from a group. This task is mostly handled by the group administrator and this task also acts as a building block of authorizing services.

In service provider, group related activities requires functions to import or delete existing group and also allow administrator to view which members of the group are active in service provider. The active members information is stored by service provider during the first login of the user to access the service.
The third functionality in the group related activity is the aggregation of group information data. Multiple systems will connect to Shibboleth identity provider, the group aggregator function pulls relevant group information related data and attributes and packs them for sharing via Shibboleth identity provider.

Access regulation activity requires two functions. One of the core function will grant or revoke access to a service based on the membership information of a user to a predefined set of group defined by group administrator. Another function will synchronize the group information for logged in users to support real time authorization of services.

### 3.2.5. Use cases for the functions

Use cases surrounds the two prominent users of the system i.e. group administrator and group members. Figure 3.4 shows the functions discussed above and the supporting users who can access these functions.
As shown in fig 3.4, group administrator has access to all the functions except the supporting functions. At service provider, group administrator can import a group, delete an existing group or view the active users of the group who have made the first login with the service provider. Before first login, shibboleth service provider is completely unaware of the existence of a user. Users data is generally stored in service provider for saving user specific preferences and to identify returning users to the service. A group user might be allowed to view the members of the group.

An organization hosting identity provider, group administrator can add or remove users from existing groups, create, modify or delete groups and view existing members of the group. A group user might be allowed to view the members of the group.

Supporting functions is the group of functions which are not directly controlled by the group admin or group members but it provides aid in the functioning of the overall system group management system. Group data aggregator works at the organization hosting identity provider. Grant or revoke access as well as function to sync user’s group information are part of organization providing the service to users.
4. Implementation

In this chapter I have discussed the planning and development involved in building individual modules of the architecture of group information sharing system.

4.1. Core elements and architecture

For allowing shibboleth to share group information in an federated environment, some additional modules are required. These modules support various tasks of group information sharing as well as group member’s authorization.

![Diagram of modules used in group sharing framework](image)

Figure 4.1.: Modules used in group sharing framework

Figure 4.1 shows the shibboleth architecture improvised with the additional modules (highlighted in bold) to support group sharing in federated environment using different types of heterogeneous group information storage systems. The first task is to prepare
the group management systems and the user interface accessible by group administra-
tors and group members. User interface will allow the group administrator to create, update, delete, hide or view a group as well as add or remove new members from the group. Additionally user interface will allow group members and group administrator to view the current members in the group.

Next requirement is to develop APIs for connecting shibboleth IdP with individual group information provider application. This needs development of individual API for every group management system due to their heterogeneous nature. The task of the API is to read the parameters and requirements from shibboleth IdP and query the same from group information provider and return the results to shibboleth IdP. Shibboleth IdP forwards the group information as attributes to the SP. Details for development has been discussed in upcoming sections.

The above steps will complete the process of group information sharing in the identity provider’s organization. The next step is to setup the modules at the service provider’s organization. The user manager module in figure 4.1 is not part of the development in this thesis work, but normally user manager exists in the service provider’s organization to store the attributes and preferences of logged in user. User manager stores the data in a dedicated database associated with the service for storage of user information. Even for services allowing anonymous logins, existence of a user manager is possible which stores the IP address of the user for the duration of the session to save preferences. It's even possible to store the anonymous users information longer for auditing purpose.

Group manager module stores and processes group related data received from identity provider. A database is associated with the group manager to store group related data. Group manager will additionally provide group administrator with the interface to import or delete groups and will also allow administrators to check active members of the group. To check active members information, support of both user manager module and group manager module will be required. User manager acts as the supplier of active members information which is matched against group information by group manager to find out active members.

Single Logout (SLO) module supports the access revocation of a user from active services. Its main purpose is to remove all existing session of the user as he might have logged in into multiple services through single sign-on. During the user’s first login, sessions are created at both IdP and SP in shibboleth as well as a session might be created in the service itself. Afterwards, additional session could be created when user accesses additional services through single sign-on. The core purpose of single sign-on is to allow users to login once and access multiple independent services without the need to login again. Logout from services is an important aspect from the point of view of access revocation. Therefore, services residing on sub-domains of a single parent domain can achieve SLO by destroying the cookie to its parent domain. SSO sessions
could exist within the same domain or the services could reside on different unrelated domains, in this case it is not possible to destroy all the active session by just destroying the session for one service. A more sophisticated logout mechanism is required to achieve true single logout in systems supporting SSO. Shibboleth’s in-built SLO mechanism is not very efficient\footnote{https://wiki.shibboleth.net/confluence/display/CONCEPT/SLOIssues} and requires custom development to support true SLO from all the services.

The last module is the attribute synchronizer. The core task of this module is to periodically query updated attributes from IdP for a specific user. Updated attributes reflects changes in group membership for a user. This will help in access grant or access revocation of a service for the user. Both SLO and attribute synchronizer will be invoked by group manager when required. Group manager will periodically invoke attribute synchronizer for updated attributes and will invoke SLO when a user has been removed from a group and his active sessions needs to be removed. The SLO will logout the user from all the existing sessions of a service for which access have been given due to the users membership in the respective group linked with authorization of a service.

Following sections describes the implementation method for individual module. Additionally it contains information on various available solution for each module and valid arguments on the selected solution.

### 4.2. Group information providers UI

Group information management system requires a graphical user interface to support group managers with a easy to use mechanism to maintain their groups. In this section, I have discussed the graphical user interface support for the four selected group information provider application I am going to consider, namely Mailman, UserFrosting, LDAP and Active Directory.

#### 4.2.1. Mailman

Mailman provides a built-in user interface in the form of a web portal accessible with the help of CGI scripts. The user interface is ready to use for new group creation in the form of mailing lists. Each mailing list contains an administrator and many users. Users identification is there email address. The portal is accessible in the path http://[hostname]/mailman/admin/[list name]/. The portal allows users with sufficient privileges to create new mailing list. This option is accessible from the admin panel in the path http://[hostname]/mailman/admin/. New members can be added or removed from...
the membership management panel in the administration page. Membership management displays the current members in the list and also provides panels to bulk insert many users simultaneously or select and delete multiple users at a time. The membership panel is accessible at http://[hostname]/mailman/admin/[list name]/member/. Users can control the privacy of the mailing list through the control panel. First open the particular mailing list’s administration panel and go to privacy options, under privacy option there is “subscription rules” with option to advertise the mailing list. Advertised mailing lists are returned in result set when a user requests all the mailing list in the server through shell command. This feature of advertising mailing list could be used later during API development for the purpose of hiding the list from sharing among other federated organizations.

Mailman supports all the functions listed in the use case for identity provider’s organization in fig 3.4. All the functions are supported in the default functionality of mailman and there doesn’t exists any requirement to modify or extend any feature of mailman.

4.2.2. UserFrosting

UserFrosting supports membership management and group management. The user interface supports user creation, user deletion, group creation and group deletion. Group membership is provided to all new users being added to the system. The current user interface lacks vivid functionalities to manage group information and membership efficiently, but it could be easily extended and customized as per requirement due to the opensource licensing of the software. In conclusion, the user interface partially supports group management as it lacks the ability to remove users from groups using the user interface. But the software supports the sole purpose of group management system backed by a database system (MySQL), which could be used as group information provider among federated organizations.

The user interface of UserFrosting will not be modified in this thesis work because for testing perspective it is sufficient to only receive group attributes from the system.

4.2.3. Light Weight Directory Access Protocol

Light Weight Directory Access Protocol (LDAP) is a popular open source directory access protocol which is often used for user management as well as authentication. In fact shibboleth by default authenticates users through LDAP or Active Directory. There are many LDAP servers freely available in the market for installation. OpenLDAP\(^2\) is the preferred LDAP server for installation as it supports `memberof` attribute, I will later show the requirement for this attribute in group sharing in federated environment.

\(^2\)http://www.openldap.org/
LDAP installation doesn’t come with a pre-installed user interface. But there are many freely available and highly capable user interfaces that can interact with LDAP and helps in performing administrative work on LDAP server. Apache directory studio[^3] and phpldapadmin[^4] are known and stable for using as an user interface with LDAP server.

After installing an user interface it’s very easy to control and administer user groups in LDAP server. The user interface supports all the functions in the use case for identity provider’s organization in fig 3.4. Group administrators can create new groups or delete existing ones and add or remove members from the group. The `memberof` attribute is useful for returning the list of groups in which the user is a member, without the need to search and loop through every group. Although this attribute has the drawback that it doesn’t return results from nested group structure, in that case recursive search is an better option to find group membership. The other requirement of the search filter is to return the owner of the group in response to the query. The `managedby` attribute in LDAP could be leveraged here to find out the prospective members with administrative privileges. The owner will be given the privilege to import the group in shibboleth service provider.

### 4.2.4. Active Directory

Active directory (AD) is a windows based directory service, ready to use for authentication and authorization services in windows operating system. AD can be used for user management as well as group management. Active directory’s user interface could be used for supporting the functions in the use case for identity provider’s organization in fig 3.4. Active Directory Administrative Center[^5] is the supported active directory GUI for Windows based systems. The user interface allows group creation, deletion, modification as well as provides the ability to view current group members. New users could be added by modifying the members tab in properties for a group. The `memberof` attribute in active directory is also beneficial as it eliminates the need to loop through every group of the active directory system to search membership information for a specific user. Though the `memberof` attribute doesn’t report nested groups in the result. A carefully developed query is required which can search all the relevant groups in which the user is a member. Search filters might be helpful in hiding groups from search results for private groups that are restricted for sharing in a federated environment. The search filter should also be able to return the owner or user with administrative power for the group. There exists the `managedby` attribute in AD groups which could be helpful in finding the list group administrators for the consecutive group. This member will be given the right to import the group at shibboleth service provider.

[^3]: http://directory.apache.org/studio/
4.3. API development

This section explains the process of extension development to build API’s for individual group management systems and query group information.

4.3.1. Basics

API development for shibboleth IdP is the process of developing extensions or plug-ins in java which can be embedded with shibboleth IdP. The documentation given at the shibboleth wiki page provide highly technical information on extension development and contains adequate detail for an experienced developer\[18\]. This section contains general overview of extension development for shibboleth IdP, the expected knowledge of a developer and the steps needed for planning an extension development.

There are many types of possible extensions which could be developed for shibboleth IdP. The most interesting for this research topic is development of a data connector\[19\]. Shibboleth wiki strongly recommends the need to have good understanding of Spring framework\[6\] as well as the knowledge of XML\[7\] and XML schema\[8\] to proceed with development of extensions. Additionally a thorough understanding of java web development is expected from the reader as well as the developer.

Extension development for shibboleth IdP can be divided into three parts as described in tab 4.1.

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>Lists the set of actions the extension can perform on the connected application.</td>
</tr>
<tr>
<td>Input attributes</td>
<td>List of attributes necessary for the extension to work properly.</td>
</tr>
<tr>
<td>Output attributes</td>
<td>Predefined or dynamic output variables could be set in the extension.</td>
</tr>
</tbody>
</table>

Table 4.1.: Identity provider extension parts

Operations are a set of actions the extension can perform on the connected application. For instance, a database related extension can establish connection to the database, execute some queries to search data and close the active connection to the database. Mostly, data connector extensions are used for data retrieval from an existing database. So it

\[6\]http://www.springframework.org/
\[7\]http://www.w3.org/XML/
\[8\]http://www.w3.org/XML/Schema
will include connection establishment and connection removal from the external application. Further, it will allow to run a custom search query provided by shibboleth IdP administrator or developer. The custom search query will return a response with some result.

Input set of attributes prepares the extension with all the required values useful while a users data is retrieved by attribute authority. This attributes could be connection parameters like host, username and password for establishing connection. Further a pre-defined search query could be provided by administrators which runs during attribute authority call, users identification attribute could be supplied additionally to the search query.

Output variables are either predefined within the extension or dynamically defined by administrators. Data type of the output attribute is defined during configuration along with column names for identification.

```
<resolver:DataConnector id="DBRef" xsi:type="RelationalDatabase" xmlns="urn:mace:shibboleth:2.0:resolver:dc">
  <ApplicationManagedConnection>
    jdbcDriver="com.mysql.jdbc.Driver"
    jdbcURL="jdbc:mysql://localhost:3306/shibdb"
    jdbcUserName="shibadmin"
    jdbcPassword="abc123"/>
  <QueryTemplate>
    <![CDATA[
      SELECT groupname FROM user_groups WHERE username = '$requestContext.principalName' AND groupname <> 'role'
    ]]>  
  </QueryTemplate>
  <Column columnName="groupname" attributeID="groups"/>
</resolver:DataConnector>
```

Code Snippet 4.1: MySQL attribute resolver configuration

Code snippet 4.1 shows configuration of MySQL with input and output attributes and search query. Input attributes provides the connection parameters (jdbcdriver, jdbcURL, jdbcusername, jdbcpassword) and output attribute (<column columnName="" attributeID="" />) defines the column name mapping between the attribute and search queries response. The QueryTemplate takes a string which is a select query, username for the user currently being authenticated is provide by the $requestContext.principalName. All the
Chapter 4. Implementation

operations takes place using the MySQL JDBC\(^9\) connector plug-in which creates the bridge between shibboleth IdP and MySQL server.

Once all the three requirements are prepared, extension development can be initiated. Following is a flow of steps required for developing the extension based on shibboleth wiki\(^{18}\):

**Step 1:** First step is to setup the extension template provided at the wiki page of shibboleth IdP extension development\(^{18}\). The imported project is Maven\(^{10}\) based, so any dependencies of the extension development can be imported through adding dependencies in pom.xml metadata file. Final result of the extension development is a JAR file. Shibboleth IdP needs to be built/rebuilt with this JAR file placed in lib directory to be able to use the extension.

**Step 2:** Plug-in’s schema definition describes the properties needed by the extension. Properties corresponds to input and output attributes. Schema file allows the IdP to perform validation of input attributes based on constraint like mandatory field. The schema definition is an XML schema file, it contains many different elements describing the properties of the attributes like data type, requirement, etc. The schema definition file is used to declare the input and output arguments supported by the extension. The schema definition file is placed at “project_directory/main/resources/schema” location from project directories perspective.

**Step 3:** In this step I create the Spring (framework) configuration files i.e., spring.schemas and spring.handlers. Both files are placed in the path “project_directory/main/resources/META-INF”. The spring.schemas file informs Spring the location of the schema file based on a particular namespace. Below shown is an example of a schema file which contains the XML namespace URI followed by the location of the schema file from projects perspective.

```
urn\:example.org\:shibboleth\:2.0\:resolver =
schema/myConnectors.xsd
```

The spring.handlers file tells Spring the NamespaceHandler used for identifying a particular namespace. A typical example is shown below with XML namespace URI followed by fully qualified class name for the given NamespaceHandler.

```
urn\:example.org\:shibboleth\:2.0\:resolver =
com.example.shibboleth.MyDataConnectorNamespaceHandler
```

\(^9\)http://dev.mysql.com/downloads/connector/j/
\(^{10}\)https://maven.apache.org/
Chapter 4. Implementation

Backslash (\) are used as escape character due to special meaning of colon (:) character in property files.

**Step 4:** In this step I create a Spring `BeanDefinitionParser` and a `BeanFactory` method. The work of this methods are, to parse XML configuration for the extension and set the required bean properties and/or constructor arguments for the new java class developed for the extension in step 4. In simple terms this class will get the list of attributes required by the extension, gets the corresponding values of the attributes and sets them for use by the extension’s main java class file from step 5. The `SCHEMA_TYPE` parameter is set in this class which provides the QName\[11\] for the extension type.

**Step 5:** Create a new java class file by extending `BaseDataConnector` class. Return statement consists the list of attributes which will act as output attribute’s value. All the required input arguments are passed as properties and accessed through constructor or by means of a setter method. The set of operations the extension is expected to perform are written in this new java class file. Supplementary java class files could exist supporting the functionalities of the new java class file. The default function `resolve` is used to return the output attributes received after completing operations. The attributes are returned in `Map<String, BaseAttribute>` format to attribute handler.

**Step 6:** Here I implement a `NamespaceHandler` class by extending `BaseSpringNamespaceHandler`. Its work is to instruct Spring to invoke `BeanDefinitionParser` for the given extension using an `init()` method. `init()` method calls the function `registerBeanDefinitionParser()` with two arguments. First argument is `SCHEMA_TYPE` field defined in `BeanDefinitionParser` at Step 4 and the second argument is an instance of `BeanDefinitionParser`.

**Step 7:** Installation of the extension is straightforward. The JAR file produced after extension development is placed in `lib` folder of shibboleth IdP source folder and afterwards the IdP is rebuilt/installed. After rebuilt/installation is completed the new namespace is added in `attribute-resolver.xml` for using the extension. The namespace and XML schema file location are added to the root element as shown below:

```xml
<resolver:AttributeResolver
 ... 
 xmlns:uid="urn:example.org:shibboleth:2.0:resolver"
xsi:schemaLocation="
 ... 
 urn:example.org:shibboleth:2.0:resolver
classpath:/schema/myConnectors.xsd">
```

\[11\]http://docs.oracle.com/javase/7/docs/api/javax/xml/namespace/QName.html

30
Next requirement is to add a data connector whose work is to establish connection to the external application being used for data extraction. DataConnector element takes the input and output arguments additionally. Finally an AttributeDefinition element which defines the new argument going to be shared with shibboleth SP. DataConnector acts as a dependency to the AttributeDefinition element.

4.3.2. Mailman API

The first step towards API development for mailman is to identify the input and output attributes as well as the operation supported by the API. Group information from mailman will be extracted using Unix shell commands, as mailman already provides built-in commands to extract membership information as well as group administrators information. Special measures needs to be taken for distinguishing groups which could be shared among all federated organizations and those which are hidden from external organizations. First way of hiding the groups has been mentioned in section 4.2.1 where privacy options from mailing list administration panel could be used to mark whether the group should be advertised or not during a group listing. The API will work by establishing a secure channel from shibboleth IdP to mailman server and execute shell commands to retrieve the data. Following are the steps required to get group information from mailman server for an user. This steps designates the operations performed by the API extension.

**Step 1:** First step is to establish a secure connection to mailman server. JSch is a java based library which is used to establish SSH connection to a Unix server. To establish SSH connection using JSch following parameters are required: hostname, username, password and port number. All this parameters will be provided as input attributes in `<resolver:DataConnector>` element in attribute-resolver.xml file. hostname (IP address of mailman server), username, password and port number are used to establish SSH connection with mailman server. Thus an utmost requirement is to setup a SSH server at mailman server to accept SSH connection.

**Step 2:** Once the ssh connection is established the first shell command is executed to fetch the list of mailing lists from mailman after filtering out private mailing lists. The command `list_lists -ab` is executed to get the required mailing list names. Here list_lists by default displays all lists with description. The additional parameters `-a` and `-b` displays only advertised lists (hides private list) and removes mailing list description respectively.

**Step 3:** The list of mailing list obtained from mailman is further used for querying the membership information of a particular member. Member’s username or user id is

12http://www.jcraft.com/jsch/
provided in the input attributes with other attributes as mentioned in step 1. The command `find_member -w -l [groupA] -l [groupB] ... -l [groupZ] [username]` is executed to find out if the username exists in the given list list names and whether the user id is an admin in any of the given lists. The main task of `find_member` command is to identify a user’s membership in a list of mailing lists. `-w` argument is used for identifying the group administrator and `-l` argument suffixed with list name(s) is used to conduct search only on given list name(s). The final argument of the command is the username which needs to be searched for membership. The final output of the query is like following:

```
find_member -w -l mailman -l gwdg idp@localhost.de
```

Output:

```
mailman (as owner)
gwdg
```

The output shows result of search on lists `mailman` and `gwdg`. The user `idp@localhost.de` is present in both the lists and the user is the owner or admin of the `mailman` mailing list.

Step 4: In the next step the connection from the SSH server is closed and the output of the Unix command is prepared to be sent to shibboleth IdP in the form of output attribute column name defined in `<resolver:DataConnector>` element in attribute-resolver.xml file.

There are two other ways to control the privacy of a group name or mailing list in mailman. The first alternative way uses the built-in access control in Unix using which access to a folder or file could be restricted. For hiding a group, the read access to the group’s location in Unix server could be restricted for the user initiating the search query. This user is provided in the input attribute as `username`. General path to group’s location is `/var/lib/mailman/lists/[group’s name]`. To make this solution work, mailman’s admin panel needs to be modified to provide a new option for group privacy management. Changing the privacy option will run a shell script in the background to modify read permission of the folder/file for the respective group. So, when the user executes `find_member` command read access to the private groups is denied and by default the search for membership is made only on the public mailing lists. This solution involves running only one shell command `find_member -w [username]` to get the desired output.

The second alternative way uses an additional parameter of the `find_member` command. The argument `-x` excludes the mailing lists from membership search criteria. So, the final command is `find_member -w -x [groupA] -x [groupB] ... -x [groupZ] [username]`. All the list names provided with argument `-x` will be hidden from membership search. This method requires shibboleth IdP administrator to provide the exclusion list in the input attribute list in `<resolver:DataConnector>` element in attribute-resolver.xml file.
The above steps combined with extension development steps will give the mailman API extension.

### 4.3.3. UserFrosting

The API for UserFrosting needs the following input attributes: `hostname` and `user_id`. `hostname` is the URL used for accessing UserFrosting’s index page. `user_id` is an unique id generated for each user at the time of new user addition. The extension operates by accessing user’s group information by calling the backend PHP script stored in UserFrosting web server for serving Ajax calls. The request is sent in the form of HTTP GET request to the server and the URL for the request is of the following format:

```
http://[UserFrosting URL]/api/load_users.php?
user_id=[user’s id]
```

The output is received in JSON format from the server which contains users information in addition to group information. By default UserFrosting releases very few group information which is generally not sufficient in my case. Therefore, some modifications where made in the backend PHP scripts to release relevant group information stored in the MySQL database. The final output with group information is returned in the output attribute.

The above steps combined with extension development steps will output the UserFrosting API extension.

### 4.3.4. LDAP

LDAP API is pre-existing in shibboleth IdP. Therefore, I only show the usage of the LDAP to retrieve group information. The basic setup for `<resolver:DataConnector>` element is provided in code snippet 4.2.

```xml
<resolver:DataConnector id="myLDAP"
    xsi:type="dc:LDAPDirectory"
    xmlns="urn:mace:shibboleth:2.0:resolver:dc"
    ldapURL="ldap://localhost:10389"
    baseDN="ou=Users,ou=ShibUsers,ou=system"
    principal="uid=admin,ou=system"
    principalCredential="secret">
    <dc:FilterTemplate>
        <![CDATA[

(After code snippet 4.2)
```
Chapter 4. Implementation

The basic input attributes are `ldapURL`, `baseDN`, `principal` and `principalCredential`. These are basic parameters required to connect to LDAP server. `FilterTemplate` is used to provide LDAP filtering query. This query helps in filtering result sets. Filter query can fetch relevant group information for a user identified by the parameter `$requestContext.principalName`. The most important parameter to identify group membership is `memberof` parameter in LDAP filter query. The `managedby` parameter can be used to get group administrators information.

### 4.3.5. Active Directory

Shibboleth IdP actively supports AD as an attribute provider. Therefore, API development is not required because a carefully constructed filter query can release relevant group information from AD. Configuration for `<resolver:DataConnector>` element to release attributes from AD is similar to code snippet 4.2 provided for LDAP. The parameters to get group membership information and group owner information is also similar to LDAP, the functionality is supported by `memberof` and `managedby` parameters in AD filter query.

### 4.4. Single logout

“Web single sign-on (SSO) systems enable users to authenticate themselves to multiple online services with one authentication credential and mechanism offered by an identity provider”[22]. According to Suoranta[22] et al. most of the SSO systems gives more emphasis on SSO and only a few systems gives equal emphasis to the Single Logout phenomenon as well. An SLO is under development for the latest version of shibboleth IdP (version 3). According to their official documentation single logout is planned as an optional feature in a future release, currently it is not supported due to the lack of space in cookies for tracking the sessions[23]. When a user logout of a service, the usual expectation is all other active sessions of the user are destroyed as well which might have came into existence due to single sign-on[24]. Similar feature is expected by a group administrator when the access rights of an user is revoked. That means, the user should be logged out of all the active sessions which has been a result of SSO and due to the users group membership.
In general three types of sessions are created when user is provided access to a service in a shibboleth based federated environment. The first session is created at shibboleth IdP after the user authenticates, post that shibboleth SP creates a session based on the SAML assertion received from shibboleth IdP as well as the IDP session cookie and finally a session might be created at service level. A full fledged SLO mechanism should be able to destroy all the three sessions as well as the session created with other service provider by utilizing SSO. Currently logout is invoked using /Logout handler configured in shibboleth SP. Initial tests revealed that using the /Logout handler is able to delete the IdP session and the SP session of the logout initiator. Additionally the /Logout handler also tries to remove the session for all the other sessions created by means of SSO but it fails in removing other sessions. Please note the above result belongs to shibboleth IdP version 2.4.4. With shibboleth IdP version 2.4.3 /Logout is only able to delete the session of IdP and may or may not delete the SP session from the user’s browser.

Different possible solutions for SLO has been discussed by Suoranta et al. [22] and some effective solution has been proposed by them. Two solutions can help in implementing SLO in our case. First solution uses polling mechanism to check for active IdP sessions. This solution works for sessions created between multiple SP’s but authenticated by same IdP. The polling mechanism works in SP by periodically sending a message to IdP to check if its session still exists, in which case it renews the session within SP and of the service. If it fails to receive positive response from IdP then the session at SP and the service is destroyed. A negative response from IdP might come in two scenarios, first if IdP’s session has expired and second, if user has already initiated a logout request from another SP’s service. The drawback of this mechanism is added network load generated by the polling mechanism which periodically sends and receive messages from IdP.

Second solution which could be implemented is through session tracking by IdP. In this method IdP stores information of all the SSO sessions for every user. In case of SLO IdP could initiate a SAML logout on each SP based on collected session information. This method has its own drawbacks though, first it breaks users privacy as IdP is tracking each user’s service access. Moreover, IdP initiated logout is more efficient in SOAP based logout requests, this generates added burden for each service provider to support SOAP based logout requests. Moreover SOAP based logout work in the background, therefore it doesn’t provides any information to the user regarding the reason of the logout.

A third solution has been developed by Hungarian NIIF institute as an extension for shibboleth IdP. The NIIF SLO extension[25] can initiate logout from IdP as well as from SP. Moreover, it supports SOAP based logout as well as URL based logout. SOAP based logout works silently whereas URL based logout actually shows the logout process to the user in the form of redirection of pages and also displays proper messaging on success or error. NIIF SLO extension also returns proper response in JSON format after
Chapter 4. Implementation

The logout is successful. The extension works in both systems supporting JavaScript or those not supporting JavaScript. The extension might fail in some situations due to lack of support for session cache by Shibboleth IDP, though very less information is provided about the failure situations. Logout could fail definitely during IDP session timeout.

A fourth solution is based on the new feature of Shibboleth IDP version 2.4.4. The feature provides the user with information about all the SSO sessions the user has initiated during his service access. When the user executes logout using `/Logout` handle of Shibboleth, the user is logged out of the current session and IDP. Logout response contains the list of all the services the user is logged in into and an attempt is made to remove those sessions. Though in reality the session is not removed. As a solution, a background script could execute logout on individual sessions based on the response received from first logout request. This logouts could run on a hidden iframe and execute `/Logout` on every service. In this way all the users sessions will be deleted and true SLO could be achieved.

The only possible way to attain complete SLO is through a customized logout mechanism. A future solution by Shibboleth developers for IDP version 3 might be a permanent solution.

### 4.5. Attribute synchronizer

Attribute synchronizer will work in the background specifically at service provider’s server and periodically query IDP for updated attributes. The main motive to use the module is to receive updated group related attributes. Any change in the group information will help in access management for a given user. A logged in user will be forced to quit the service in case the updated attributes doesn’t contain the group information required for using the respective service. Similarly, a user might be given access to a service based on updated group information returned by the attribute synchronizer post login.

The attribute synchronizer uses an extension developed for Shibboleth service provider which can be invoked to get attributes for an user. This extension has been developed by Academic Access Management Federation in Japan\[26\]. The extension uses the built-in function of Shibboleth to request attributes if it was not delivered during the first login. This was the default method to get attributes until Shibboleth IDP version 1. A SOAP call is placed in the backend directing to IDP to release the attributes. The installation details for the plug-in is available in appendix A.1.

For using the extension an URL is formed which accesses the extension and passes on additional parameters. Following is the list of acceptable runtime parameters:\[34\]

\[14\]https://bitbucket.org/PEOFIAMP/shibsp-plugin-attributequery-handler#rst-header-runtime-parameters
Chapter 4. Implementation

- **protocol**: a protocolSupportEnumeration value to be used for querying IdP, or more simply it designates the SAML message encoding format which is acceptable to IdP. If omitted, urn:oasis:names:tc:SAML:2.0:protocol is used. You can also use the following aliases.
  - SAML2.0 alias of urn:oasis:names:tc:SAML:2.0:protocol
  - SAML1.1 alias of urn:oasis:names:tc:SAML:1.1:protocol
  - SAML1.0 alias of urn:oasis:names:tc:SAML:1.0:protocol

- **entityID**: entityID of an IdP, as mentioned in the metadata of IdP. It is the URL where IdP is reachable.

- **format**: a SAML name identifier format. If omitted, the following value is used
  urn:oasis:names:tc:SAML:2.0:nameid-format:persistent

- **nameId**: a SAML name identifier value, precisely the unique key to identify the user.

Mandatory parameters are **entityID** and **nameId**, all other parameters are optional and could be used as per requirement. A typical attribute request looks like following:

```
https://X.X.X.X/Shibboleth.sso/AttributeQuery?entityID=https://X.X.X.X/idp/shibboleth&nameId=user1@localhost.de
```

The first part of the attribute query format contains the shibboleth service provider’s path with an additional invoke to the handler AttributeQuery, followed by the entityID of shibboleth IdP and the nameId for each attributes are being queried. The response comes in JSON[27] format as shown below:

```
"sn" : "One",
"cn" : "User One"
```

The response will contain all the information attributes for the requested user released from IdP. The attribute query is limited to users whose IP address has been provided in the access control list in the handler invocation in shibboleth SP’s configuration file. In general the access should be provided only to the service provider’s server. Presently its not possible to release the attributes after some kind of authentication, that means if all the parameters are correct then IdP will response with the attributes even if the user has never log-in to access any service via the service provider. Attributes are released even if there is no active session of the user with the service provider. This is a drawback of the extension from user’s privacy perspective.
### 4.6. Group manager

Group manager is a PHP script assigned with the task to handle incoming group information and authorizing services to users. I have developed the group manager in PHP but there is no restriction on the chosen language for implementation. This section describes the main design concepts and criteria of a group manager plug-in, which is hosted by the service provider’s organization. Group manager doesn’t integrates with the shibboleth SP but works as a separate entity in collaboration with shibboleth SP. Following are the use cases being handled by the group manager plug-in:

**Group management:** Group management is the process of adding or removing groups to facilitate shared service authorization for group members. Figure 4.2 shows the flow of processes involved in group management. After a user logs in he/she can try to access the panel for group management. If the user is not an admin then he/she can only see the list of group names already imported and the user is a member of.

![Flow chart for group management](image)

Figure 4.2.: Flow chart for group management
If the user is an admin then access is granted to the group management panel. This panel could be used to add or delete groups into the database stores by the service. The added groups will allow members to access a shared service. The panel displays a list of groups available for addition and a list of groups already added. The lists already added could be deleted. Only those group names can be added for which the user is an administrator. The group name is accompanied by an additional attribute which identifies the group administrator. This parameter is used to displaying the group names in import list. Further attributes could be added into group information to identify the source of the group name, e.g. mailman, LDAP, etc.

Service provider receives group information on a per user basis and not on a per group basis. This means the group information will come as a binding for respective users. My solution doesn’t shares individual groups and the list of its members. Group sharing on a per user basis is faster as whenever group members are updated within a group, there is no need to send the updated member list to shibboleth SP. Moreover, its not mandatory to store the group names the user is a member of. We need to store only those group names which have been imported and access can be granted if there is a match between users group information attribute and stored group names. The access management has been discussed in next use case.

It’s very much possible that the imported group names are not unique among inter or intra organization level. Therefore, some mechanism needs to be in place to distinguish the group names. Several mechanisms are possible to resolve this issue, I have listed two possibilities to resolve the issue, one at shibboleth IdP and the second at the service. The first method involves modification of attribute definition at IdP in attribute-resolver.xml file. Additional unique parameter is prefixed or suffixed to the group name using a ResolverScopedAttributeDefinition element. A scoped attribute definition creates an attribute from the values of another attribute definition or data connector combined with a specified scope value. The prefix or suffix added to the group information could be a UUID. The second method works at service where it prefix/suffix the group name with the domain name of the user. This method expects the group names to be unique within an organization.

**Access management:** Access management handles the access grant/revoke for shared services. Figure 4.3 shows the flow of processes involved in access management. Once the user logins he/she can proceed to access a shared service available to the members of a specific group. When the user tries to access the service the group manager checks if the available group list in the local database has a match with the user’s group information attribute provided by shibboleth IdP. If there is a match user can proceed to access the service.

---

15https://wiki.shibboleth.net/confluence/display/SHIB2/ResolverScopedAttributeDefinition
If there is no match between local group list stored in database and user’s group information attribute then the user is denied access to the service and he/she is prompted to contact the group admin responsible for the respective service’s authorization. At the same time a time-based PHP scheduler is running in the background which checks for updated attributes from Shibboleth IdP using the attribute synchronizer plug-in for respective user. The time period for the scheduler can be set by an administrator and it could vary for individual services. A normal service will have a longer cycle (24 hours) while a more important service will have a shorter cycle (1 hour) between attributes are synchronized.

After the receiving updated attributes the group manager again checks for matching group names between local group list stored in database and user’s group information attribute. If there is a match then a user will be provided access to the service or an existing user’s access will be continued. If the access is denied then SLO mechanism of Shibboleth designed in section 4.4 is initiated to revoke access from any service authorized by membership to a group.

Finally, a user can logout voluntarily or a session might terminate at service/SP/IdP level which will logout the user from the service and force him to login again to access...
either the group management panel or the shared service controlled by group membership. Local storage of group information could be realized using MySQL database.
5. Testing group sharing

This section contains information related to performance testing of the group sharing architecture. Precisely, it contains information related to response time for two modules. For login test, I have used the IdPUnsolicitedSSO mechanism. This is an IdP originated login process, which means there is no redirection from SP to IdP, instead a dedicated URL belonging to IdP is called to initiate login process. Below I have shown the URL pattern for unsolicited login process:

https://{IdP Host}/idp/profile/SAML2/Unsolicited/SSO?
    providerId={SP’s entity ID}

IdPUnsolicitedSSO call takes the providerId as input parameter. providerId is the name of the service provider entitled to work with shibboleth IdP. The test file is executed on Apache JMeter\(^2\) and is available at shibboleth IdP wiki for version 3\(^3\). The test takes username/password from a list of 1000 users stored in a CSV file and runs 100 parallel threads looping with a delay of 500 milliseconds, trying to login into shibboleth IdP. A test is considered successful after receiving the SAMLResponse from IdP. Login test is sub-divided into two tests. The first part of the test logsins without any group management plug-in and the second part of the test logsins with mailman and UserFrosting plug-in returning group information. Delay in response time can be seen at the graphs below in fig 5.1. The complete test duration is of 60 seconds.

\(^1\)https://wiki.shibboleth.net/confluence/display/SHIB2/IdPUnsolicitedSSO
\(^2\)http://jmeter.apache.org/
\(^3\)https://wiki.shibboleth.net/confluence/display/IDP30/Load+Testing+Contributed+Results
The output shows an increase in response time due to the additional querying of group data by IdP to two different group management systems.

Second test result is related to the standalone response time of attribute synchronizer extension as shown in fig 5.2. Response time has been measured for 100 concurrent threads trying to access the attribute synchronizer extension to get updated attributes with a delay of 5 seconds between each call. The test duration is of 60 seconds. There is a constant delay of 500 milliseconds between for an individual thread. The test has two parts, one for attribute query without any group API’s and one test for attribute query with group API’s in shibboleth IdP. The results shows expected increase in response time due to increased operations to fetch group information at IdP.
Chapter 5. Testing group sharing

Due to lack of resources the tests were performed using a single IP address. That means all the requests have originated from the same IP address. Both the test scripts could be found at the Appendix B.

Figure 5.2.: Shibboleth SP attribute synchronizer response time
6. Preparations for production system

This section contains step by step instructions to implement the group sharing system in a production environment. Majority of time is spent in planning. Initial expectation is to have the required group management systems and shibboleth installed in the organization. The installation and planning could be divided between IdP’s organization and SP’s organization. All the necessary installation guide has been provided in the Appendix A to aid a developer.

The IdP only requires API development. Foremost requirement is to plan the input and output arguments as well as the operation the API must be able to perform. Afterwards API development could be initiated. Each API development including planning could vary from 1-4 weeks depending on the complexity of the operations it could perform.

Rest of the modules are required in the SP’s organization. Three solutions has been provided for implementing SLO. The best one needs to be selected, customized and developed in accordance to organizational needs. Using SLO could be avoided in case all the services are under a single parent domain, in which case session cookie will handle access revocation. The SLO implementation will require 2-4 weeks for implementation and testing. The attribute synchronizer extension requires re-installation of shibboleth SP. Post installation configuration are straight forwards and requires very less time and effort. The JSON based response from the extension is supplied to the next and final module of SP, the group manager. The attribute synchronizer extension will require 1 week of time but could extend for another week in case of any issues faced during shibboleth SP source build installation. Implementing group manager will vary between different service providers as not every organization will handle the attributes in a similar fashion. Planning and implementation of group manager will require 3-6 weeks depending on the complexity of the tasks to be handled by group manager.

Approximately 3 months time should be sufficient to bring the system into production. The time from development to production will also depend on the experience level of the user with development for shibboleth. Some organizations might already have a SLO in place working in production. Such an organization would be able to save a couple of weeks in bringing the system to production.
7. Similar work

Grouper\textsuperscript{1} is a tool offered by Internet\textsuperscript{2} the organization behind the development of shibboleth. Core functionality of grouper is to facilitate collaboration within an organization by allowing users to create groups themselves and administrate them. This group information is stored within Grouper and can be used to authorize multiple systems like email list, calendar group, web site, and so on. An application needs to have an API connecting with grouper to fetch the relevant group information for user authorization. Grouper behaves as a centralized group information storage system. Any modification in group information will be propagated to all other dependent systems. This functionality is different from my work in which each system acts as a standalone group information provider. Figure 7.1 shows a system based on grouper and shibboleth to provide federated identity management with group sharing. Every application including shibboleth is dependent on grouper to retrieve group information.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{grouper_based_federated_group_sharing.png}
\caption{Grouper based federated group sharing}
\end{figure}

Using grouper delegates group administration to individual group managers, thus re-
lieves workload from system administrators. Moreover, automatic role management is addressed in multiple services due to the centrally available group information provided by grouper. Grouper can be integrated with shibboleth as well. An API could be designed to retrieve group information from grouper. It is also possible to directly access group information from LDAP.

Similarity between Grouper and my solution is that, both allows local users to manage groups and share this information with service providers for authorization purposes. Difference between Grouper and my work is that, Grouper acts as a centralized group information provider whereas my solution assembles multiple group management systems to provide the required data.
8. Conclusion

In this master thesis I have proposed a solution using which self created group information could be shared among federated organizations for service authorization. Most of the services control authorization themselves based on parameters set by system administrators. A solution to minimize the burden on system administrators by allowing users to create group and authorize services based on group membership has been proposed. The architecture for sharing group information using Shibboleth has been provided with detailed analysis of core modules and requirement analysis. The core modules like API designing, SLO, group attribute synchronizer and group manager has been discussed thoroughly in order to facilitate designing such a system for real production environment. Testing has been done to identify the performance of the group attribute synchronizer as well as to identify the delay in login due to additional querying of group information by Shibboleth during authentication. Requirement analysis of the system has been provided to ease planning before initiating development.

Further work could be done to migrate the system to support Shibboleth IdP version 3.x. Currently due to lack of sufficient documentation for Shibboleth IdP version 3.x, the development and testing work has been restricted to Shibboleth IdP version 2.x.
References


References


A. Installation Guides

A.1. AttributeQuery handler plug-in shibboleth SP

The extension comes with precompiled *.so files\(^1\), these files could be directly placed in shibboleth SP for using the extension. But there is chance that the extension won’t work due to different versions of shibboleth SP and other softwares like Apache, etc. being used by different organization. The solution here is to build shibboleth SP from source and then build the extension by pointing to the installation path of shibboleth SP. Building the extension requires header files from shibboleth SP and its dependencies. A quick list of acceptably arguments to run configure on the extension can be obtained by running the command `.configure –help` inside the extensions source folder. It will enlist a long list of arguments of which the most important ones are, `–prefix` for providing designated installation directory and `–with-shibsp` for providing path where shibboleth SP is installed after source build. A detailed installation step for shibboleth SP source build in Linux can be found on the wiki page of shibboleth\(^2\).

Post installation the *.so files will be available inside shibboleth SP installation at the path /lib/shibboleth. The next step is to configure shibboleth SP to be able to access the extension. The code snippet A.1 shows the extension initiation in shibboleth SP. The OutOfProcess and InProcess blocks are placed at the beginning of the SPConfig in shibboleth2.xml file.

```xml
<SPConfig>
  <OutOfProcess>
    <Extensions>
      <Library path="attributequery-handler.so" fatal="true"/>
    </Extensions>
  </OutOfProcess>
  <InProcess>
    <Extensions>
      <Library path="attributequery-handler-lite.so" fatal="true"/>
    </Extensions>
  </InProcess>
</SPConfig>
```

\(^1\)https://bitbucket.org/PEOFIAMP/shibsp-plugin-attributequery-handler/downloads
\(^2\)https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPLinuxSourceBuild
Appendix A. Installation Guides

Next step is to invoke a `handler` element in shibboleth SP’s configuration file `shibboleth2.xml`. Code snippet A.2 shows a typical setup for invoking the `handler` element in shibboleth configuration file.

```xml
<Handler type="AttributeQuery" Location="/AttributeQuery"
  acl="127.0.0.1 ::1 x.x.x.x ./32" />
```

Code Snippet A.2: AttributeQuery handler invocation

Typical configuration of the `handler` element requires the type, Location and acl attributes. While the first two attributes remains the same, acl or access control list contains the IP addresses having the access right to invoke the extension remotely to get the response. The acl takes a CIDR block for allowing access to an IP address.

### A.2. Shibboleth 2.x Installation

#### A.2.1. Basic installation

The complete Shibboleth installation requires the installation of Shibboleth identity provider and Shibboleth service provider. Shibboleth installation encapsulates the installation process followed by configuration. The installation process generates the IDP’s entity id, IDP’s initial metadata, the key pair with self-signed certificate and IDP configuration files. Prior to installation the SSL certificate (used to secure IDP’s browser) and source of SAML metadata for service provider is required.

#### A.2.1.1. System requirements

<table>
<thead>
<tr>
<th>Product</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.5 &amp; Higher</td>
</tr>
<tr>
<td>Apache tomcat server</td>
<td>6.0 &amp; Higher</td>
</tr>
<tr>
<td>Apache web server</td>
<td>2.2 &amp; Higher</td>
</tr>
</tbody>
</table>

Table A.1.: System Requirements for Shibboleth Installation

3https://wiki.shibboleth.net/confluence/display/SHIB2/IdPInstall
The basic requirements of system for IDP installation are:

A.2.2. Shibboleth Identity provider Installation & Configuration

Installation begins by setting up Apache tomcat 8.0 server available in Apache tomcat’s official website. I am using the Apache web server installed in WAMP server (2.5). The Shibboleth IDP is downloaded from Shibboleth website\(^4\) Once the IDP zip file is downloaded the file is unzipped and install.bat file is executed for windows. The installation path is set to “path/to/opt/Shibboleth-idp” and domain name is set to “127.0.0.1” i.e. localhost.

The configuration of Apache webserver requires the addition of “ProxyPass /IDP/ ajp://localhost:8009/IDP/” line in “HTTPd.conf” file. This is done to pass the IDP requests from HTTP to tomcat. We are doing username/password based authentication so, the user’s credential information on LDAP server. The Apache tomcat server configuration requires to add “<Connector port='8009' enableLookups=false redirectPort='8443' protocol='AJP/1.3' request.tomcatAuthentication=false address='127.0.0.1' / >” to ’/usr/local/tomcat/conf/server.xml’ file. This is done to send usernames to IDP and the stop tomcat server itself from authenticating the user.

The Shibboleth IDP installation generates the war file which is deployed in Apache tomcat server once above configuration is done. To deploy war file one can either paste it in “apache-tomcat-8.0.17\webapps” and delete the pre-existing IDP folder and war file. The other method could be to give the path of the war file to be deployed in “apache-tomcat-8.0.17\conf\Catalina\localhost\IDP .xml” file.

A.2.3. Shibboleth Identity provider source code installation:

The above installation was done to understand the basic working and ingredients of Shibboleth. For development of the plug-in, the Shibboleth identity provider source code installation is required. The Shibboleth source code is downloaded with SVN\(^5\). The source code is the maven project\(^6\). To run the IDP in eclipse the installation of checkstyle plug-in and TestNG framework is required. The latest Eclipse luna contains maven m2E. The jdk version 1.7 is used in our project. To import the project the file menu is opened in eclipse and the following sequence of procedure is done “import->existing maven project->select the downloaded project->finish”\(^7\). Once the project is imported it can be build using maven clean followed by maven install. The maven install for Shibboleth 2.4.3 results in jar file in target folder.

\(^4\)http://shibboleth.net/downloads/identity-provider/2.4.3/ shibboleth-identityprovider-2.4.3-bin.zip
\(^5\)https://svn.shibboleth.net/java-shib-idp2/tags/2.4.3/
\(^6\)https://svn.shibboleth.net/java-shib-idp2/tags/2.4.3/
Appendix A. Installation Guides

Post installation the following configuration files needs to be changed present at `{path/to/IdP/installation}/conf`:

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>relying-party.xml</td>
<td>Add <code>&lt;metadata:MetadataProvider&gt;</code> with IdP’s and SP’s metadata.xml files path</td>
</tr>
<tr>
<td>login.config</td>
<td>Add LDAP connection parameters</td>
</tr>
<tr>
<td>logging.xml</td>
<td>Change logging level as per requirement</td>
</tr>
<tr>
<td>handler.xml</td>
<td>Allow only <code>&lt;ph:LoginHandler xsi:type=&quot;ph:UsernamePassword&quot;&gt;</code> and <code>&lt;ph:LoginHandler xsi:type=&quot;ph:PreviousSession&quot;</code> to provide proper login page to user</td>
</tr>
<tr>
<td>attribute-resolver.xml</td>
<td>Add <code>&lt;resolver:DataConnector&gt;</code> and <code>&lt;resolver:AttributeDefinition&gt;</code> to query attributes and set them to release</td>
</tr>
<tr>
<td>attribute-filter.xml</td>
<td>Add <code>&lt;afp:AttributeRule&gt;</code> to filter attributes before release to SP</td>
</tr>
</tbody>
</table>

Table A.2.: Shibboleth IdP configuration changes

A.2.4. Problem while building in Shibboleth 2.4.3

The missing “tools.jar” file error occurred as the eclipse was pointing to JRE instead of JDK. To overcome this JDK path was set in “window->preferences->java->Installed Jre’s”.

A.2.5. Problem with resulting jar file in Shibboleth 2.4.3

For the deployment of IDP in tomcat server war file is required. Jar file does not contain all the dependencies and required files.

A.2.6. Analysis for solution of problem:

The jar file renaming to war file was done. The war file was not able to give the same results as expected.

The new dynamic web project was made (file->new->other->dynamic web project->project name->finish). The web project was converted to maven project (right click->configure->convert to maven project->packaging (war)->name->finish). Tried to add the entire source file in newly created project but it was complicated due to the structural differences.
The creation of war file requires specific structure of maven project. We tried to recreate the structure by swapping the files. When project were rebuilt there were numerous built errors.

We also tried to discover the possibility of conversion of maven project into dynamic web project and then again reconverting to maven project. This was thought because conversion of maven project to dynamic web project will restructure the files required for war build. After reading various documents we found that no proper procedure was described for this technique.

After reading various documents it was concluded that pom.xml is the file that is used for jar or build. Hence, we tried restructuring ‘pom.xml’ file. Pom.xml details were read and structure was understood.

We tried modifying packaging tag to pom.xml file. The jar was replaced by war.

```xml
<packaging>war</packaging>
```

Also plug-in structure was also changed from jar to war

```xml
<plug-in>
  <groupId>org.Apache.maven.plug-ins</groupId>
  <artifactId>maven-war-plug-in</artifactId>
</plug-in>
```

The maven build was done and war file was generated. When this war file was deployed in Apache tomcat server the IDP didn’t got start. When the tomcat logs were checked “java.io.FileNotFoundException: Could not open ServletContext resource [/$IDP_HOME$/conf/internal.xml]” error occurred. The digging of error led to information that there is some issue with web.xml file present in web-INF folder.

When the difference between pre-installed web.xml and presently generated web.xml was analysed using win merge it was

Pre-installed

```xml
<param-value>file:/f:/opt/Shibboleth-idp/conf/internal.xml;file:/f:/opt/Shibboleth-idp//conf/service.xml;</param-value>
```

present

```xml
$IDP_HOME$/conf/internal.xml; $IDP_HOME$/conf/service.xml;
```

The value was hardcoded and the war file was again deployed and it runs successfully in outside the browser but inside eclipse it still give 404 error.

### A.2.7. Service provider installation

SP comes in binary format for Windows operating system which could be installed using windows installer. In Linux systems, SP could be installed using `apt-get install`
Appendix A. Installation Guides

libapache2-mod-shib2 or it could be built from source. Following installation steps has been copied from shibboleth official wiki available at [https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPLinuxSourceBuild](https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPLinuxSourceBuild).

The following packages must be built in the following order using the `./configure` commands listed. Unless specific version notes are given, any new release is sufficient. Be sure to `make` and `make install` as appropriate for each package, including Shibboleth itself. You will also need the Boost headers available (but don’t need to actually build or install the full Boost library set). Due to a bug not yet fixed in a release, only versions up to 1.52 will work[28].

- **log4shib**: `./configure --disable-static --disable-doxygen --prefix=/opt/shibboleth-sp`
- **Xerces-C**: `./configure --prefix=/opt/shibboleth-sp --disable-netaccessor-libcurl`
- **XML-Security-C**: `./configure --without-xalan --disable-static --prefix=/opt/shibboleth-sp`
- **XMLTooling-C**: `./configure --with-log4shib=/opt/shibboleth-sp --prefix=/opt/shibboleth-sp -C`
- **OpenSAML-C**: `./configure --with-log4shib=/opt/shibboleth-sp --prefix=/opt/shibboleth-sp -C`

When building Shibboleth, you can usually rely on the configure script to detect your Apache version and do the right thing, but in special cases, you can pass various options to control the Apache module version and locate the apxs script to use.

- **Shibboleth**: `./configure --with-log4shib=/opt/shibboleth-sp --enable-apache-24 --with-apxs2=/usr/local/apache2/bin/apxs --prefix=/opt/shibboleth-sp`

Here `/opt/shibboleth-sp` could be any path where shibboleth SP should be installed. Post installation the following configuration files needs to be changed present at `{path/to/SP/installation}/etc/shibboleth`:

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>shibboleth2.xml</td>
<td>Add <code>&lt;Host name=&quot;&quot;/&gt; of server, </code>&lt;ApplicationDefaults entityID=&quot;&quot;/&gt; of SP’s URL, <code>&lt;SSO entityID=&quot;&quot;/&gt; of IdP’s URL and </code>&lt;MetadataProvider&gt; for IdP</td>
</tr>
<tr>
<td>attribute-map.xml</td>
<td>Add `&lt;Attribute&gt;coming from IdP before passing on to service</td>
</tr>
</tbody>
</table>

Table A.3.: Shibboleth IdP configuration changes

Finally generate the SP’s metadata by accessing the URL \{SP’s URL\}/Shibboleth.SSO/Metadata. Share this metadata file with shibboleth IdP.

The trouble shooting guide is very handy as it lists the most frequent issues faced by the users. The guide is available at https://wiki.shibboleth.net/confluence/display/SHIB2/Troubleshooting.

A.3. Mailman

Mailman can be installed in on Unix based systems. This guide is for installation on Ubuntu. First task is to install a mailserver. I have installed Postfix. Installation command is sudo apt-get install postfix. I have installed postfix with default configuration. My mailservers host was localhost.

Next step is to install mailman. The command is sudo-apt get install mailman. During installation an example Apache configuration file comes with mailman and is placed in /etc/mailman/apache.conf. In order for Apache to use the config file it needs to be copied to /etc/apache2/sites-enabled using the command sudo ln -s /etc/mailman/apache.conf /etc/apache2/sites-enabled/mailman. This will setup a new Apache virtualHost for the Mailman administration site. Once the symbolic link is created, you’ll need to enable it and restart Apache sudo /etc/init.d/apache2 restart.

Now we can add new lists into mailman and users to them for sharing their group information.

---

8http://www.postfix.org/
9https://help.ubuntu.com/community/Mailman