Research and development approach on the enhancement of e-Catalunya platform

Master's Degree in Information Technology Project

Director: Rosa Mª Martín Santiago
Co-Director: Hector Puente
Tutor: Jose María Barceló Ordinas

Alba Coll
September 2008

To Rosa Mª Martín and Hector Puente for their support and their never ending patience.

To the e-Catalunya team for making me go to work every morning with a smile and for helping me every time the text processor revolted against me.

To my flat mates for forgiving my last two weeks cleaning tasks skipping

To Ada, Carles and Laura for being there.

To Ada for making our sleepless-, working-sessions nights a hilarious time.

To my parents for putting up with me in my most hysterical moments and for helping me with everything they can.

Index

1. Introduction

This chapter attempts to introduce the context and main objectives of this master thesis as well as the motivation that drove its development.

1.1. Background

This thesis has been developed in the LCFIB (Laboratori de Calcul de la Facultat d'Informàtica de Barcelona) inside the e-Catalunya project. e-Catalunya is a collaborative project between the Generalitat de Catalunya and the Barcelona School of Informatics, and is developed by the LCFIB (Laboratori de Càlcul de la facultat d'Informàtica de Barcelona). e-Catalunya is a web 2.0 platform where users dispose of a set of collaborative tools where they can share opinions and knowledge and can interact with each other. The platform is structured in portals where both professional and social groups form social communities.

The LCFIB was interested in the "new" federated identity technologies that have gained popularity in the last few years. Federation describes scenarios in which no single group or organization manages all users and resources in a distributed application environment. Federations allow different organizations to share user information between them so their users may have access to resources outside their own association. Inside a federation organizations may have different roles. Suppose a federation composed of two organizations: A and B. Organization A gives access to its resources to all users that have been authenticated either in A or B. Instead organization B does not share any of its resources and inside the federation model it only authenticates its users for organizations A.

Initially the federated model only attempted to enable Single Sign-On between applications of different organizations. Single Sign-On is a method that allows a user to authenticate only once and gain access to multiple applications inside a domain. What federations tried to accomplish initially was to implement a Single Sign-On with applications of different domains, thus allowing users to access to external applications without the need to authenticate again.

The LCFIB wished to do a research of the federation world for future references. Thus if needed at some point, the basic information about the federation model would be already gathered and analysed.

With all this muddle of concepts of federations and Single Sign-On in mind together with e-Catalunya and the new trends of the web, the e-Catalunya team

considered the possibility of enabling a federated identity management with the e-Catalunya platform. After all, e-Catalunya is a result of the Web 2.0 where the sharing of information and the user participation are the main goals. To federate e-Catalunya would mean to open its doors and allow other communities and organizations to interact together.

1.2. Project objectives

This project has two main goals. The first one is to provide to the LCFIB a general study about the federation identity model. This includes the basic concepts behind federations as well as the current specifications that define them and some of the open source products that implement them.

The second is to study the viability of federating the e-Catalunya platform. This is done with the development of a pilot test in which the e-Catalunya platform is prepared to form a single sign-on federation. The idea of the pilot test is not to build a federation itself, but to perform the necessary changes in the platform to enable a future federation.

1.3. Motivations

Nowadays Web 2.0 and social networks are in everyone's mouth. Few things remain of the original Word Wide Web. It seems like the objectives that Tim Berners-Lee had imagined when the web was born have finally been achieved. The Web is not anymore one way channel or a read only content repository; is about interaction, participation, the sharing of information. People around the world join virtual communities with other people that share the same interests or everyday activities. The ubiquity of the Internet has made numerous applications available online to the general public, from banking to investment to shopping to paying bills to playing music. The Web is not a technological concept anymore but a social phenomenon.

e-Catalunya is one of the Web 2.0 child's and it belongs to this world of sharing and participation. The possibility of federating the e-Catalunya platform with other entities offers rich expectations. e-Catalunya source is distributed under an open licence and so there are an infinity of possible federated identity models cases surrounding the platform.

Let's take some imaginary example. The e-Catalunya platform would be federated with the Technical University of Catalonia (UPC) and the Mercè Rodoreda Library. When a student logged inside the UPC web page or to the intranet he would automatically gain access to the Mercè Rodoreda library catalogue. Imagine this user that while consulting the catalogue discovers that the library is lending the Monty Python's Big Red Book. At that moment he could immediately go to the e-Catalunya "British TV and cinema" portal to let everyone know the news and where to find the book in a blog entry. Accessing the platform would be completely transparent to the user.

The federation model can have a great impact inside the e-Catalunya platform, opening it to universities, libraries and even with other social networks like facebook or MySpace. Users could gain new services and functionalities complementing the ones offered by the platform with no need of new passwords or the requirement to have to authenticate each time he leaves an organization service to access another.

Federated identity management can enhance the service offered from many organizations to their users. Therefore the study of federations may be really useful in the future for the LCFIB, not only in the e-Catalunya behalf, but for any other systems that may be considered being federated.

Figure 1 shows a fictional federation. The user belongs to one of the organizations that form the federation, but she may have access to some of the resources of any other organizations.

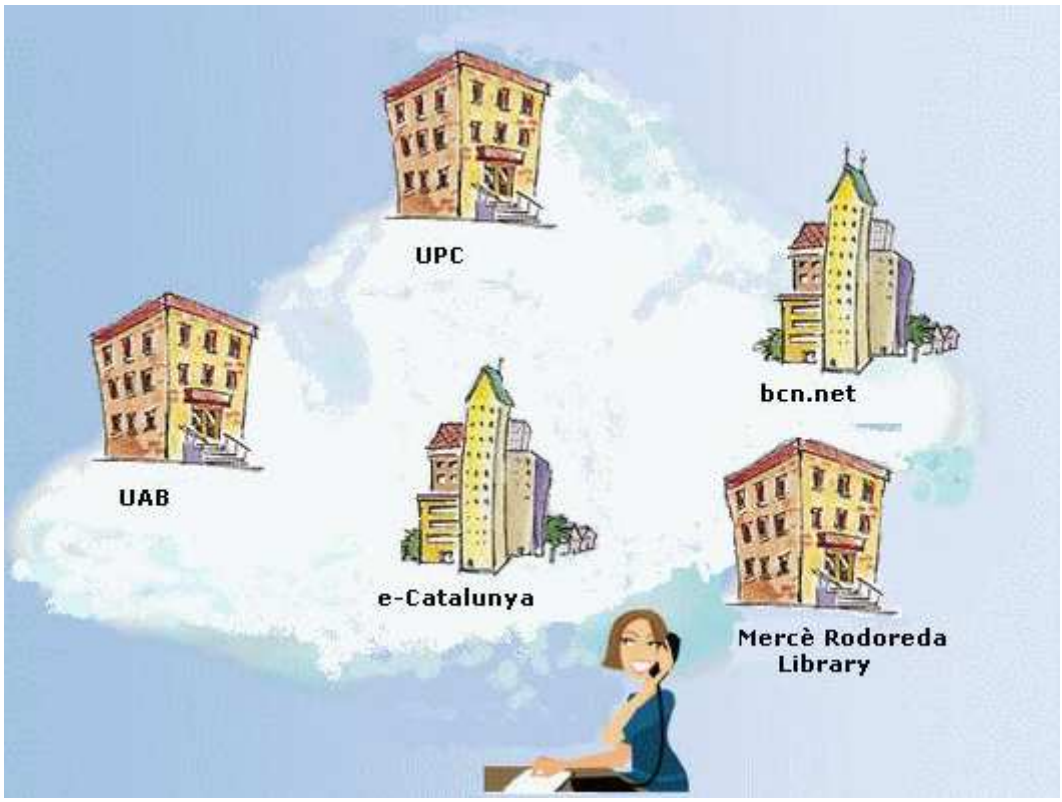

Figure 1: Federation prototype.

1.4. Work environment

As it was said in the lasts sections, this thesis takes places inside the LCFIB in the Technologic Projects Development Area (ADEPT). The e-Catalunya team is composed both with students doing scholarships and PAS (administration and service staff) members.

Once a week the e-Catalunya team meets to put in common what is every one doing, what has been achieved during the week, what does still remain to be done and to discuss for the best solutions. Since the beginning of this thesis nearly every week the team has discussed what the best federation specifications were for the e-Catalunya, what changes might be made, how the integration with the Shibboleth 2.0 middleware had to be done etc.

The 18th of June an internal meeting took place where all the Final Master Thesis that were being developed inside the LCFIB were briefly explained to all the LCFIB staff.

1.5. Document organization

This section explains in general terms how this document is structured and what the topics of each chapter are.

The current chapter introduces the subject of this thesis and explains which are its main goals and the motivation behind them.

The second chapter explains the e-Catalunya platform. e-Catalunya main functionalities and basic architecture are reviewed along with the software infrastructure that supports the platform.

The third chapter exposes the federation model structure more deeper and reviews the main concepts and technologies used in federated identity managements systems.

The fourth chapter presents a brief overview of the different specifications that define the federated model and its behaviour. The specifications reviwed are SAML 1.x, 2.0, ID-FF 1.1, 1.2, WS-Federation and OpenID. At the end these specifications are compared.

The fifth does a more in depth study of the specifications presented in the previous chapter.

The sixth chapter does a brief research of the current open source solutions that deploy federated identity management. The products reviewed are Lasso, ZXID, JBoss Federated SSO Framework, Higgins, OpenSSO, Shibboleth and PAPI.

The seventh chapter explains the firsts steps required before the deployment of a federated identity management system between different entities and describes how e-Catalunya is going to be integrated with Shibboleth 2.0.

The eighth chapter expands on Shibboleth 2.0 working and structure along with the explanation of its specific deployment in the e-Catalunya.

The ninth chapter covers the project costs, the initial time planning and its modifications and exposes possible future enhancements or modifications that could be done related to this thesis along with some personal conclusions.

The tenth chapter presents the bibliography used during the development of this thesis.

The appendix contains a glossary for this document.

2. The e-Catalunya Project

This Master Thesis takes place inside the e-Catalunya project. Before focusing on the topic, objectives and deployment of this thesis, the reader should be introduced to its context. Therefore this chapter exposes a brief introduction to the e-Catalunya platform, reviewing its main objectives, features and architecture.

2.1. Introduction

e-Catalunya is a collaborative project between the Generalitat de Catalunya and the Barcelona School of Informatics, and is developed by de LCFIB (Laboratori de Càlcul de la facultat d'Informàtica de Barcelona). e-Catalunya is a platform that offers a collaborative framework targeting both professional groups (doctors, teachers, etc.) and social groups that might need a common space to share and organize information. Its main purpose it's the creation of social, professional and knowledge networks between the users of the platform. e-Catalunya is an assemblage of different community portals that provide room for discussion, research and organization between its members, allowing them to generate, administrate and exchange knowledge and experiences on the net.

One of the premises in e-Catalunya platform is that all software used must be open source.

e-Catalunya is based in the new Web 2.0 technologies and it comprises web tools to share information such as wikis, forums, blogs etc. E-Catalunya is composed of portals which in turn may comprise as many groups as desired.

The platform is in constant development and nearly every three or four month a new version is released. With every new version new functionalities are implemented and bugs from the older versions are corrected.

Currently the e-Catalunya version in the production environment is 1.7, released last July, though this project was developed with e-Catalunya 1.6.

2.2. Portals and groups

As mentioned above, the e-Catalunya platform is composed of portals that may contain different groups. It is in these groups where users may work together and share information among them.

For an example we could have a portal called "British TV and cinema" with some groups such as "Monty Python", "Cambridge Footlights", "Fry & Laurie" and "Blackadder". It is inside these groups where users share knowledge and opinions through the web tools offered. Inside the "Fry & Laurie" group a user could share opinions, write critics or recommend some chapter of the TV series "A bit of Fry and Laurie" or "Jeeves and Wooster", while inside the "Blackadder" there could be information about new DVDs releases of the "The Blackadder" TV series.

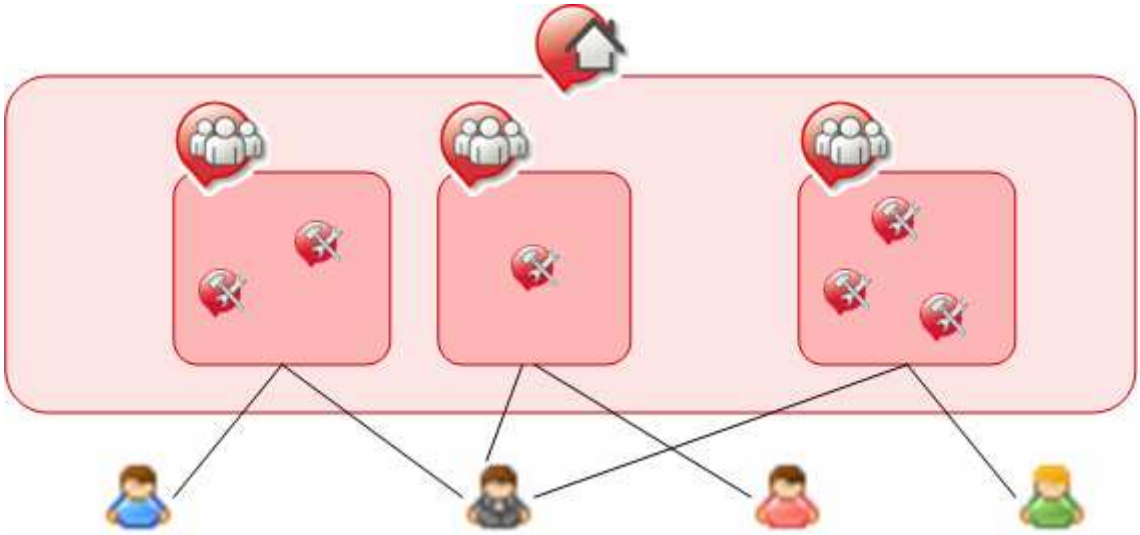

Figure 2: Example of a portal hosting three different groups and users interaction with them.

Both portals and groups can be public or private. A public portal is visible and accessible for all users of the platform regardless their memberships or if their logged or their anonymous users. In contrast a private portal is only visible and therefore accessible by its own members.

A public group is visible for any user, just like the public portal, if, and only if, it belongs to a public portal. On the contrary, if the public group belongs to a private portal it is only visible for the portal members. In case of a private group this one is only visible and accessible for its group members, and other portal members that do not belong to this group will not be able to access to it.

Also, groups have other properties like the formality level of the group (formal or informal) or the time duration (temporary or indefinite), which define whether group activity is meant to finish at a certain known date or not.

2.3. User Roles

The e-Catalunya platform is a complex framework that offers lots of functionalities and features. In order to manage the behaviour and features of the portals/groups/tools some especial users with privileges are needed.

In e-Catalunya the following roles are defined:

- **Member:** any user that belongs to a portal or a group. They are able to access any content inside the group they belong to.
- **Moderator:** members that have a moderation role. They can perform the same activities as a member and they can edit, delete or publish content as well.
- **Administrator:** administrators can manage all the resources of the system/portal/group they administrate. They can create new users, new tools etc.

A fourth role may be considered with those users that participate without being registered or logged. This is the **Guest** role, and users with this role may only access to the public content of public portals.

2.4. Infrastructure Software

e-Catalunya requires for its working a set of infrastructure software that composes the base where the platform is built. Following there is a brief overview of each of these components.

2.4.1. Jakarta Tomcat

Most of e-Catalunya, the parts developed by the LCFIB as well as the open source software used, is developed in Java. Therefore e-Catalunya requires an

application server that allows the execution of Java code, such as Tomcat. Tomcat is responsible to serve the dynamic content of the platform.

2.4.2. Apache and PHP

Apache is a web server that, in contrast to Tomcat, is very efficient when serving static content. Thus, the main reason of having an Apache + Tomcat architecture is the optimization of request serving.

Moreover Apache has a large set of modules that enable the serving of other types of content, such as perl, php etc. One of the e-Catalunya tools, explained below, is the forum which is implemented by the phpBB2 software. Therefore the php apache module is required for the e-Catalunya forums

2.4.3. Apache Tomcat Connector

All petitions sent to the e-Catalunya platform go to the Apache server first. Therefore Apache needs to communicate with Tomcat to pass those petitions that Tomcat needs to process. To enable this communication between both servers it is required a connector Apache-Tomcat. e-Catalunya uses mod_jk. Apache knows the petitions that need to be sent to Tomcat through a configuration file.

2.4.4. MySQL

e-Catalunya requires a storage system to store and manage a large amount of information. MySQL is a simple and optimal Data Base Management System (DBMS) that consumes few resources. MySQL is one of the most DBMS used nowadays due to its free license and it simplicity.

2.4.5. OpenLDAP

OpenLDAP is a free implementation of the Lightweight Directory Access Protocol (LDAP) protocol. LDAP is an application protocol for querying and modifying directory services running over TCP/IP. It is very common to use it for storing user related information, such as usernames passwords, telephone numbers etc. since it is more optimal than a relational data base with data that can be easily structured in a hierarchical way.

e-Catalunya stores all the data related to the user authentication in the OpenLDAP server.

2.4.6. Sympa

e-Catalunya creates mailing lists for each group in the platform, some with all its members and others only with its administrators and moderators. Sympa is a perl mailing list software that enables the creation of mailing lists through schemas and it is highly configurable.

2.4.7. Postfix

e-Catalunya offers a notification email service. Due to Sympa characteristics it is necessary to have a mail server for this notification service. This email server is the responsible of sending all notification messages. The server used is Postfix since is one of the more popular mail servers and nearly all Linux distributions have it by default.

2.5. e-Catalunya Authentication

e-Catalunya is composed both with tools developed by the e-Catalunya developer team and with external open source software. Two of these external open source softwares are the phpBB2, which implement the e-Catalunya forums, and the Mantis Bug Tracker, implementing the e-Catalunya Bug Tracker. Both of them have already an authentication system and a database of their own. That means that if a user that already been authenticated in the platform wanted to access one of these tools he should need to authenticate again, which probably would make him a "not a very happy user". The user should authenticate three times in order to have access to all the functionalities offered by the platform.

It's for that reason that e-Catalunya has a Single Sign-On system. Single Sign On allows users to authenticate only once and gain access to resources of multiple systems. The system chosen for the platform was JOSSO, Java Open Single Sign-On. JOSSO is the one responsible of the authentication of the platform users against the OpenLDAP.

2.6. Tools

Until 1.6 e-Catalunya version, tools were only assignable to groups as the original idea was that the work environment was centred in groups and not in portals. Since 1.6 all tools except mailing lists may be assigned at portal level. Hence users may interact and share information with other portal users that belong to different groups. Following the "British TV and cinema" portal example, there could be spaces where all portal users could discuss about British performing techniques or British cinema history and evolution.

Just like groups and portals, tools may be public or private. Public tools are visible and accessible for those who can access and visualize the group or portal where the tool is. A private tool is only visible and accessible for group members, in case of a group tool, or for portal members, in case of a portal tool.

The next tools are the available ones in e-Catalunya:

2.6.1. Wiki

A wiki is a writing collaborative web based tool that comprises the collective work of many authors. A wiki allows editing, deleting or modifying content that has been placed on the Web site, including the work of previous authors.

In e-Catalunya 1.6 wikis were implemented using the XWiki software; however in e-Catalunya 1.7 the Xwiki software was substituted for a new wiki software developed inside the LCFIB by Lluís Sunyol.

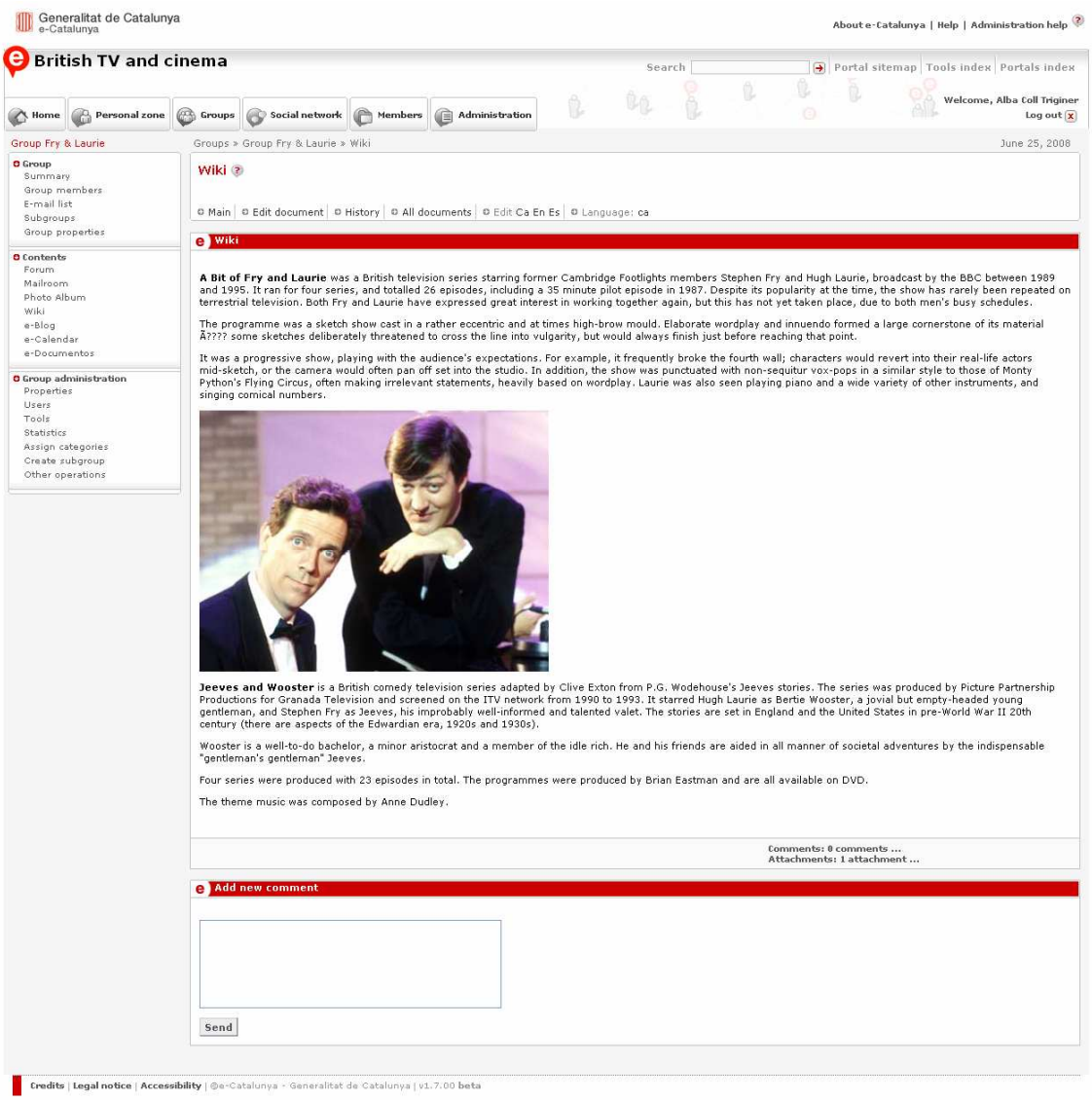

Figure 3: Example of a wiki

2.6.2. Blog

A blog is a web based tool with regular entries of commentary, descriptions of events, or other material. These entries are typically displayed in reverse chronological order. Users may write comments to any blog entry.

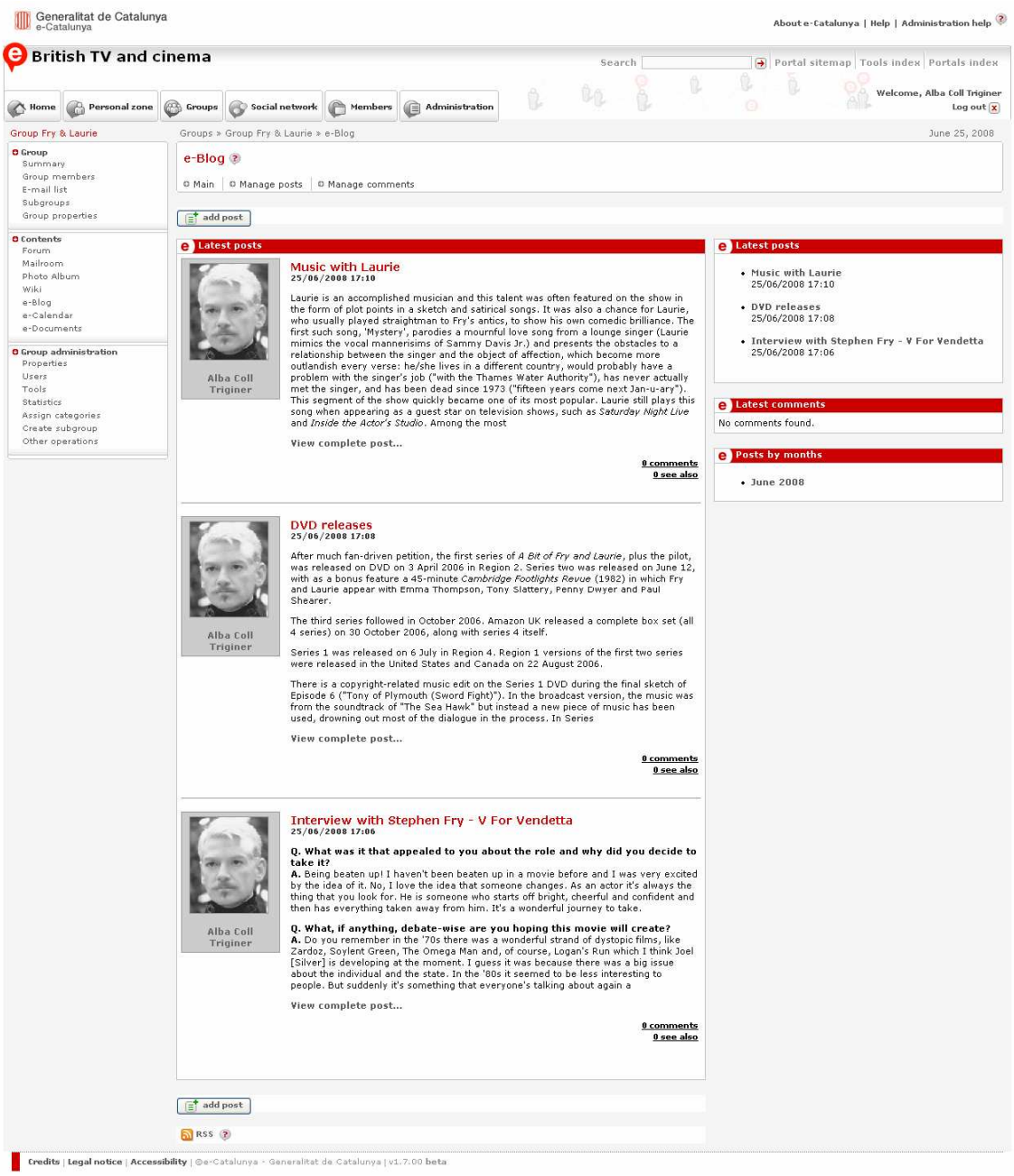

Figure 4: Example of a Blog

2.6.3. Calendar

The calendar tool is a interactive calendar where users may insert activities, joint activities and consult where this activities take place using links to the Institut Cartogràfig de Catalunya.

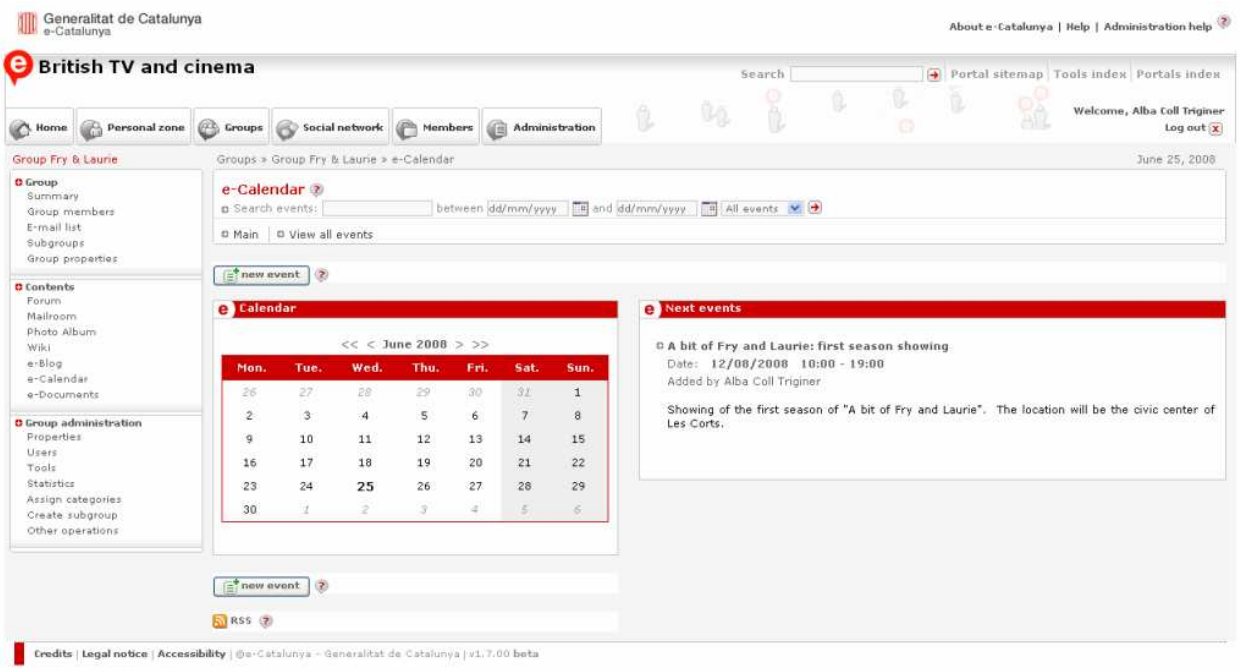

Figure 5: Example of a calendar

2.6.4. Repository

The repository tool creates a virtual space where users can share files between them. It simulates the file system of an operative system and allows the users to create folders and build a hierarchic structure.

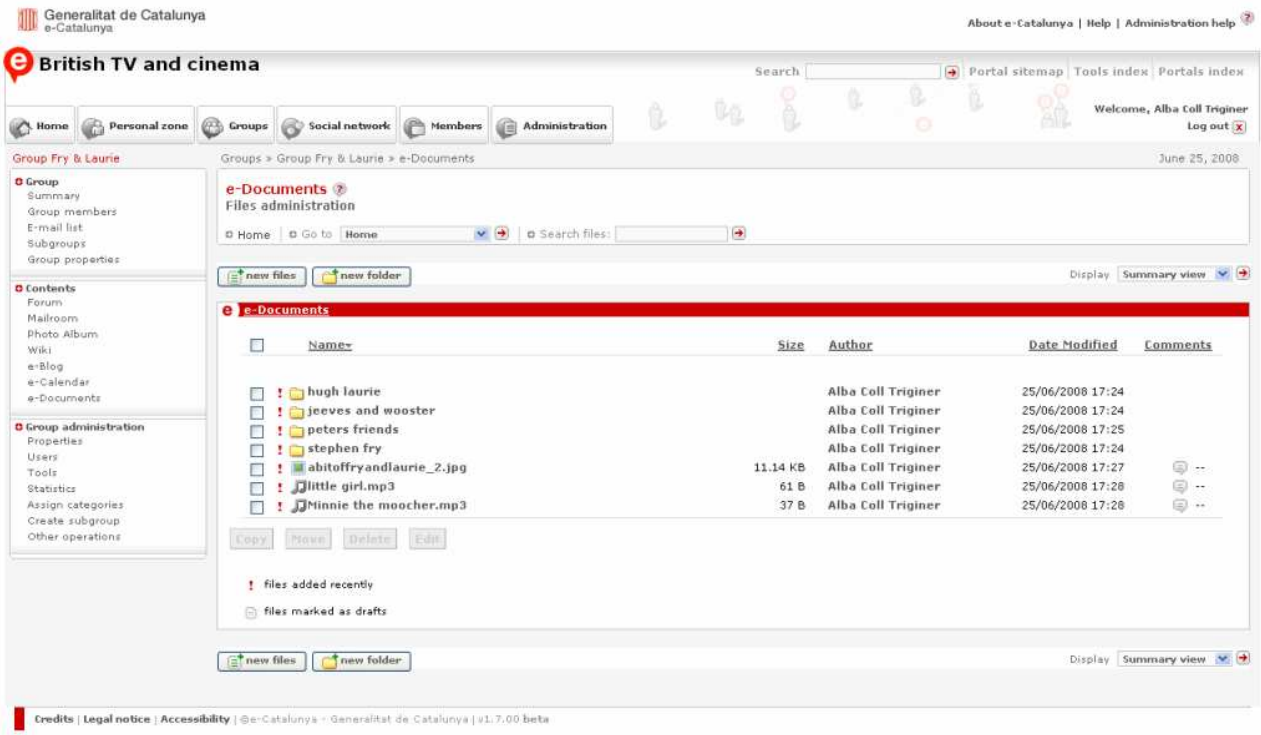

Figure 6: Example of a repository

2.6.5. Photo Album

The Photo Album tool allows the users to share picture and images between them. Just like in the repository, in the photo album folders can be created and organised in a hierarchic structure. Users can comment every picture stored in the photo album.

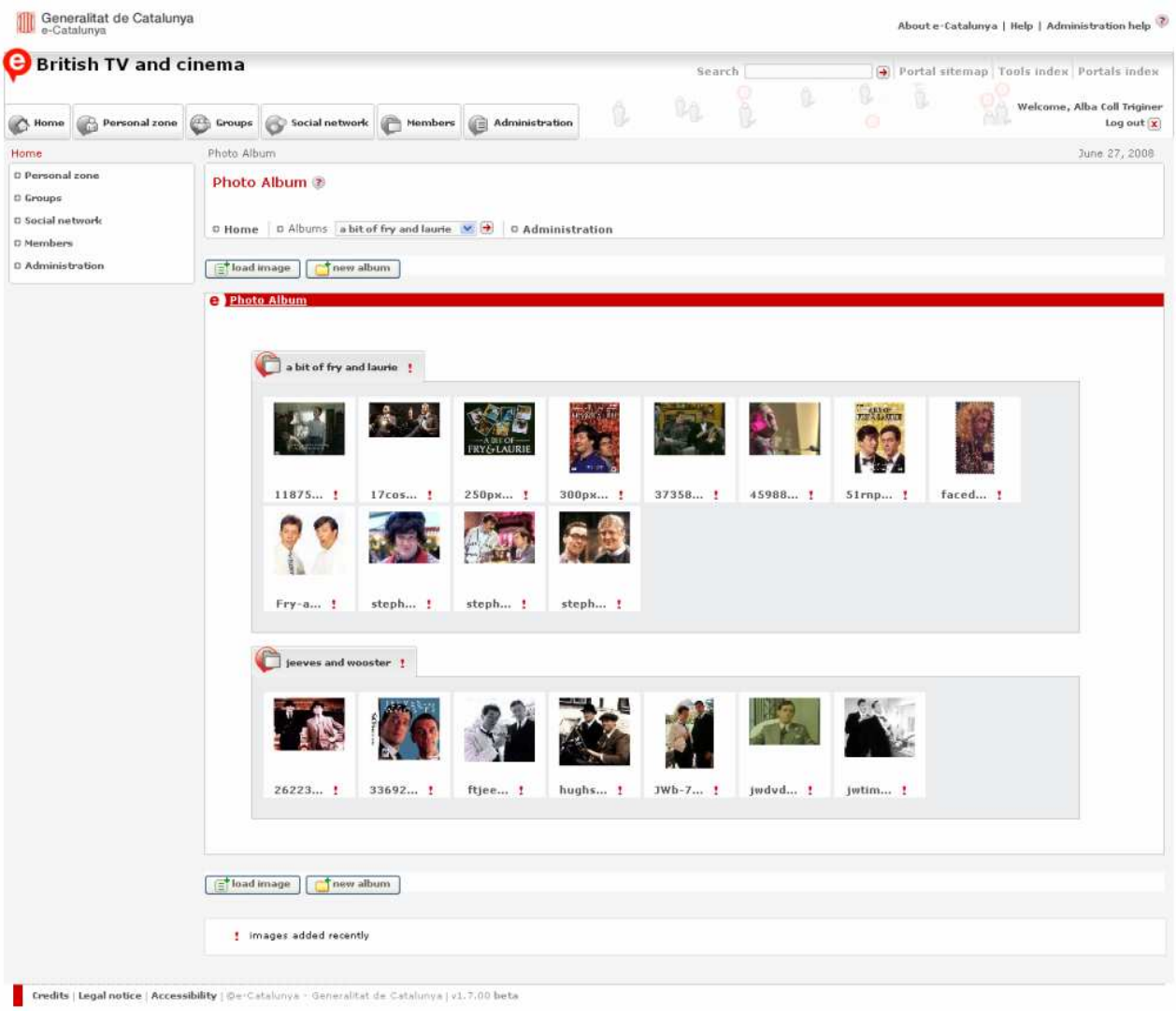

Figure 7: Example of Photo Album

2.6.6. Mailroom

The Mailroom is a questionnaires tool normally used to do opinion polls. The tool provides means to create questionnaires and to extract its results easily.

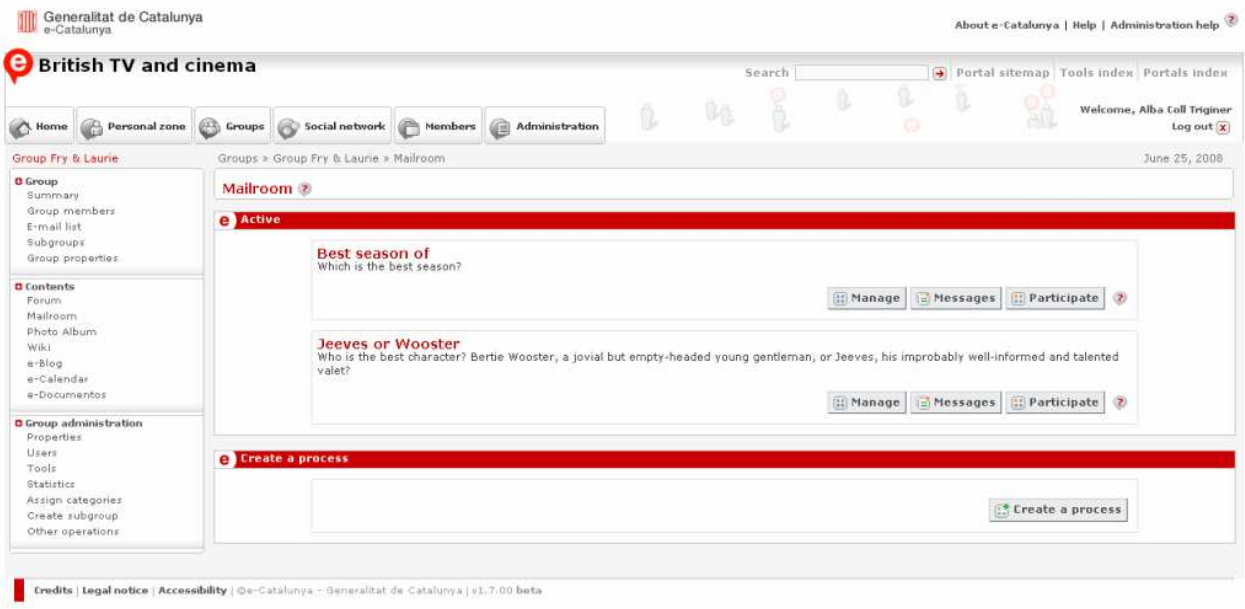

Figure 8: Example of a mailroom

2.6.7. Mailing list

Each group has two mailing lists by default: one that allows all users of the group to sent mails and the other that only administrators and moderators may use it to send mails. A mail sent in any of the two lists will arrive to all the members of group. No more mailing lists tools can be assigned to a group.

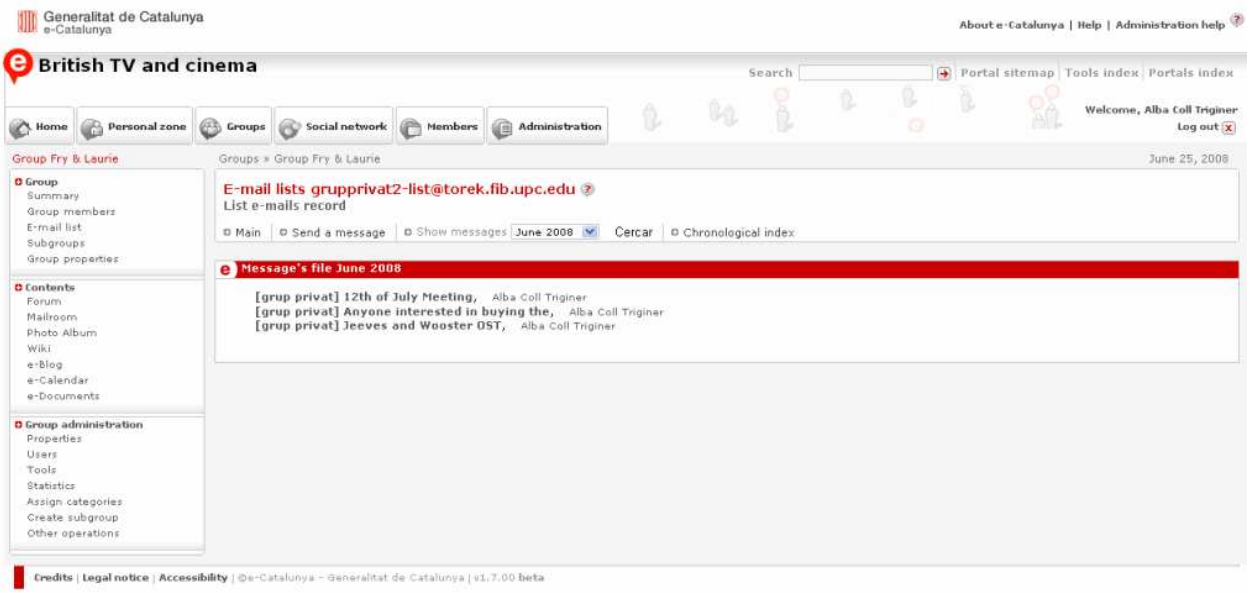

Figure 9: Example of a mailing list

2.6.8. Forum

A forum is a web based tool that holds discussions. It allows the users to share comments and experiences and to make questions that anyone may answer.

The forums in e-Catalunya are implemented using the phpBB2 software (php Bulletin Board version two).

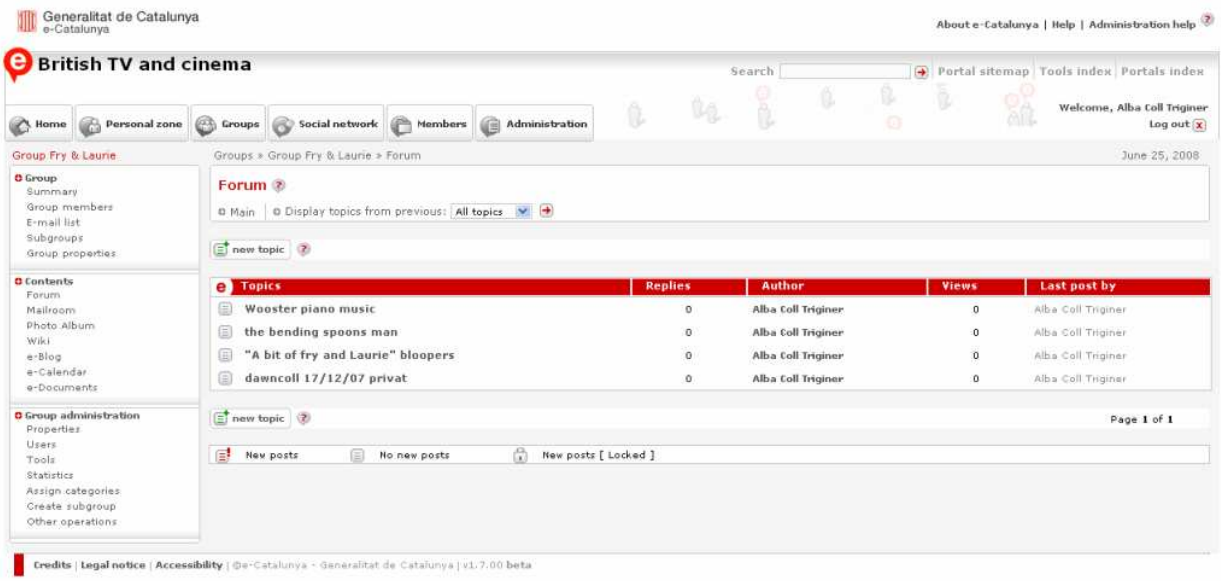

Figure 10: Example of a forum

2.6.9. eBugTracker

Through a bug tracker users may report platform incidents to their portal and group administrators in a simple and organized way.

The eBugTracker is implemented using the Mantis BugTracker software and it is only available in the last version of the platform. Therefore this thesis has not contemplated its presence in the platform since the version used was the 1.6 instead of the 1.7.

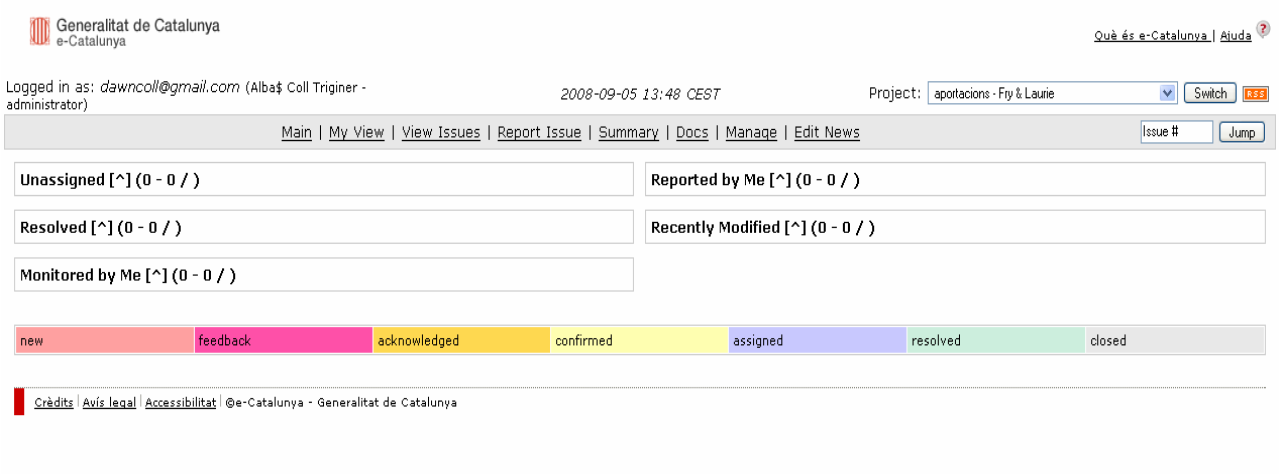

Figure 11: Example of bug tracker

2.7. Social and Knowledge Network

In any portal users have available a personal space where, among other things, they can store their contacts of other members of the portal.

The social network is the graphic representation of a user and all his contacts. Any of the user contacts may have more contacts of their own that can be visualized as well building a network of relationships.

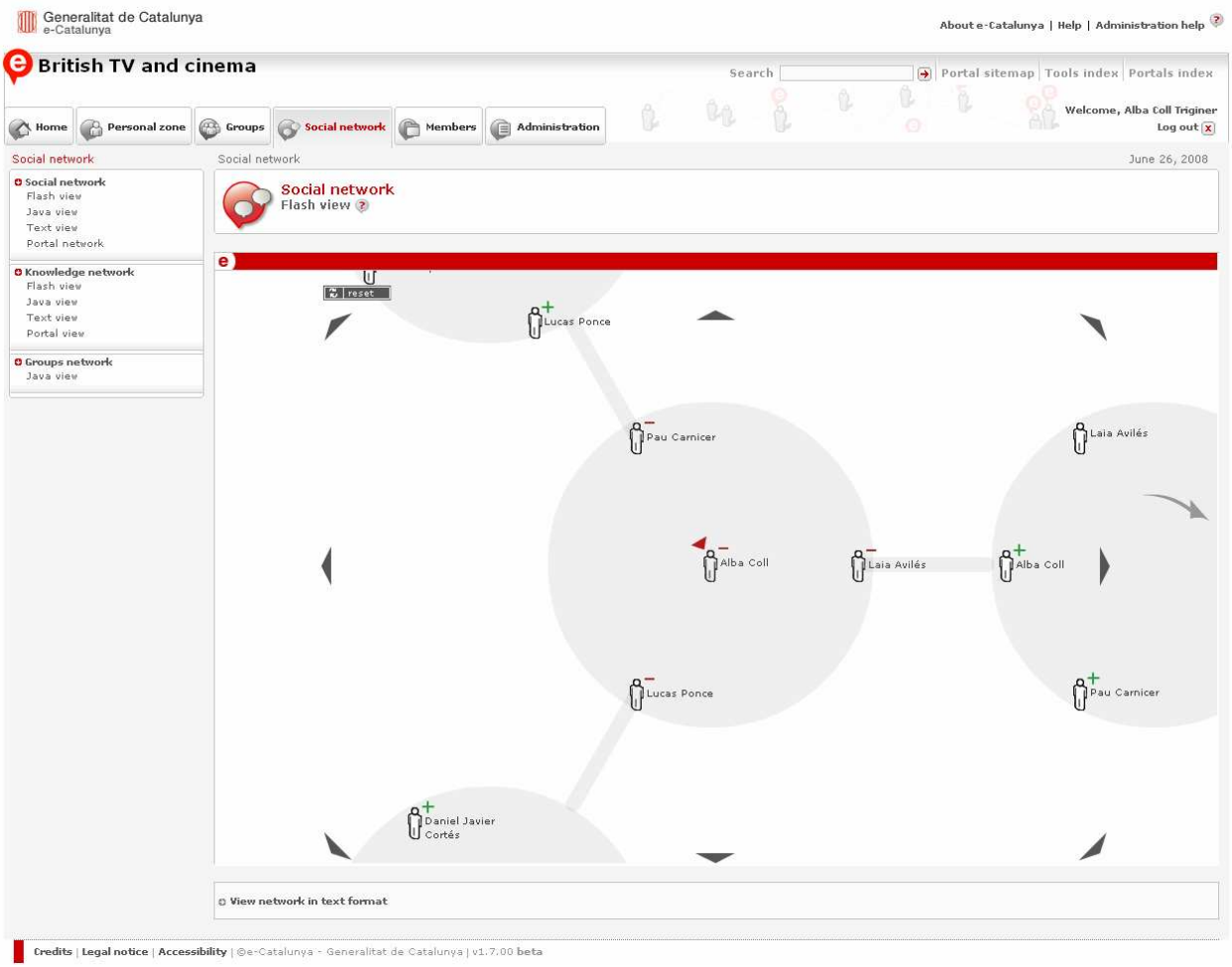

Figure 12: Example of a social network

While the social network is built using the user contacts, the knowledge network is built with the user activities performed inside the e-Catalunya platform. So the network shows closer to the user the portal members that use the same tools, read the same documents… in short, those members that have more affinity to the user. The more affinity a member has with the user, the closer this member is shown to the user.

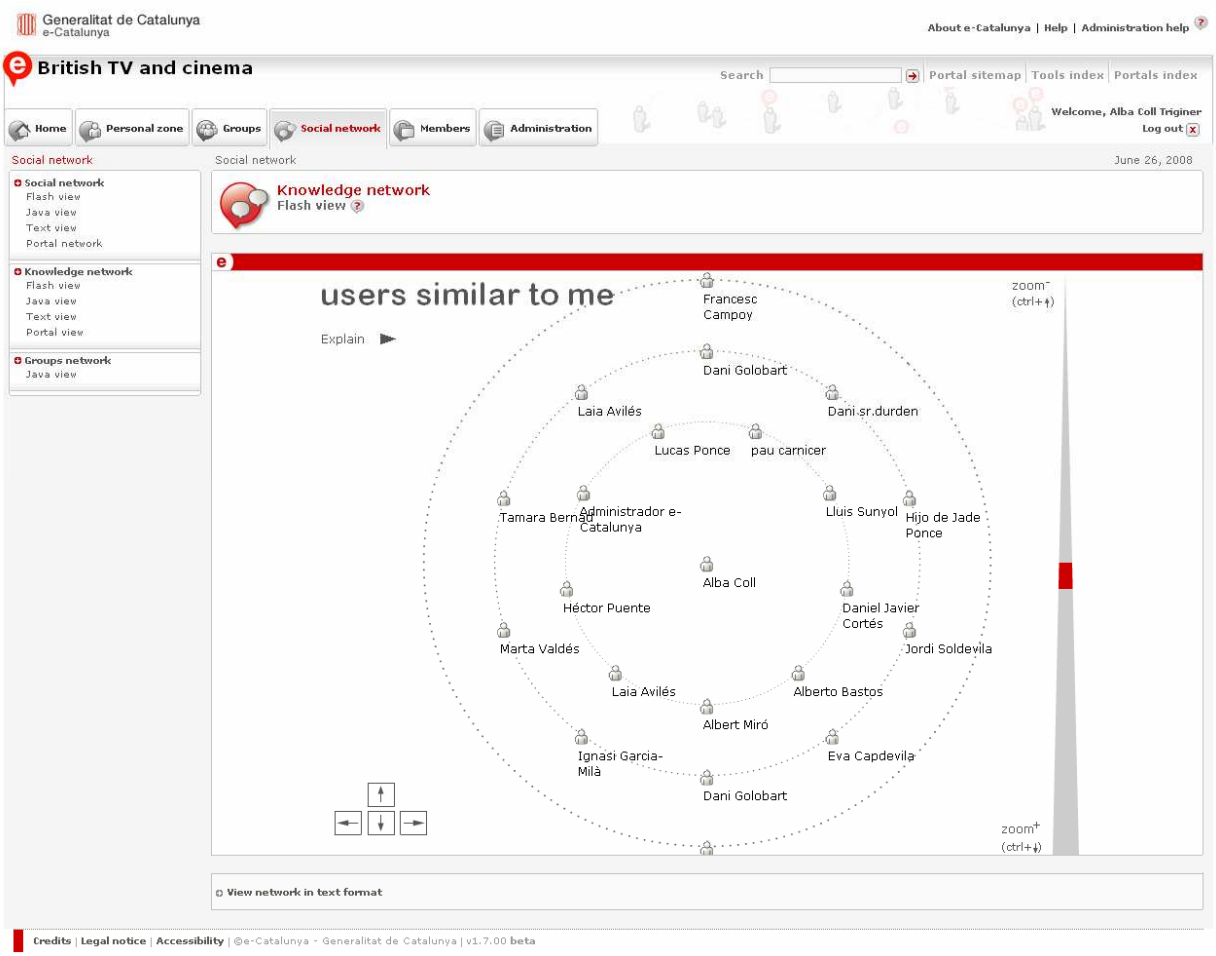

Figure 13: Example of a Knowledge network

With the knowledge network the following aspects are deduced:

- **Similar users:** users that share interests with the user.
- **Recommended documents:** the documents consulted by the similar users are recommended by the platform.
- **Related documents:** through a document consulted by a user, other documents seen by him can be discovered.

2.8. Administration Panel

This panel is only accessible for portal and system administrators. Through it administrators can manage the portal and group resources as well as users.

Also there is a statistic section where some activity information can be consulted to see what users do in the portal etc.

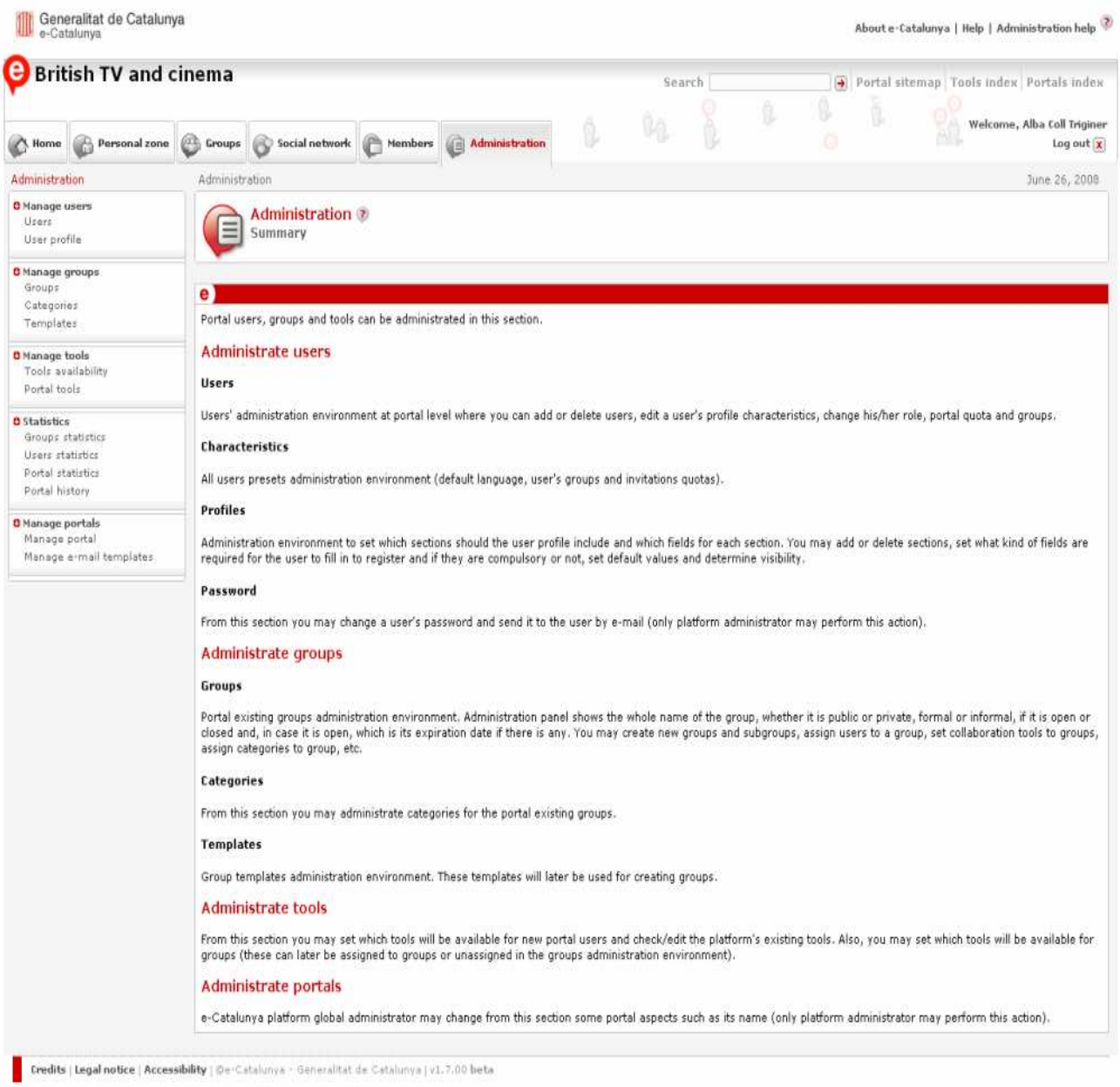

Figure 14: Example of the administration panel

2.9. ECAT Platform

When the Generalitat of Catalunya began to use the e-Catalunya platform other groups and institutions began to show interest for it. For that reason it was decided to distribute the platform source under some kind of open source licence. Currently the e-Catalunya source is being distributed with a Gnu Public License (GPL) with the name of ECAT Platform.

3. Federation system

This chapter attempts to explain what a federated model is as well as familiarizes the user with the basic concepts and technologies used in a federation system. Some of the most relevant concepts that are going to be constantly used throughout the document will be presented here. This is not a full coverage of issues, but focuses only on aspects that are relevant to understand the rest of this thesis.

3.1. Technological Concepts

3.1.1. Digital Identity

Digital Identity could be defined as the electronic representation of a real-world entity. The term is usually taken to mean the online equivalent of an individual human being, which participates in electronic activities on behalf of the person in question. However a broader definition also assigns digital identities to organizations, companies and even individual electronic devices.

A digital identity is defined with a set of identity attributes. A person normally has multiple digital identities on the internet services which have different identity attributes about this person. As an example there is Laura, a biology student in the UAB with a part time job at the zoo of Barcelona. She is a user of fotolog, where she uploads the pictures of the dolphins she takes care of and she buys some ethology books on eBay. Laura has multiple of digital identities that belong to different contexts. She is laura@uab.cat, the student in the UAB that is currently cursing a biology degree, born in 1984; she is also "karral", a girl that loves to go out dancing and from the Sants neighbourhood in fotolog and Laura Creus, the dolphin caretaker, with a number identifier document and a security service number. All these identities composed of different attributes represent the same real person.

3.1.2. Digital Certificate

In public key cryptography the key used to encrypt is different from the key used to decrypt. The user or entity possess one public key, which may be distributed, and one private key, which is a secret only known by the owner. With

the public key other users or entities may encrypt messages sent to the owner that may only be decrypt with the private key. This assures confidentiality. Also the owner of the pair of keys may sign a message using his private key. This signature can only be verified with the public key thus assuring the authentity of the message sender. Moreover, with the use of signature, the receiver may know if he message have been tampered along the way.

To distribute public keys bound with a user or entity identity digital certificates are used. A digital certificate is an electronic document which incorporates a digital signature and it may be used to verify that a public key belongs to an individual.

In a public key infrastructure (PKI) scheme, one or more parties, known as certificate authorities (CA), certify ownership of the key pairs signing the digital certificate. In a web of trust scheme, the signature is either of the user (a self-signed certificate) or other users ("endorsements").

In this thesis digital certificates are going to be used to build a circle of trust between the different entities that will comprise the simulated federation.

3.1.3. HTTP

Hypertext Transfer Protocol (HTTP) is a request/response communication protocol for the transfer of information on the Internet between a client and a server. Its development was coordinated by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF), culminating in the publication of a series of Request for Comments (RFCs).

HTTP is generally used over TCP/IP though it may be implemented over any other network protocol. Typically HTTP communications are initiated by the client by sending a request message to a particular port of a server. When the server receives a request message it sends a reply message back. This message is composed by a status line such as "403 Forbidden" and a message body that may contain a message error, an html document etc.

There are eight request methods defined in HTTP, but the most important two are the GET and POST methods. The GET method retrieves whatever information is identified by the Request-URI whereas POST submits data included in the body of the request to be processed to the identified resource.

HTTP is a stateless protocol, meaning that it doesn't store any information about older connections. To be able to implement some web applications that need to "remember" what the user had been done before is necessary the use of cookies.

Cookies are parcels of text sent by a server to a web client and then sent back unchanged by the client each time it accesses that server.

3.1.4. HTTPS

HTTPS refers to the use of the HTTP protocol explained above over TLS or SSL. TLS adds a security layer to the Internet protocol stack as shown in Figure 15.

The TLS protocol allows applications to communicate across a network in a way designed to prevent eavesdropping, tampering, and message forgery. TLS provides endpoint authentication and communications privacy over the Internet using cryptography. Technically TLS provides two things: It encrypts data transfer between communicating parties with a chosen cipher and it allows parties to authenticate with certificates usually using X.509 certificates.

Typically authentication is usually done only one way since only the server is authenticated while the client remains unauthenticated. Hence the end user can be sure with whom it is communicating. Communications where both ends are authenticated is known as mutual authentication and normally require public key infrastructure deployment to clients.

A TLS session begins with a handshake and after that the control is given to another application layer protocol.

The Handshake is responsible for the authentication and key exchange necessary to establish or resume secure sessions. When establishing a secure *session*, the Handshake Protocol manages the following:

- **Cipher suite negotiation**: The client and server make contact and choose the cipher suite that will be used throughout their message exchange.
- **Authentication of the server and optionally, the client**: The server sends back its identification in the form of a digital certificate. Then the client, in HTTPS case a browser, verifies whether it is signed by one of the valid CAs from the internal list of the browser. This list is distributed within the browser. The browser should also check that the certificate is not in the Certificate Revocation List (CRL), but usually this is not activated by default.
- **Session key information exchange**: The client and server exchange random numbers and a special number called the Pre-Master Secret. These numbers are combined with additional data permitting client

and server to create their shared secret, called the Master Secret. The Master Secret is used by client and server to generate the write MAC secret, which is the session key used for hashing, and the write key, which is the session key used for encryption.

This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the key material until the connection closes.

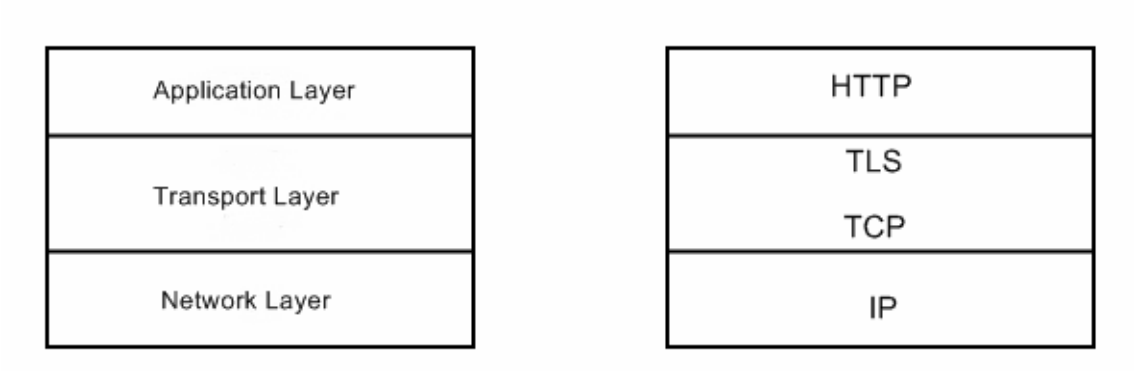

Figure 15: HTTPS on the protocol stack

3.1.5. SOAP

SOAP is a protocol for exchanging XML-based messages over computer networks, and normally it is used over HTTP or HTTPS. SOAP forms the foundation layer of the web services protocol stack providing a basic messaging framework upon which abstract layers can be built.

SOAP began at Microsoft as XML-RPC, created by Dave Winer back in 1998. It was developed to replace the existing RPC's on the market that were not suited for use over the Internet. In 2003 version 1.2 became a W3C recommendation.

XML was chosen as the standard message format because of it's widespread acceptance by major corporations and open source development efforts. Additionally, a wide variety of freely available tools significantly ease the transition to a SOAP-based implementation.

SOAP is a XML based way to send and receive information over a network such as the Internet. With the right software support such as the Jakarta Tomcat server you will get the ability to do Remote Procedure Call (RPC) with your XML messages and receive the information back to you in a XML message. There are different ways to use this; in its easiest form you use a web browser to access an on-line site that provides you with an interface to call the objects. The technical solutions are hidden behind the web interface.

Another way to utilize SOAP is to incorporate it in an application and use the XML messages to send information to a server containing the objects used in the application. The application is then only an empty shell that resides on your local computer allowing the calls to be made to the server where the objects and the data are stored. In this way all the clients don't have to be replaced when changes in the objects are done. Also it's possible to ensure that the data is stored in a safe environment and accessed only as intended.

A SOAP message is contained in an envelope. Within this envelope are two additional sections: the header and the body of the message. SOAP messages use XML namespaces. The header contains relevant information about the message. For example, a header can contain the date the message is sent, or authentication information. It is not required, but must always be included at the top of the envelope when it's present.

Most of the federation software products use SOAP to create communications channels between different entities.

3.2. Federation Fundamentals

3.2.1. Single Sign-On

Single Sign-On (SSO) is a method of access control that enables users to authenticate themselves only once and gain access to resources of multiple software systems.

3.2.2. Single Sign-Out

Single Logout (SLO) or Single sign-off is the reverse process of Single Sign-On whereby a single action of signing out terminates access to multiple software systems.

3.2.3. Federations

A federation is a group of organizations or service providers which allow sharing of user identity information among each other based on a built trust. Initially the primary motivation behind federations was solving the single-sign-on problem associated with the secure exchange of user data among cooperating organizations, either within an enterprise or among partners, suppliers and

customers through extranets. Traditional SSO was impractical for extranets or Web services because partners may not agree on a single SSO vendor, and it is not possible to have a unified database. Such a database might have to include up-to-date information on both companies' employees, for example, a task hampered not just by practical but also privacy and business considerations. So, due to this lack of solutions the federation concept was born.

Federations normally comprise Identity Providers (IdP) and Service Providers (SP). Depending on which specification and/or implementation are being used specific functionalities can vary; nevertheless their main role generally remains the same. An Identity Provider is a trusted entity which authenticates users, maintains sessions and issues claims to other entities. A Service Provider is a role performed by a system that provides services to end users or other system entities. It controls the access to the services it offers (applications) and asks for user's identity information.

The name Identity Provider and Service Provider were introduced by the Liberty Alliance in its first specifications release. Most of the federation specifications use this terminology but some exceptions may be found such as in OpenID where the IdP is called OpenID Provider (OP) and the SP is known as the Relying Party.

Figure 16 presents and overview of a Federation concept. The basic purpose is the access possibility to any resource (hosted in SP) inside the Federation using the authentication service (IdP) from any of the Institutions that belongs to the Federation.

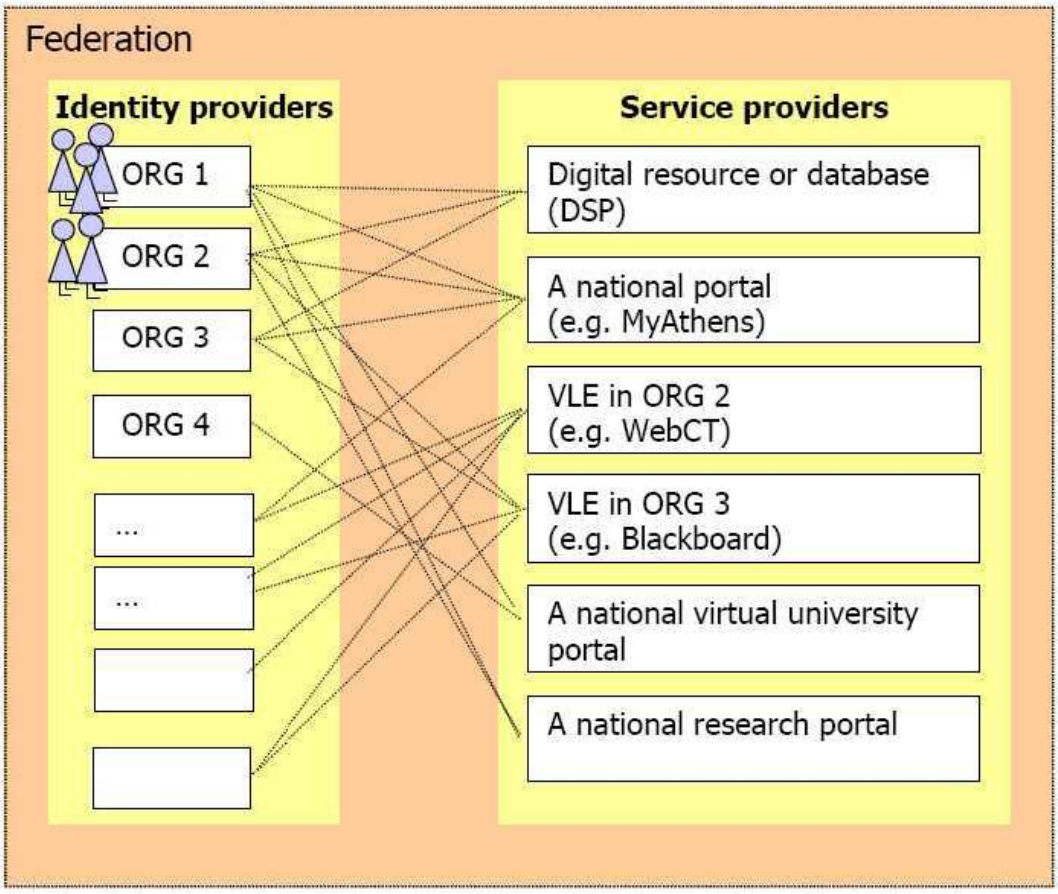

Figure 16: Overview of the Federation concept

Related to that is the term **Identity Federation**. It is used when a principal's identity, stored at an identity provider, is in some way associated with another identity of that principal, stored at a different identity provider.

Federated Identity Management enables enterprises and service providers to securely link and exchange identity information across partner, supplier and customer organizations. Leading enterprises have deployed identity federation to get closer to partners, improve customer service, accelerate the execution of business partnerships and alliances, cut the cost and complexity of integrating outsourced services, and free themselves from vendor lock-in.

3.2.3.1. Federation motivation

In the current web oriented information technology world, an increasingly large number of services take place via the web. These web services often require the user to identify himself in order to receive personalized information or services. The result is an increase in the amount of authentication procedures a user must

deal with. One of the methods to cope with this is using federated identity management.

Federations provide the framework for the organizations to implement methods that reduce the frequency in which users must authenticate. Single sign-on is a prominent method, and is normally used in federations. With single sign-on, a user needs only one authentication procedure in order to be granted access to resources on all federation members.

This not only eases the user authentication process but erases the need to manage a countless number of passwords that a user may have. Having to manage a lot of passwords may result in passwords written down on pieces of paper, the use of one password for multiple services, passwords easy to remember and therefore computationally weak etc. All these solutions normally used reduce severely the level of safety that the identity of the user possesses.

In addition to solving these matters, federations allow for efficient management, control, and movement in a radically distributed world as well as offer interaction between different types of authenticity control. As organizations integrate more tightly with trading partners and outsourcers, federated identity provides a flexible mechanism that authenticates users from partner organizations and provides them with seamless access to protected online resources.

This way companies can share applications without needing to adopt the same technologies for directory services, security and authentication. Even more application developers can forget about authentication and security issues and rely on the federation architecture.

3.2.4. Attributes

An attribute is a characteristic of the identity of a user or an entity. Mainly, attributes are stored at the identity provider, in the form of a value that is identified by its key. This format and the key name are crucial in order to guarantee portability in the communication protocol and therefore compatibility among providers.

An identifier is a special kind of attribute that, alone or in a set, identify an identity and thus a principal. A similar kind of attribute is the Credentials. These are sent by the principal to prove its identity (the act of Authentication) to the identity provider and establish a trust relation called a Authentication Session. Although credentials are identifiers, not all identifiers are credentials.

Attributes sent by the IdP are normally used by SPs to determine if an already authenticated user has authorization to access protected resources. Let's

put an example of a federation between two universities that share resources. An authenticated student of university A tries to access a data base of papers and articles of university B. University A would send an attribute on behalf of this user claiming that he is a student. But that database can only be accessed by professors and students that are doing a scholarship. Thus when receiving the "student" attribute from university A the database would reject the access request. On the contrary, if instead of a student a professor tried to consult that same database on university B, he would be granted access when the system protecting the database received from university A the attribute "professor" on behalf of the user.

3.2.5. Account linking

Account linking or account federation is the process by which an identity provider and service provider agree on some common unique identifier, and bind their local user identity to this common unique identifier. This linking can be anonymous using pseudonyms instead of the user identifier in cases where user privacy is required.

4. Federation Specifications

There are different specifications that define means to form a federation and its performance. This chapter will expose a brief overview of some of the most important specifications. This chapter attempts to illustrate each specification in a non technical way so those readers that only wish to know the basic features of each one should not have to read their specific working. For a more extended explanation of each specification, readers can consult the next chapter.

At the end of this chapter a comparison between these specifications is done.

4.1. SAML

The quest for developing one standard began with a company named Securant. Securant worked for several months, with a few dozen of its customers, partners, and other vendors, to create a standard called "AuthXML." A few days after AuthXML was publicly announced, Netegrity and VeriSign announced their own standardization effort for the same problem "S2ML." Through the encouragement of customers and analysts, it was decided that it was best for all involved if the efforts were combined.

During this time, a couple of meetings were held with representatives of all of the leading Web access control vendors. Ultimately, it was decided to merge the two standards efforts at the OASIS standards organization.

The Organization for the Advancement of Structured Information Standards (OASIS) is a global consortium that drives the development, convergence and adoption of e-business and web service standards. Members of the consortium decide how and what work is undertaken through an open, democratic process.

All of the members of that group joined OASIS to work on the new standard, which took its name from the two standards it was based on, and was named SAML (the "Security Assertion Markup Language"). Their work resulted in the SAML specification, released on January 9, 2001 and in November of 2002 SAML V1.0 became an OASIS standard. In September of 2003 SAML 1.1 was released with some minor changes. In this thesis SAML is not going to be explained in so much detail as to notice the changes between SAML 1.0 and 1.1 and thus there going to explained together as SAML 1. Only mention that the few changes done in 1.1 are clarifications, corrections, deprecations of some SAML 1.0 elements and some minor changes and SAML schema and digital signature guidelines.

SAML 1.x main objective was the single sign-on between different domains and the first specification done to achieve such a thing, so compared with other specifications it is quite simple.

4.2. Liberty Phase 1

The Liberty Alliance was formed in 2001 by approximately 30 organizations to establish open standards, guidelines and best practices for federated identity management. These standards are divided into three modules providing different functionality, namely ID-FF, ID-WSF and ID-SIS.

The Liberty Identity Federation Framework (ID-FF) is a set of protocols, schema and profiles that can be used to implement identity federation by supplying features such as account linkage and single sign-on. It is developed, based on the SAML v1.1 standard, with flexibility in mind and is therefore suitable for heterogeneous platforms and all kinds of systems, so it will integrate well with systems and specifications that are used. ID-FF versions 1.0, released on July 2002, and 1.1 comprise what is known as Liberty Phase 1 while ID-FF version 1.2, released on November 2003, and is referred as Liberty Phase 2.

Liberty Phase 1 extends SAML 1.0 by adding its own profiles for how to wield SAML assertions. These additional profiles add support for account linking, discovery service, and global logout. Discovery service is the process that the SP performs in order to determine which IdP it has to request Single Sign-On.

ID-FF 1.0 introduced the terminology of Identity Provider (IdP) and Service Provider (SP). Liberty ID-FF1.1 incorporates feedback and errata from the 1.0 specification.

The Liberty Identity Web Services Framework (ID-WSF) provides another set of protocols, schema and profiles for interoperability that is built on top op the ID-FF. ID-WSF can be used for discovery, consumption and creation of identity web services and provides features for enhanced and personalized identity services and attribute sharing.

The Liberty Identity Service Interface Specification (ID-SIS) is built on top of the ID-WSF and contains a collection of specifications for services providing specific functionality, so Liberty enabled organizations are able to exchange these services.

In this thesis the study of Liberty Alliance specifications set has centred in ID-FF even though ID-WSF and ID-SIS are going to be mentioned.

4.3. Liberty Phase 2

This set of standards extends ID-FF with new functionality, such as one-time assertions of identity (for anonymity), metadata exchange, and affiliate relationships. Liberty Phase 2 is a set of standards that extend the existing Liberty framework with functionality for discovering and offering identity-related services.

4.4. SAML 2.0

SAML 2.0 is a major upgrade with respect to SAML 1.0/1.1 and was ratified as an OASIS Standard in March 2005. It is based on SAML 1.0 and 1.1 and ID-FF 1.1. It introduces features like account linking, metadata exchange between providers, identity provider discovery and Single Logout.

4.5. WS-Federation

WS-Federation specification was developed by BEA Systems, BMC Software, CA, Inc., IBM, Layer 7 Technologies, Microsoft, Novell, and VeriSign and it is built on WS-Security and WS-Trust specifications. WS-Federation describes how to use the existing Web services security building blocks to provide federation functionality, including single sign-on, single logout and attribute management across a federation. WS-Trust enables applications to construct trusted SOAP message exchanges. This trust is represented through the exchange and brokering of security tokens. This specification provides a protocol agnostic way to issue, renew, and validate these security tokens. WS-Security specification enables applications to conduct secure SOAP message exchanges.

4.6. OpenID

The original OpenID authentication protocol was developed in May 2005 by Brad Fitzpatrick, creator of popular community website LiveJournal, while working at Six Apart. OpenID support was soon implemented on LiveJournal and fellow LiveJournal engine community DeadJournal for blog post comments, and quickly

gained attention in the digital identity community. Web developer JanRain was an early supporter of OpenID, providing OpenID software libraries and expanding its business around OpenID-based services.

By early December, Non-Assertion Agreements were collected by the major contributors to the protocol, and the final OpenID Authentication 2.0 and OpenID Attribute Exchange 1.0 specifications were ratified on December 5 2007. OpenID is a lightweight specification that only defines the basic functionalities of a federated identity model.

The characteristic that differentiates OpenID from the other specifications is its definition and management of user identifiers. OpenID defines user identifiers as URLs or XRIs. XRI is a scheme and resolution protocol for abstract identifiers compatible with URIs and IRIs. The goal of XRI is a standard syntax and discovery format for abstract, structured identifiers that are domain, location, application, and transport independent, so they can be shared across any number of domains, directories, and interaction protocols.

In OpenID Single Sign-On is accomplished sending the user identifier to the SP so the SP may be able to discover the user IdP either using the URL or through the XRI identifier.

4.7. Comparison

SAML and ID-FF are the specifications that share more common properties, which is normal since both have inspired in each other. Their specifications organization is similar in document arrangements, number of documents, information organization and terminology. Consequently it was easier to understand them and comprehend their basic working.

WS-Federation has lots of dependencies with other WS-* Specifications, which does not make easier the job to understand its basics. Moreover WS-Federation does not present its federated model with the same concepts or terminology as SAML and ID-FF and that makes the comparison harder and more complex.

OpenID is the specification that goes further away from the others. It's a very simple specification whose uses are in different contexts than the other specifications. OpenID is very popular in wikis, blogs and communities like LiveJournal where security is not the main objective and where little functionality is required. In these cases OpenID is the perfect solution since OpenID deployments are easy and simple. One issue to consider about OpenID is its vulnerability to

phishing attacks. The goal of a phisher is to be able to log in to some website, the Real Website, as another user. In order to do this, the phisher "persuades" the victim user to go to a website that the phisher controls and that looks like the Real Website. When the victim logs in, thinking it is the Real Website, the phisher gets the user username and password. So, in the normal case the phisher has to "convince" the victim to go to a fake page while in OpenID the phisher only need to persuade the victim to go to any page and get him lo log in using his OpenID identifier. Then the phisher finds out where your OpenID provider is, but instead of sending you there the phisher sends you to his fake provider, which then just proxies the real provider, stealing the victim login as it does.

The following sections do a brief comparison of some aspects of the specifications reviewed.

4.7.1. Single Sign-On control flows

Communication between the identity provider and the service provider can be done using a front channel or a back channel. Front channel communicates the IdP and the SP through the user's browser. Back channel connects the identity provider and the service provider without redirection via the user's browser (normally using a SOAP channel between the IdP and the SP).

ID-FF, SAML 1.x, SAML 2.0, OpenID and WS-Federation specify both front-and-back channel mechanisms.

4.7.2. Single Logout

Liberty Alliance, WS-Federation and SAML 2.0 support Single Logout initiated both by the IdP or the SP. SAML 1.x and OpenID doesn't support any kind of Single Logout.

4.7.3. Account federation / Linkage

Liberty Alliance, WS-Federation and SAML 2.0 support account federation via persistent or transient pseudonymous identifiers. SAML 1.x and OpenID don't specify any means for account federation.

4.7.4. Client profiles

Liberty Alliance, WS-Federation and SAML 2.0 specify client profiles for both browser and smart clients. SAML 1.x and OpenID specify only profiles for browsers, thought there actually are some implementations of OpenID that work with smart clients.

4.7.5. Security Tokens/Security Assertions

Security Tokens are assertions exchanged between the different actors (IdP, SP, User Agent etc.) of federation use cases. Except for OpenID, all specifications define the use of formalized assertions.

Liberty Alliance and extend SAML assertions for communicating authentication and authorization security tokens between providers. Other token types can be used embedded in SAML assertions. WS-Federation builds his Security Tokens on WS-Security's profiles of X509v3 and Kerberos. SAML 1.x and 2.0 uses SAML assertions which specifications offer an explicit schema design for assertions that are explicitly delineated, self-contained data objects, encoded in XML. OpenID doesn't feature a formalized notion of a security token. Security assertions are comprised of a set of key-value pairs, thus limiting reusability in other protocol contexts.

4.7.6. Security

Liberty Alliance, WS-Federation and SAML 1.x and 2.0 provide robust security provisions based on explicit stipulations in profiles, the bindings profiles to employ, as well as in the design of the SAML assertion and WS-Secuity profiles semantics. In the OpenID specifications there are security provisions in terms of key establishment, message signature and verification mechanisms, and use of SSL/TLS-protected channels. However, employment of these security provisions is entirely optional on the part of implementers and/or deployers.

4.7.7. User Privacy Controls

Liberty Alliance and SAML 1.x and 2.0 and WS-Federation supplies privacy controls in their specifications. An example of privacy control the use of pseudonyms between providers to refer to the same user so his identity may remain unknown. In WS-Federation privacy controls offer optional privacy support by deferring to WS-Policy for access controls. In OpenID end-user privacy is not presently explicitly addressed in its specification.

4.7.8. Developer Organization

Liberty Alliance is an open standard community that includes vendors, end-users and non-profit organizations. Their specifications include ID-FF 1.x and 2.0, ID-WSF and ID-SYS, but in this document all of them have been seen as a whole pack and referenced as "Liberty Alliance specifications".

WS-Federation specifications were developed by Microsoft, IBM, VeriSign, BEA and RSA Security.

SAML 1.x and 2.0 were developed by OASIS, a global consortium that drives the development, convergence and adoption of e-business and web service standards.

OpenID has arisen from the open source community. Its father is Brad Fitzpatric.

4.7.9. Implementation Cost

All specifications compared in this document, Liberty Alliance, WS-Federation, SAML 1.x and 2.0 and OpenID are free to implement in products and services.

4.7.10. IdP Discovery

Except for SAML 1.x, all other specifications compared in this document define different ways to deploy a Discovery Service.

4.7.11. Metadata

Federations require agreements between system entities regarding identifiers, binding support and endpoints, certificates and keys etc. To describe this information in a standardized way the use of metadata is very useful, and in some cases, imperative.

Liberty Alliance, WS-Federation and SAML 2.0 define in their specifications a metadata schema, and means for publishing and retrieving service locations. Liberty Alliance metadata schema is based in ID-WSF Metadata specifications, WS-Federation metadata schema is based in WS-Metadata Exchange specifications and SAML 2.0 metadata schema is based in SAML Metadata Specifications (based

on Liberty Alliance Metadata). All three set of specifications define DNS and well-known location for publishing and retrieving services.

SAML 1.x does not include any definition of metadata in its own specification, but SAML 2.0 includes one profile in its metadata specification for use in describing SAML 1.x entities and profiles.

OpenID relies on XRDS documents, which can be found by resolving an XRI, for what is essentially "service metadata" (information about an end-point or service of the federation) in the other specifications terminology. Additionally, OpenID relies upon establishing so-called associations for exchanging keying material between an SP and an IdP.

4.7.12. Protocol bindings

Mappings from request-response message exchanges into standard messaging or communication protocols are called protocol bindings. Liberty Alliance and SAML 2.0 specify six bindings used in different profiles: HTTP redirect, HTTP POST, Artifact (reference to an assertion + SOAP call), SOAP-over-HTTP, reverse SOAP-over-HTTP, URI binding. WS-Federation defines four bindings, missing the reverse SOAP-over-HTTP and the URI binding. SAML 1.x specifies just one binding, the SAML SOAP Binding. In addition to SOAP, implicit in SAML 1.x Web Browser SSO are the precursors of the HTTP POST Binding, the HTTP Redirect Binding, and the HTTP Artifact Binding. These are not defined explicitly, however, and are only used in conjunction with SAML 1.x Web Browser SSO. The notion of binding is not fully developed until SAML 2.0. OpenID only specifies HTTP POST and HTTP GET bindings.

4.7.13. User Identifier Treatment

Liberty Alliance, WS-Federation and SAML 1.x and 2.0 do not specify the nature of the user-wieldable end-user identifiers. It is left as an exercise for profiliers, and/or implementors, and/or deployers. OpenID presumes and specifies that user identifiers, termed *OpenID identifiers* herein, are necessarily directly dereferenceable in order to contact the user's IdP using HTTP or HTTP/TLS — thus the identifiers are specified to be of "HTTP:" or "https:" URI schemes.

	SAML 1.x	SAML 2.0	ID-FF	WS-Federation	OpenID
SSO control flow front channel	Yes	Yes	Yes	Yes	Yes
SSO control flow back channel	Yes	Yes	Yes	Yes	Yes
SLO	No	Yes	Yes	Yes	No
Account federation	No	Yes	Yes	Yes	No
Client profiles	Browser	Browser and smart client	Browser and smart client	Browser and smart client	Browser
Security Tokens	SAML assertions	SAML assertions	SAML assertions	Security tokens built on WS-Security's profiles	key-value pairs
User Privacy Controls	Yes	Yes	Yes	Yes	No
Developer Organization	OASIS	OASIS	Liberty Alliance	Microsoft, IBM, VeriSign, BEA and RSA Security.	Open source community
Implementation Costs	Free	Free	Free	Free	Free
IdP Discovery	No	Yes	Yes	Yes	Yes
Metadata	Yes	Yes	Yes	Yes	Yes
Protocol Bindings	SAML SOAP Binding	HTTP redirect, HTTP POST, Artifact, SOAP-over-HTTP, reverse SOAP-over-HTTP, URI binding.	HTTP redirect, HTTP POST, Artifact, SOAP-over-HTTP, reverse SOAP-over-HTTP, URI binding.	HTTP redirect, HTTP POST, Artifact, SOAP-over-HTTP,	HTTP POST and HTTP GET bindings
User Identifier Treatment	None specified	None specified	None specified	None specified	URI schema

Table 1: Specifications comparison.

5. Federation specifications in depth

In this chapter each specification is reviewed in more depth than in the last chapter, both in how it works and how it is structured.

5.1. SAML 1

The SAML specification is made up of the following components:

- **Assertion and protocols (SAML core):** refers to the general syntax and semantics of SAML assertions as well as the protocol used to request and transmit those assertions from one system entity to another. So SAML Core defines SAML assertions along with SAML request and response elements.
- **Bindings and profiles (SAML bindings):** A SAML binding determines how SAML requests and responses map onto standard messaging or communications protocols. A SAML profile is a concrete manifestation of a defined use case using a particular combination of assertions, protocols, and bindings.
- **Conformance specifications:** Different SAML implementations may only implement part of these specifications. Conformance specifications set the basic standards to which an implementation of the SAML specification must conform before it can be called a conformant implementation. This helps with interoperability and compatibility.
- **Security and privacy considerations***:* This specification covers the security risks in the SAML architecture specifically, the way SAML addresses those risks and the risks that are not addressed.

Figure 17 shows how SAML assertions, protocols, bindings and profiles relate to each other.

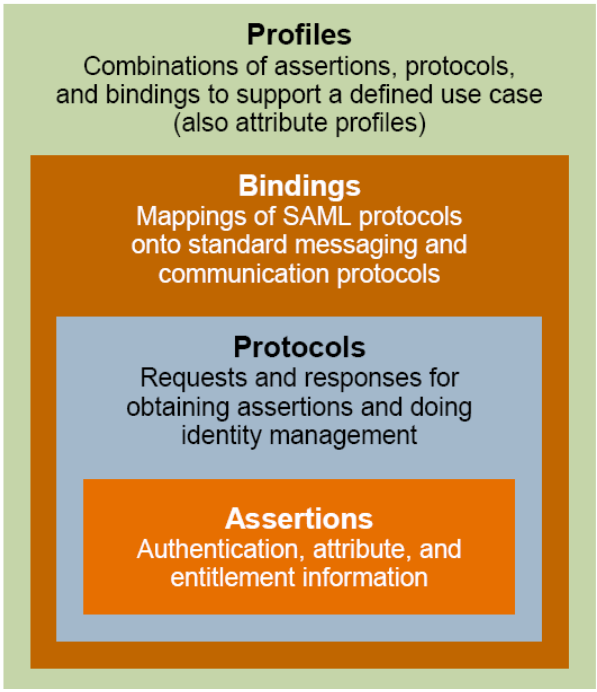

Figure 17: SAML components and how they relate to each other.

5.1.1. SAML 1 Assertions

An assertion is a package of information that supplies one or more statements made by an issuer. SAML allows issuers to make three different kinds of assertion statement:

- **Authentication:** The specified subject was authenticated by a particular means at a particular time.
- **Authorization Decision:** A request to allow the specified subject to access the specified resource has been granted or denied.
- **Attribute:** The specified subject is associated with the supplied attributes.

5.1.2. SAML 1 Protocols

A SAML *protocol* describes how certain SAML elements are packaged within SAML request and response elements, and gives the processing rules that SAML entities must follow when producing or consuming these elements.

For the most part, a SAML protocol is a simple request-response protocol.

Figure 18: SAML Request-Response protocol.

SAML 1 defines different requests message. The **AssertionIDReference** requests assertions by reference to its assertion identifier and the **AssertionArtifact** requests assertions by supplying an assertion artifact that represents it.

The **Query** request is an extension point that allows extension schemas to define new types of query. There are three query requests: Authentication, Attribute and AuthorizationDecision.

The **AuthenticationQuery** makes a query for authentication information. The AuthenticationQuery element is used to ask available authentication assertions for a subject. A successful response will be in the form of assertions containing authentication statements. The AuthenticationQuery has an optional element called AuthenticationMethod. If it is present, the query made is "What assertions containing authentication statements do you have for this subject with the supplied authentication".

The **AttributeQuery** makes a query for attribute information. The AttributeQuery element is used to make the query "Return the requested attributes for this subject." A successful response will be in the form of assertions containing attribute statements.

The **AuthorizationDecisionQuery** makes a query for an authorization decision. The AuthorizationDecisionQuery element is used to make the query "Should these actions on this resource be allowed for this subject, given this evidence?" A successful response will be in the form of assertions containing authorization decision statements.

Response messages specifie the status of the corresponding SAML request and a list of zero or more assertions that answer the request.

5.1.3. SAML 1 Bindings

Bindings are the mappings from SAML request-response message exchanges into standard messaging or communication protocols.

SAML 1 only specifies the **SOAP binding**. In this binding is mandatory that the use of SOAP is over HTTP. Moreover, the SAML request response elements must be embedded inside the SOAP message body.

A system entity acting as a SAML requester transmits a SAML <Request> element within the body of a SOAP message to a system entity acting as a SAML responder. The SAML requester never includes more than one SAML request per SOAP message or includes any additional XML elements in the SOAP body. The SAML responder returns either a <Response> element within the body of another SOAP message or a SOAP fault code. Like the requester, the SAML responder never includes more than one SAML response per SOAP message or includes any additional XML elements in the SOAP body. If a SAML responder cannot, for some reason, process a SAML request, it returns a SOAP fault code.

5.1.4. SAML 1 Profiles

Profiles describe in detail how SAML assertions, protocols, and bindings combine to support a defined use case. A profile describes how SAML assertions are embedded in or combined with other objects by an originating party, communicated from the originating site to a destination, and subsequently processed at the destination.

SAML 1 does not specify any Discovery Service. As said above that means that it does not specify the necessary steps or mechanisms that let the user arrive from the Service Provider to the IdP and so, the description of both profiles begins with the request at the identity provider.

The two profiles specified are the Browser/Artifact Profile and the Browser/POST Profile.

5.1.4.1. Browser/Artifact Profile

The browser/artifact profile relies on a reference to the needed assertion travelling in a SAML artifact, which the destination site must dereference from the source site in order to determine whether the user is authenticated. The interaction sequence is shown in Figure 19:

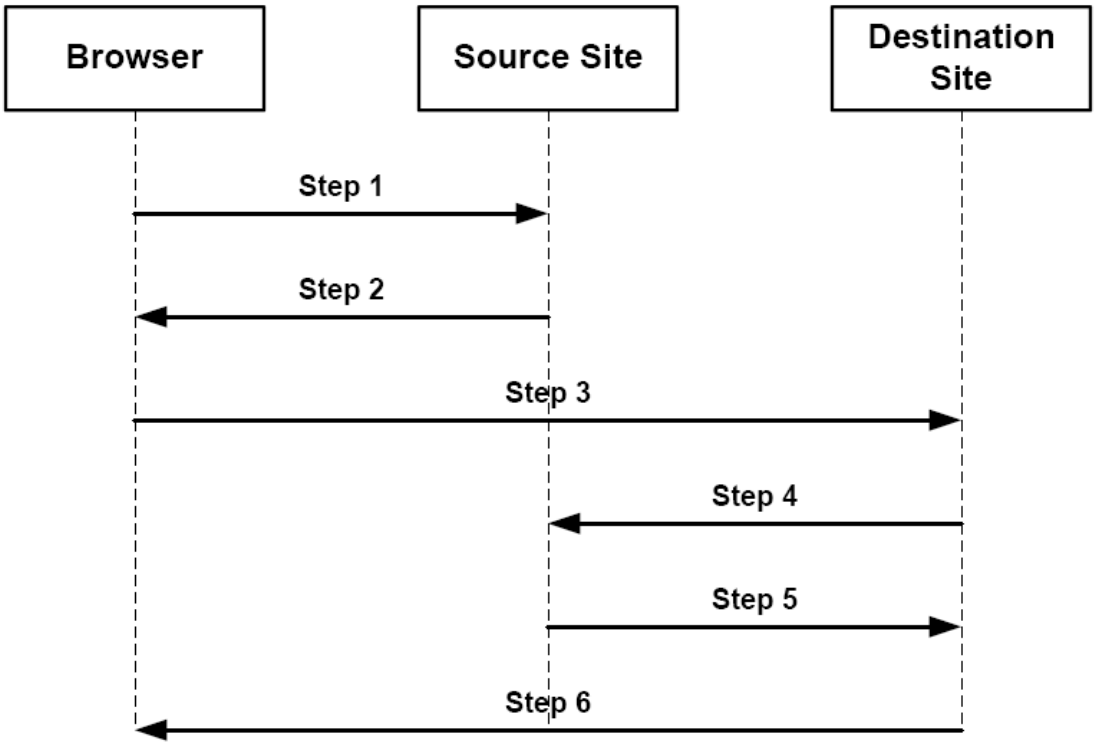

Figure 19: SAML Browser/Artifact Profile.

In step 1, the user's browser accesses the IdP, with information about the desired target at the destination site attached to the URL. Next, in step 2, the IdP responds and redirects the user's browser to the assertion consumer service at the SP. Afterwards, in step 3, the user's browser accesses the artifact receiver URL, with a SAML artifact representing the user's authentication information attached to the URL. In steps 4 and 5, the SP dereferences the one or more SAML artifacts in order to acquire the SAML authentication assertion that corresponds to each artifact. Both steps use a SAML protocol binding for a SAML request-response message exchange between the IdP and the SP. The SP sends a Request message to the source site, requesting assertions by supplying assertion artifacts. If the source site is able to find or construct the requested assertions, it responds with the requested assertions. Otherwise, it returns an appropriate error code. Finally, in step 6, the user's browser is sent an HTTP response that either allows or denies access to the desired resource.

5.1.4.2. Browser/Post Profile

The browser/POST profile consists of a series of two interactions, the first between a user equipped with a browser and a source site, and the second directly between the user and the destination site. The interaction sequence is shown in the following figure, with the following sections elucidating each step. The interaction sequence is shown in Figure 20:

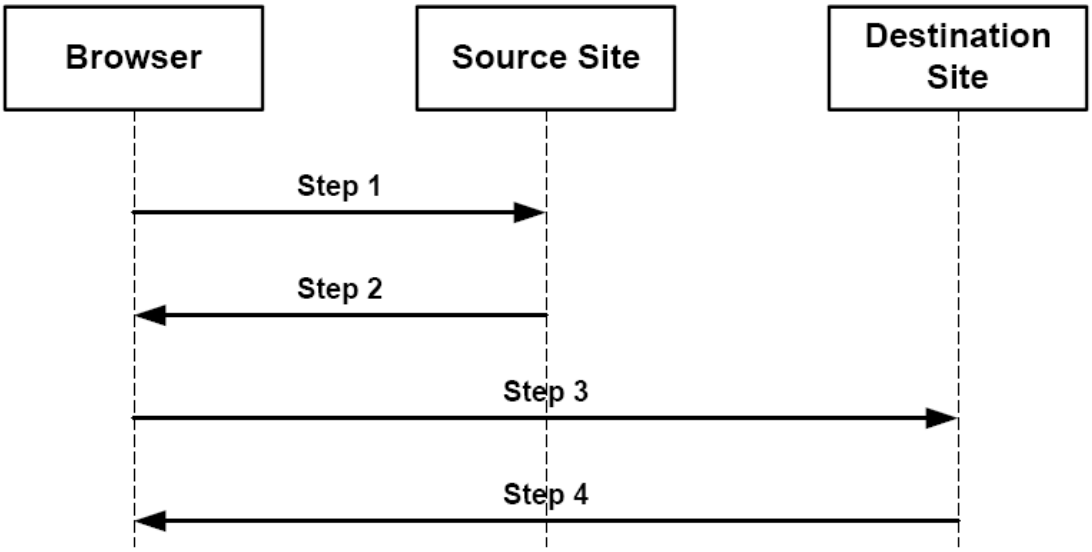

Figure 20: SAML Browser/Post Profile

In step 1, the user's browser accesses the IdP, with information about the desired target at the destination site attached to the URL. Afterwards, in step 2, the source site generates HTML form data containing a SAML Response which contains an SSO assertion.

One SAML response is included within the FORM body and it may include multiple SAML assertions which at least one is an SSO assertion. Next in step 3, the browser submits the form containing the SAML response using the following HTTP request. Finally in step 4, the user's browser is sent an HTTP response that either allows or denies access to the desired resource.

5.2. Liberty Alliance Specifications

5.2.1. Liberty Phase 1

As said in last chapter, Liberty Phase 1 is composed of ID-FF 1.0 and 1.1 specifications. In this thesis ID-FF specifications are not going to be explained in so much detail as to notice the changes between versions 1.0 and 1.1. Hence they are going to be explained together as Liberty Phase 1. Only mention that ID-FF 1.1 incorporates mainly feedback and errata from the 1.0 specification.

It was the Liberty Alliance with this specification set that introduced the terms Identity Provider and Service Provider which latter were accepted and broadly used in other specifications such as SAML. In this thesis this terms have been used from the beginning since currently are the most used inside the federation context.

ID-FF specifications comprise the following components:

- **Liberty architecture authentication context**: defines syntax for the definition of authentication context statements and an initial list of Liberty authentication context classes. An authentication context is defined as the information additional to the authentication assertion itself that the service provider may require before it makes an entitlements decision.
- **Liberty architecture protocol context:** defines a set of protocols that collectively provide a solution for identity federation management, cross-domain authentication, and session management. It also defines provider metadata schemas that may be used for making a priori arrangements between providers. The term protocol has the same meaning as in SAML specifications.
- **Liberty architecture binding profiles:** defines the bindings and profiles of the Liberty protocols and messages to HTTP-based communication frameworks. The terms bindings and profiles have the same meaning as in the SAML specifications. Moreover this document provides some security considerations.
- **Liberty architecture overview and guidelines:** This document is non-normative. However, it provides implementers and deployers guidance in the form of policy/security and technical notes. Moreover introduces the federation and identity concepts with some practical examples.

5.2.1.1. Liberty Authentication contexts

The function of the authentication contexts is to ease the SP's task of assessing and comparing authentication assertions by defining particular authentication contexts that are representative of current technologies and practices among identity providers. The main objectives of the authentication contexts are:

- Make it easier for the IdP and SP to come to an agreement on what are acceptable authentication contexts by giving them a framework for discussion.
- Make it easier for SPs to indicate their preferences when requesting a step-up authentication assertion from an authentication authority.
- Simplify for SPs the burden of processing authentication context declarations by giving them the option of being satisfied by the associated class.
- Insulate SPs from the impact of new authentication technologies.
- Make it easier for IdPs to publish their authentication capabilities, for example, through WSDL. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

A particular Liberty authentication context class defines a list of required characteristics of the processes, procedures, and mechanisms by which the identity provider verifies the Principal before issuing an identity, protects the secrets on which subsequent authentications are based, and the mechanisms used for this authentication. These characteristics are categorized as:

- **Identification:** characteristics that describe the processes and mechanism the identity provider uses to initially create an association between a Principal and the identity (or name) by which the Principal will be known.
- **Physical Protection:** characteristics that specify physical controls on the facility housing the identity provider's systems (for example, site location and construction, access controls).
- **Operational Protection:** characteristics that describe procedural security controls employed by the identity provider (for example, security audits, records archival).

- **Technical Protection:** characteristics that describe how the "secret" (the knowledge or possession of which allows the Principal to authenticate to the identity provider) is kept secure.
- **Authentication Method:** characteristics that define the mechanisms by which the Principal authenticates to the identity provider (for example, a password versus a smartcard).

5.2.1.2. Liberty Protocols

There are four protocols defined in the Liberty specifications which are built on SAML:

- **Single Sign-On and Federation**: The protocol by which identities are federated and by which single sign-on occurs. This protocol extends SAML request response protocols.
- **Name Registration**: The protocol by which a provider can register an alternative opaque handle (or *name identifier*) for a Principal.
- **Federation Termination Notification**: The protocol by which a provider can notify another provider than a particular identity federation has been terminated (also known as defederation).
- **Single Logout**: The protocol by which providers notify each other of logout events.

The **Single Sign-On and Federation Protocol** defines a request and response protocol by which single sign-on and identity federation occurs. The protocol works as follows:

1. A service provider issues an <AuthnRequest> request to an identity provider, instructing the identity provider to provide an authentication assertion to the service provider.
2. The identity provider responds with either an <AuthnResponse> containing authentication assertions to the service provider or an artefact that can be de-referenced into an authentication assertion as in the SAML artifact profile.

During federation, the identity provider generates an opaque handle that serves as the initial name identifier that both the service provider and the identity

provider use in referring to the Principal when communicating with each other. This name identifier is termed the <IDPProvidedNameIdentifier>.

The **Name registration Protocol** allows the service provider to register a different opaque handle with the identity provider. This opaque handle is termed the <SPProvidedNameIdentifier>. Until the service provider registers a different name, the identity provider will use <IDPProvidedNameIdentifier> to refer to the Principal when communicating with the service provider. After a service provider's name registration, the identity provider uses the <SPProvidedNameIdentifier> for <saml:NameIdentifier> elements when communicating to the service provider about the Principal.

With the **Federation Termination Notification protocol** a principal may terminate an identity federation either from the service provider or the identity provider. The service provider where the termination notification has been done sends a <FederationTerminationNotification> message to the other provider. Semantically, if it is done from the service provider, this is stating that it will no longer accept authentication assertions from the identity provider for the specified Principal. Otherwise, if it is done from the identity provider, the identity provider, this is stating that it will no longer provide authentication assertions to the service provider for the specified Principal.

The **Single Logout Protocol** provides a message exchange protocol by which all sessions authenticated by a particular identity provider are near-simultaneously terminated. The Single Logout Protocol is used either when a principal logs out at a service provider or when the principal logs out at an identity provider. When the Principal invokes the single logout process at a service provider, the service provider sends a <LogoutRequest> message to the identity provider that provided the authentication service for the session. Either if the principal invokes a logout at the identity provider or a service provider, the identity provider sends a <LogoutRequest> message to each service provider to which it provided authentication assertions in the current session with the Principal, with the exception of the service provider that sent the <LogoutRequest> message to the Identity Provider.

5.2.1.3. Liberty Binding

Like SAML 1, Liberty Phase 1 only specifies the SOAP binding. Since the Liberty protocols are an extension of the SAML, the SOAP binding for Liberty adheres to the processing rules for the "SOAP binding for SAML". Just like SAML, the SOAP binding for Liberty uses HTTP as the transport mechanism.

5.2.1.4. Liberty Profiles

Liberty profiles are grouped into categories, according to the Liberty protocol message intent. There are five categories defined:

- **Single Sign-On and Federation**: profiles by which a service provider obtains an authentication assertion from an identity provider facilitating single sign-on and identity federation.
- **Name Registration**: profiles by which service providers and identity providers specify the name identifier to be used when communicating with each other about the Principal.
- **Identity Termination Notification**: The profiles by which service providers and identity providers are notified of federation termination.
- **Single Logout**: The profiles by which service providers and identity providers are notified of authenticated session termination.
- **Identity Provider Introduction**: The profile by which a service provider discovers which identity providers a Principal may be using.

5.2.1.4.1. Single Sign-On and Federation

Inside the **Single Sign-On and Federation** group Liberty defines four profiles:

- Liberty Browser Artifact Profile
- Liberty Browser POST Profile
- Liberty WML POST Profile
- Liberty-Enabled Client and Proxy Profile

Since the **Liberty Browser Artifact** and the **Liberty Browser Post** profiles are close to SAML Browser Artifact and SAML Browser Post profiles they are not going to be explained further in this thesis.

The **Liberty WML POST profile** relies on the use of Wireless Markup Language (WML) events to instruct a WML browser to submit an HTTP form. The Wireless Markup Language, is a markup language intended for devices that implement the Wireless Application Protocol (WAP) specification, such as mobile phones, and preceded the use of other markup languages now used with WAP, such

as XHTML and even standard HTML. This profile is an adaptation of the "Browser/form post profile" for SAML

The **Liberty-enabled client and proxy profile** specifies interactions between Liberty-enabled clients and/or proxies, service providers, and identity providers. Liberty-enabled client is a client that has, or knows how to obtain, knowledge about the identity provider that the Principal wishes to use with the service provider. In addition a Liberty-enabled client receives and sends Liberty messages in the body of HTTP requests and responses. A Liberty-enabled proxy is a HTTP proxy (typically a WAP gateway) that emulates a Liberty enabled client. Unless stated otherwise, all statements referring to LECP are to be understood as statements about both Liberty-enabled clients as well as Liberty-enabled proxies.

5.2.1.4.2. Register Name Identifier

Inside the **Register Name Identifier** group Liberty defines four profiles:

- Register Name Identifier HTTP-Redirect-Based initiated at Identity Provider profile
- Register Name Identifier HTTP-Redirect-Based initiated at Service Provider profile
- Register Name Identifier SOAP/HTTP-Based initiated at Identity Provider profile
- Register Name Identifier SOAP/HTTP-Based initiated at Service Provider profile

The **HTTP-Redirect-Based** profiles relie on a HTTP 302 redirect to communicate between the identity provider and the service provider.

The **SOAP/HTTP-Based** profiles relie on a SOAP call from the identity provider to the service provider.

Both HTTP-Redidect-Based and SOAP/HTTP-Based profiles may be initiated either in the identity provider or the service provider; thought the HTTP-Redirect-Based profile can not be self-initiated in the identity provider. **Figure 21** illustrates the SOAP/HTTP-Based profile initiated at the identity provider.

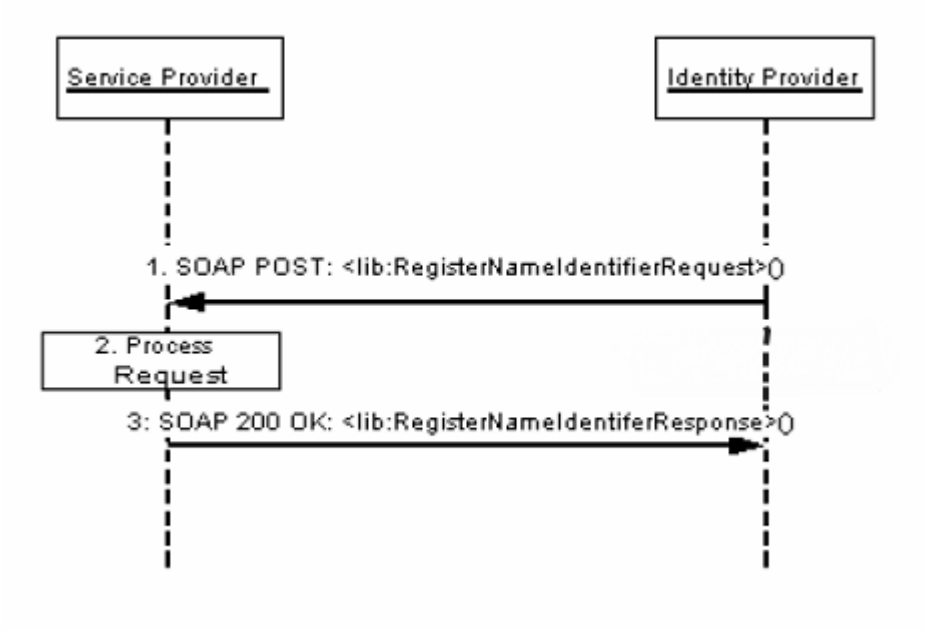

Figure 21: SOAP/HTTP-Based profile for registering name identifiers initiated at the idp

In step 1 the identity provider sends a <lib:RegisterNameIdentifierRequest> protocol message specifying the new name identifier and the old name identifier. Next, in step 2 the service provider records the new name identifier. Finally in step 3, after the service provider has successfully registered the new name identifier responds according to the rules defined in the Liberty name registration protocol.

5.2.1.4.3. Identity Termination Notification

Inside the **Identity Termination Notification** group Liberty defines four profiles:

- Federation Termination Notification HTTP-Redirect-Based initiated at Identity Provider
- Federation Termination Notification HTTP-Redirect-Based initiated at Service Provider
- Federation Termination Notification SOAP/HTTP-Based initiated at Identity Provider
- Federation Termination Notification SOAP/HTTP-Based initiated at Service Provider

The HTTP-Redirect-Based and SOAP/HTTP-Based profiles work as in the Register Name Identifier group profiles and similar to it they can be either initiated

in the identity provider or the service provider. When initiated in the identity provider this is stating to the service provider that it will no longer provide the Principal's identity information to the service provider and that the identity provider will no longer respond to any requests by the service provider on behalf of the Principal.

The other way around, when these profiles are initiated in the service provider, this is stating to the identity provider that the Principal has requested that the identity provider no longer provide the Principal's identity information to the service provider and that service provider will no longer ask the identity provider to do anything on the behalf of the Principal.

Figure 22 illustrates the HTTP-Redirect-Based profile for Federation Termination initiated at the identity provider.

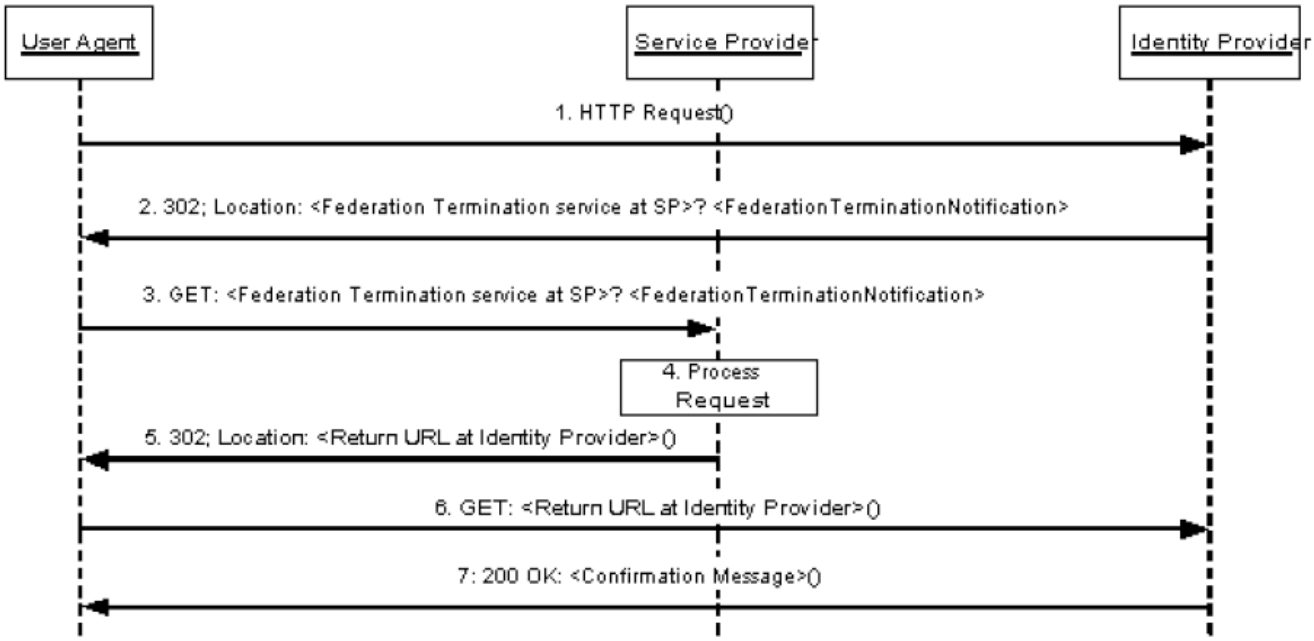

Figure 22: Http-redirected-based profile for federation termination initiated in the identity provider

In step 1, the user agent accesses the identity federation termination service URL at the identity provider specifying the service provider with which identity federation termination should occur. How the service provider is specified is implementation-dependent and is not specified in the Liberty specifications. Next, in step 2, the identity provider's federation termination service URL responds and

redirects the user agent to the federation termination service at the service provider. After that, in step 3, the user agent accesses the service provider's federation termination service URL with the <lib:FederationTerminationNotification> information attached to the URL fulfilling the redirect request. In step 4, the service provider processes the <lib:FederationTerminationNotification> according to the rules defined in the Liberty protocol specifications. Afterwards, in step 5, the service provider's federation termination service responds and redirects the user agent back to identity provider using a return URL location specified in the identity provider metadata. In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect request. Finally, in step 7, the user agent is sent an HTTP response that confirms the requested action of identity federation termination with the specific service provider.

5.2.1.4.4. Single Logout

The single logout profiles synchronize session logout functionality across all sessions that were authenticated by a particular identity provider. Inside the **Single Logout** group Liberty defines four profiles:

- Single Logout HTTP-Based initiated at the identity provider
- Single Logout SOAP/HTTP-Based initiated at the identity provider
- Single Logout HTTP-Based initiated at the service provider
- Single Logout SOAP/HTTP-Based initiated at the service provider

Either initiating the Single Logout at the identity provider or initiating it at the service provider, the identity provider will communicate a logout request to each service provider with which it has established a session for the Principal. The negotiation of which single logout profile the identity provider uses to communicate with each service provider is based upon the identity provider metadata and the service providers metadata which will participate in the profile.

The **HTTP-Based** relies on using either HTTP 302 redirects or HTTP GET requests to communicate logout requests from an identity provider to the service providers. The **SOAP/HTTP-Based** relies on SOAP over HTTP messaging to communicate logout requests from an identity provider to the service providers.

Figure 23 illustrates the HTTP-GET implementation for Single Logout initiated at the identity provider.

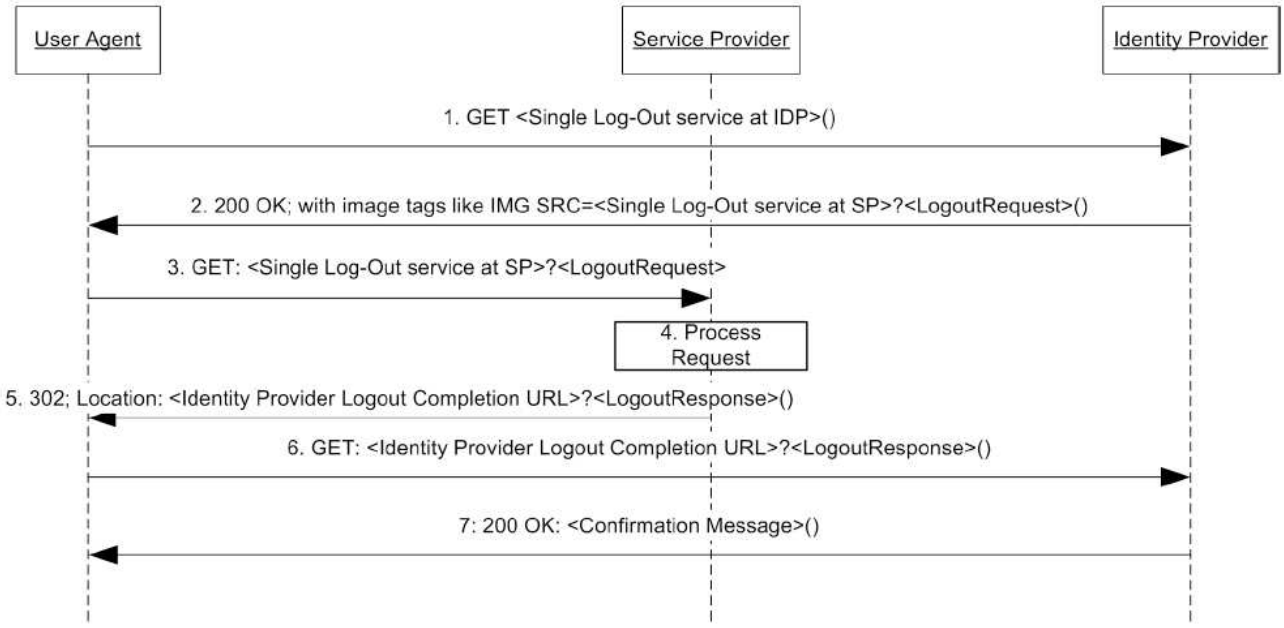

Figure 23: HTTP-GET implementation for single logout initiated at identity provider

In step 1, the user agent accesses the single logout service URL at the identity provider indicating that all service providers for which this identity provider has provided authentication assertions during the Principal's current session must be notified of session termination and requested to logout the Principal. Next, in step 2, the identity provider's single logout service responds with an HTML page that includes image tags referencing the logout service URL for each of the service providers for which the identity provider has provided an authentication assertion during the Principal's current session. Afterwards in step 3, the user agent, as a result of each image load, accesses the service provider's single logout service URL with <lib:LogoutRequest> information attached to the URL. This step may occur multiple times if the HTTP response includes multiple image tag statements (one for each service provider that has been issued authentication assertions during the Principal's current session). In step 4, the service provider processes the <lib:LogoutRequest> according to the rules defined in the Liberty protocol specifications. The service provider invalidates the session of the Principal referred to in the name identifier it received from the identity provider in the <lib:LogoutRequest>. In step 5, the service provider's single logout service responds and redirects the image load back to the identity provider's logout completion URL. This location will typically point to an image that will be loaded by

the user agent to indicate that the logout is complete. Next in step 6, the user agent accesses the identity provider's logout completion URL fulfilling the redirect request. Finally in step 7, the user agent is sent an HTTP response that confirms the requested action of a single logout has been completed.

5.2.1.4.5. Identity Provider Introduction

With this profile a service provider discovers which identity providers a Principal is using. In identity federation networks having more than one identity provider, service providers need a means to discover which identity providers a Principal uses. The introduction profile relies on a cookie that is written in a domain that is common between identity providers and service providers in an identity federation network. The domain that the identity federation network predetermines for a deployment is known as the common domain, and the cookie containing the list of identity providers is known as the common domain cookie.

The name of this cookie is _liberty_idp and the format of its content is a list of base64-encoded identity provider IDs separated by a single white space character. The identity provider ID is defined in its metadata and it is unique between all the entities.

The common domain cookie writing service appends the identity provider ID to the list. If the identity provider ID is already present in the list, it may remove and append it when authentication of the Principal occurs. The intent is that the most recently established identity provider session is the last one in the list.

Normally after the identity provider authenticates a Principal, it sets the common domain cookie. Liberty does not specify the means by which the identity provider may set the cookie since they are implementation-specific.

When a service provider needs to discover which identity providers the Principal uses, it invokes a protocol exchange designed to present the common domain cookie to the service provider after it is read by an HTTP server in the common domain. If the HTTP server in the common domain is operated by the service provider, the service provider may redirect the user agent to an identity provider's intersite transfer service for an optimized single sign-on process.

The specific means by which the service provider reads the cookie are implementation-specific and thus they are not defined in Liberty specifications.

5.2.1.5. Liberty Metadata

For providers to communicate between them, they must a priori have obtained metadata regarding each other. These provider metadata includes items such as X.509 certificates, service endpoints, and provider's identifiers. This specification defines metadata schemas for identity providers and service providers that may be used for provider metadata exchange.

Certain provider metadata are generic to both service providers and identity providers.

Metadata holds three primary functions:

- Declarations of entity metadata for providers, principals and devices, and affiliations
- Entity trust metadata, which enables entities to cast business decisions based on the characteristic trust information provided, conveyed through document signatures, server authenticated protected channel delivery of the instance using TLS, DNS zone signatures.
- Origin and document verification through signature use in HTTPS retrieval of the instance documents, DNS signatures, and document level signatures.

5.2.2. Liberty Phase 2

Liberty Phase 2 is composed by ID-FF 1.2 and ID-WSF specifications. ID-FF 1.2 modifies some elements of ID-FF 1.1 schema and adds one new protocol and two new profiles.

5.2.2.1. ID-FF 1.2

As said above, ID-FF 1.2 adds one new protocol, called Name Identifier Mapping Protocol, and two new profiles known as Name Identifier mapping Profile and Name Identifier Encryption Profile. This thesis has not gone into Liberty Phase 1 specifications in depth so the schema modifications done in ID-FF 1.2 are not going to be explained.

5.2.2.1.1. Name Identifier Mapping Protocol

When a service provider requires a name identifier for a Principal with which it has an identity federation relationship, but which references an identity federation between the identity provider and another service provider, it can use this protocol to obtain such an identifier. This allows the requesting provider to communicate with the other service provider about the Principal without an identity federation for the Principal between them. The resulting value is recommended to be encrypted so as to obscure the actual value from anyone but the second service provider. To the requester, it will be an opaque and one-time value.

Upon receipt of a <NameIdentifierMappingRequest> message, an identity provider that supports this protocol responds with a <NameIdentifierMappingResponse> message. Both messages are signed and their signature is checked to the receiving end.

5.2.2.1.2. Name Identifier Mapping Profile

This profile only defines a SOAP-based profile. It relies on a SOAP request and response to query for and return the NameIdentifier. A requesting service provider issues a SOAP request to an identity provider, requesting a different NameIdentifier for a named Principal in the namespace of a service provider or affiliation group. This NameIdentifier may then be used to query another Liberty provider offering SAML services for additional information about the named Principal. **Figure 24** illustrates the SOAP-based profile for name identifier mapping.

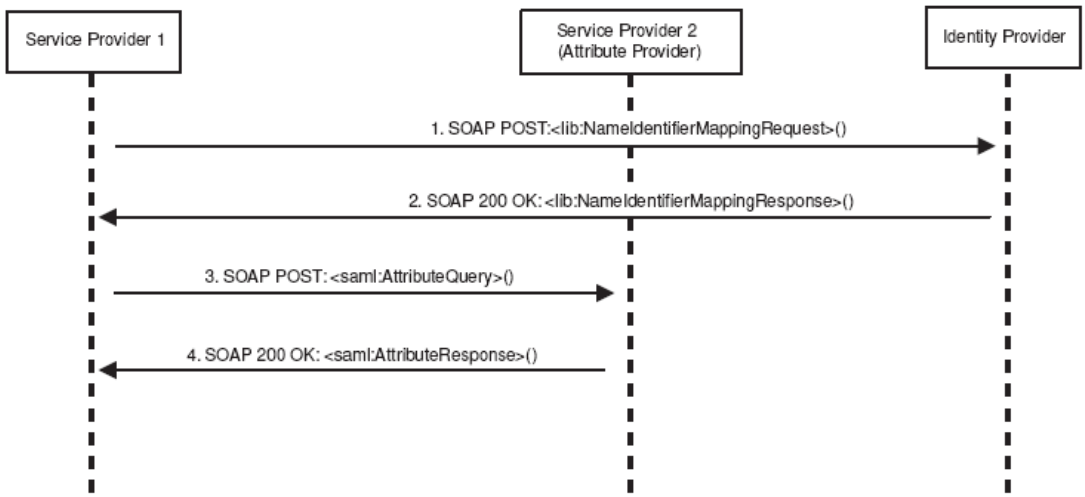

Figure 24: SOAP-based profile for name identifier mapping

In step 1, the service provider sends a SOAP over HTTP request to the SOAP endpoint of the identity provider it is querying. The SOAP message contains one <lib:NameIdentifierMappingRequest> element in the SOAP. The identity provider processes the <lib:NameIdentifierMappingRequest> according to the rules defined in the Liberty protocol specifications.

Afterwards, in step 2, the identity provider responds to the <lib:NameIdentifierMappingRequest> with a SOAP 200 OK <lib:NameIdentifierMappingResponse> message.

Steps 3 and 4 are not normatively specified by Liberty, and are shown only for illustrative purposes. A service provider receiving a <saml:AttributeQuery> may return a <saml:AttributeStatement> accordingly to the SAML specifications.

In addition to the usual considerations relating to Liberty and SAML, is recommended that the identity provider encrypts or otherwise obfuscate the NameIdentifier returned to the requesting service provider, so that it is opaque to the requester. A way of accomplishing this is using the Name Identifier Encryption Profile.

5.2.2.1.3. Name identifier Encryption Profile

The Liberty NameIdentifier encryption profile allows a principal NameIdentifier to be encrypted such that only the identity or service provider possessing the decryption key can deduce the identity of the principal when the NameIdentifier is included in a SAML or Liberty protocol message. The identifier is

encrypted in such a fashion that it is a different value when requested by different providers or multiple times, reducing the chance for correlation of the encrypted value across multiple logical transactions.

5.2.2.2. ID-WSF

The Liberty Identity Web Services Framework (ID-WSF) is a standards-based architecture for identity-based web services. An identity-based web service is a particular type of a web service that acts upon some resource to retrieve information about an identity, update information related to an identity, or perform some action for the benefit of some identity. A resource is either data related to some identity or service acting for the benefit of some identity.

ID-WSF 2.0 is optimized to work with SAML 2.0 SSO, with ID-WSF-based attribute sharing operations following SAML SSO. ID-WSF can however also work with other SSO schemes (with appropriate profiling).

ID-WSF is currently in its 2.0 version and it's the one that is going to be reviewed in this thesis. It defines two main roles: the Web Service Consumer (WSC) donned by a system entity when it makes a request to a web service, and the Web Service Provider (WSP) role donned by a system entity when it provides a web service.

ID-WSF 2.0 specifications comprise:

- **Liberty ID-WSF Discovery Service Specification:** describes protocols and schema for the description and discovery of ID-WSF identity services.
- **Liberty ID-WSF SOAP Binding Specification:** defines the Liberty Identity Web Services Framework (ID-WSF) SOAP binding. It specifies simple SOAP message correlation, consent claims, and usage directives.
- **Liberty ID-WSF Security Mechanisms Specification:** specifies security mechanisms that protect identity services. This includes mechanisms for authentication, integrity and confidentiality protection, and the means for sharing information necessary for authorization decisions.
- **Liberty ID-WSF Interaction Service Specification:** specifies an identity service that allows providers to pose simple questions to a Principal. It may sometimes be necessary for an *identity* service to interact with the owner of the *resource* that it is exposing, to collect attribute values, or to obtain permission to share the data with a *Web Services Consumer* (WSC). Additionally, in situations where the individual on whose behalf the

request is being made is not the resource owner, the *identity* service may need to interact with either or both principals. The interaction service (IS) specification defines schemas and profiles that enable a *Web Services Provider* (WSP) to interact with relevant principals.

- **Liberty ID-WSF 2.0 Static Conformance Requirements:** defines what features are mandatory and optional for implementations conforming to this version of the Liberty Alliance Specifications.
- **Liberty ID-WSF Data Services Template Specification:** provides protocols for the querying and modifying of data attributes when implementing a data service using the Liberty Identity Web Services Framework (ID-WSF).
- **Liberty ID-WSF Architecture Overview:** non-normative document intended to provide an overview of the relevant features of the Liberty ID-WSF Version 2.0 Specifications.
- **Liberty ID-WSF Client Profiles Specification:** specifies profiles for some cases where a client performs an active role in such transactions, other than performing the functions of a standard browser.
- **Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification:** defines a SASL-based ID-WSF Authentication Protocol, along with an ID-WSF Authentication Service and ID-WSF Single Sign-On Service, based on the Authentication Protocol.
- **Liberty ID-WSF People Service Specification:** A user's People Service (PS) is an interface into those other users with which the owning user wishes to interact with in some online fashion - these other users possibly categorized into arbitrary groups. The PS provides a flexible, privacy respecting framework by which a user can manage/track the people they know and how these other users are related.
- **Liberty ID-WSF Subscriptions and Notifications Specification:** provides protocols for subscription and notification. A subscription is a mechanism by which a WSC can register to receive notifications from a WSP when some data changes or some event happens. Since there is usually data involved, it is common that data services will incorporate subscription features.

5.2.3. ID-SIS

The Liberty Identity Service Interface Specification (ID-SIS) uses the ID-WSF and ID-FF specifications to provide networked identity services, such as

contacts, presence detection, or wallet services that depend on networked identity. The SIS comprises the following specifications:

- **Liberty ID-SIS Directory Access Protocol Specification**: Describes a web service offering directory information as an instance of a data-oriented identity web service, based on the Liberty ID-WSF Data Services Template.
- **Liberty ID-SIS Content SMS and MMS Specification**: Describes a web service that layers the ID-WSF 1.1 framework on MM7 to add identity-based invocation and addressing.
- **Liberty ID-SIS Personal Profile Service Specification**: Describes a web service that provides a Principal's basic profile information, such as their contact details, or name. Liberty also supplies implementations guidelines for this service.
- **Liberty ID-SIS Employee Profile Service Specification**: Describes a web service that provides a Employee's basic profile information, such as their contact details, or name. Liberty also supplies implementations guidelines for this service.
- **Liberty ID-SIS Contact Book Service Specification**: Specifies a web identity service that allows a Principal to manage contacts for private and business acquaintances, friends, family members, and even for the Principal. Liberty also supplies implementations guidelines for this service.
- **Liberty ID-SIS Geolocation Service Specification**: Specifies a web service offering geolocation information associated with a Principal. Liberty also supplies implementations guidelines for this service.
- **Liberty ID-SIS Presence Service Specification**: Specifies a web service offering presence information associated with a Principal. Liberty also supplies implementations guidelines for this service.

5.3. SAML 2.0

In the following sections the major modifications or added functionalities are going to be explained.

5.3.1. Authentication contexts

Like Liberty Alliance, and for the same reasons, SAML 2.0 introduces in its specifications the authentication context concept. A particular authentication

context declaration defined in SAML 2.0 captures characteristics of the processes, procedures, and mechanisms by which the authentication authority verified the subject before issuing an identity, protects the secrets on which subsequent authentications are based, and the mechanisms used for this authentication. These characteristics are categorized in the Authentication Context schema as follows:

- **Identification:** Characteristics that describe the processes and mechanism the IdP uses to initially create an association between a subject and the identity (or name) by which the subject will be known.
- **Technical Protection:** Characteristics that describe how the "secret" (the knowledge or possession of which allows the principal to authenticate to the IdP) is kept secure.
- **Operational Protection:** Characteristics that describe procedural security controls employed by the IdP (for example, security audits, records archival).
- **Authentication Method:** Characteristics that define the mechanisms by which the principal of the issued assertion authenticates to the IdP (for example, a password versus a smartcard).
- **Governing Agreements:** Characteristics that describe the legal framework (e.g. liability constraints and contractual obligations) underlying the authentication event and/or its associated technical authentication infrastructure.

5.3.2. Bindings

SAML 2.0 defines the following bindings:

- SAML SOAP Binding (based on SOAP 1.1)
- Reverse SOAP (PAOS) Binding:
- HTTP Redirect (GET) Binding:
- HTTP POST Binding
- HTTP Artifact Binding
- SAML URI Binding

5.3.2.1. SAML SOAP Binding

There are no major changes between SAML 1.1 and SAML 2.0 SOAP Binding and so this thesis will not go in depth in this binding and will centre in the new ones presented by SAML 2.0.

5.3.2.2. Reverse SOAP (PAOS)

The reverse SOAP binding is a mechanism by which an HTTP requester can advertise the ability to act as a SOAP responder or a SOAP intermediary to a SAML requester. The HTTP requester is able to support a pattern where a SAML request is sent to it in a SOAP envelope in an HTTP response from the SAML requester, and the HTTP requester responds with a SAML response in a SOAP envelope in a subsequent HTTP request.

The PAOS binding includes two component message exchange patterns:

1. The HTTP requester sends an HTTP request to a SAML requester. The SAML requester responds with an HTTP response containing a SOAP envelope containing a SAML request message.

2. Subsequently, the HTTP requester sends an HTTP request to the original SAML requester containing a SOAP envelope containing a SAML response message. The SAML requester responds with an HTTP response, possibly in response to the original service request in step 1.

Figure 25 shows this interaction:

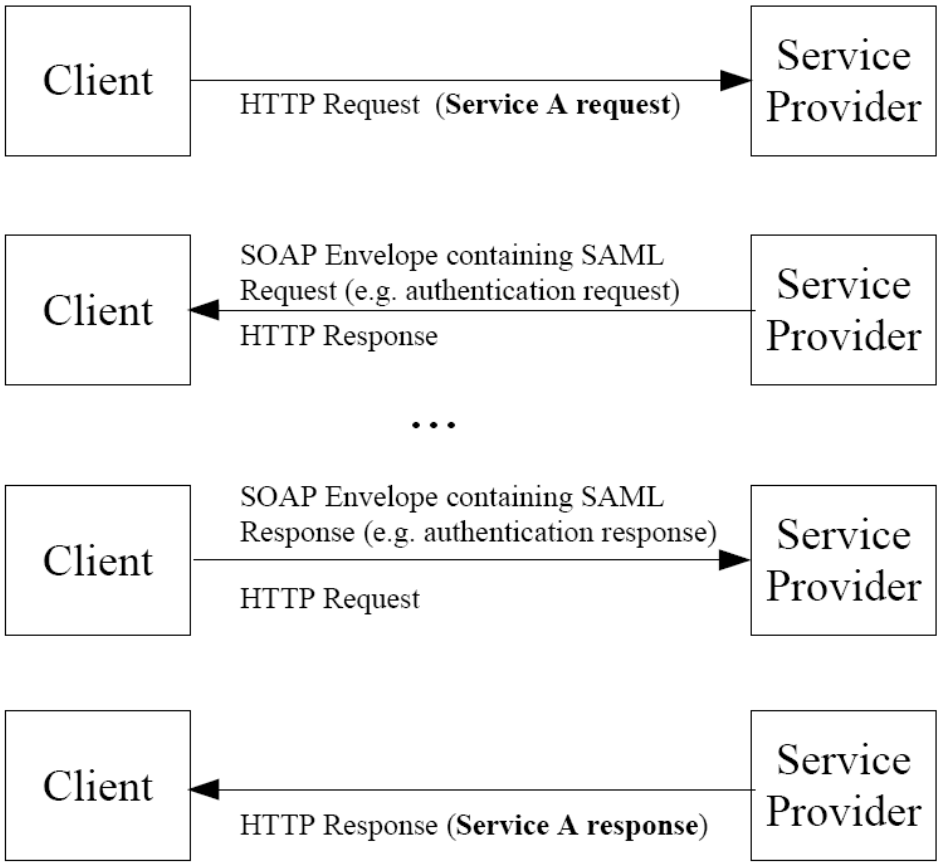

Figure 25: PAOS Binding Message Exchanges

5.3.2.3. HTTP Redirect Binding

The HTTP Redirect binding defines a mechanism by which SAML protocol messages can be transmitted within URL parameters. Permissible URL length is theoretically infinite, but unpredictably limited in practice. Therefore, specialized encodings are needed to carry XML messages on a URL, and larger or more complex message content can be sent using the HTTP POST or Artifact bindings.

This binding is some times composed with the HTTP POST binding and the HTTP Artifact binding to transmit request and response messages in a single protocol exchange using two different bindings.

The HTTP Redirect binding is intended for cases in which the SAML requester and responder need to communicate using an HTTP user agent as an intermediary. This may be necessary, for example, if the communicating parties do not share a

direct path of communication. It may also be needed if the responder requires an interaction with the user agent in order to fulfill the request, such as when the user agent must authenticate to it.

The system model used for SAML conversations via this binding is a request-response model, but these messages are sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request. The HTTP interactions before, between, and after these exchanges take place is unspecified. Both the SAML requester and the SAML responder are assumed to be HTTP responders. **Figure 26** shows the sequence diagram illustrating the messages exchanged.

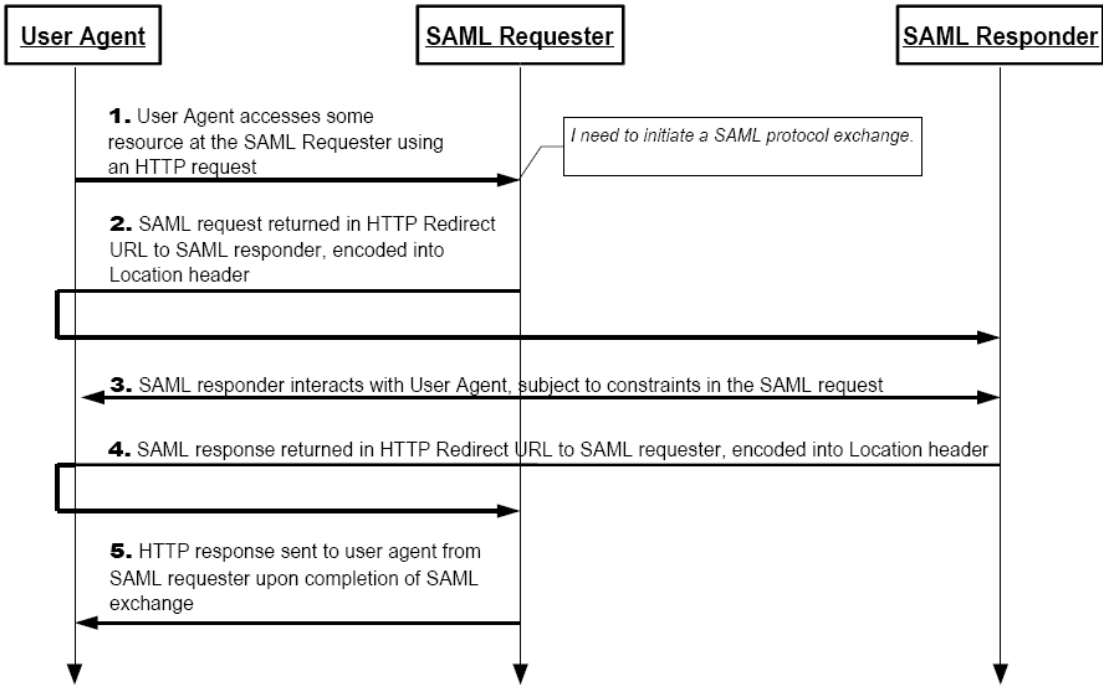

Figure 26: HTTP Redirect Binding

In step 1 the user agent makes an arbitrary HTTP request to a system entity. In the course of processing the request, the system entity decides to initiate a SAML protocol exchange. Next, in step 2, the system entity acting as a SAML requester responds to the HTTP request from the user agent in step 1 by returning a SAML request. The SAML request is returned encoded into the HTTP. The user agent delivers the SAML request by issuing an HTTP GET request to the SAML responder. Afterwards in step 3 the SAML responder normally responds to the SAML request by immediately returning a SAML response or returning arbitrary content to facilitate subsequent interaction with the user agent necessary to fulfil the request. In step 4 the responder returns a SAML response to the user agent to be returned

to the SAML requester. The SAML response is returned in the same fashion as described for the SAML request in step 2. Finally, in step 5, upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the user agent.

5.3.2.4. HTTP POST Binding

The HTTP POST binding defines a mechanism by which SAML protocol messages may be transmitted within the base64-encoded content of an HTML form control. This binding is sometimes composed with the HTTP Redirect binding and the HTTP Artifact binding to transmit request and response messages in a single protocol exchange using two different bindings.

The HTTP POST binding is intended for cases in which the SAML requester and responder need to communicate using an HTTP user agent as an intermediary. The system model used for SAML conversations via this binding is a request-response model, but these messages are sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request.

Figure 27 shows a sequence diagram illustrating the messages exchange:

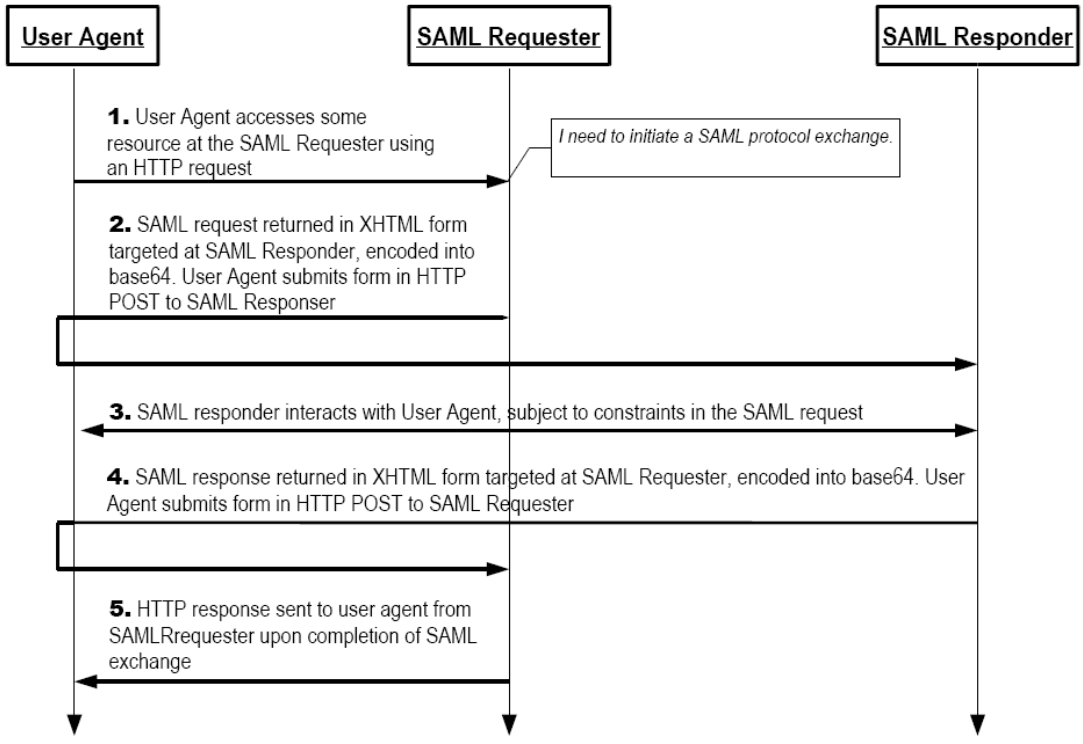

Figure 27: HTTP POST Binding

The steps shown in **Figure 27** are the same as the **Figure 26** with the difference of how the SAML request and response are sent. In **Figure 27** SAML requests and responses are embedded in an XHTML form encoded into base64.

5.3.2.5. HTTP Artifact Binding

In the HTTP Artifact binding, the SAML request, the SAML response, or both are transmitted by reference using an artifact. A separate, synchronous binding, such as the SAML SOAP binding, is used to exchange the artifact for the actual protocol message using the artifact resolution protocol.

This binding is sometimes composed with the HTTP Redirect binding and the HTTP POST binding to transmit request and response messages in a single protocol exchange using two different bindings.

The HTTP Artifact binding is intended for cases in which the SAML requester and responder need to communicate using an HTTP user agent as an intermediary, but it is not possible, for technical reasons, or not desired, for security reasons, that the transmission of the message it is done through the intermediary.

The system model used for SAML conversations by means of this binding is a request-response model in which an artifact reference takes the place of the actual message content, and the artifact reference is sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request. On the successful acquisition of a SAML protocol message, the artifact is discarded and the processing of the primary SAML protocol exchange resumes (or ends, if the message is a response).

Figure 28 shows a sequence diagram below illustrating the messages exchanged.

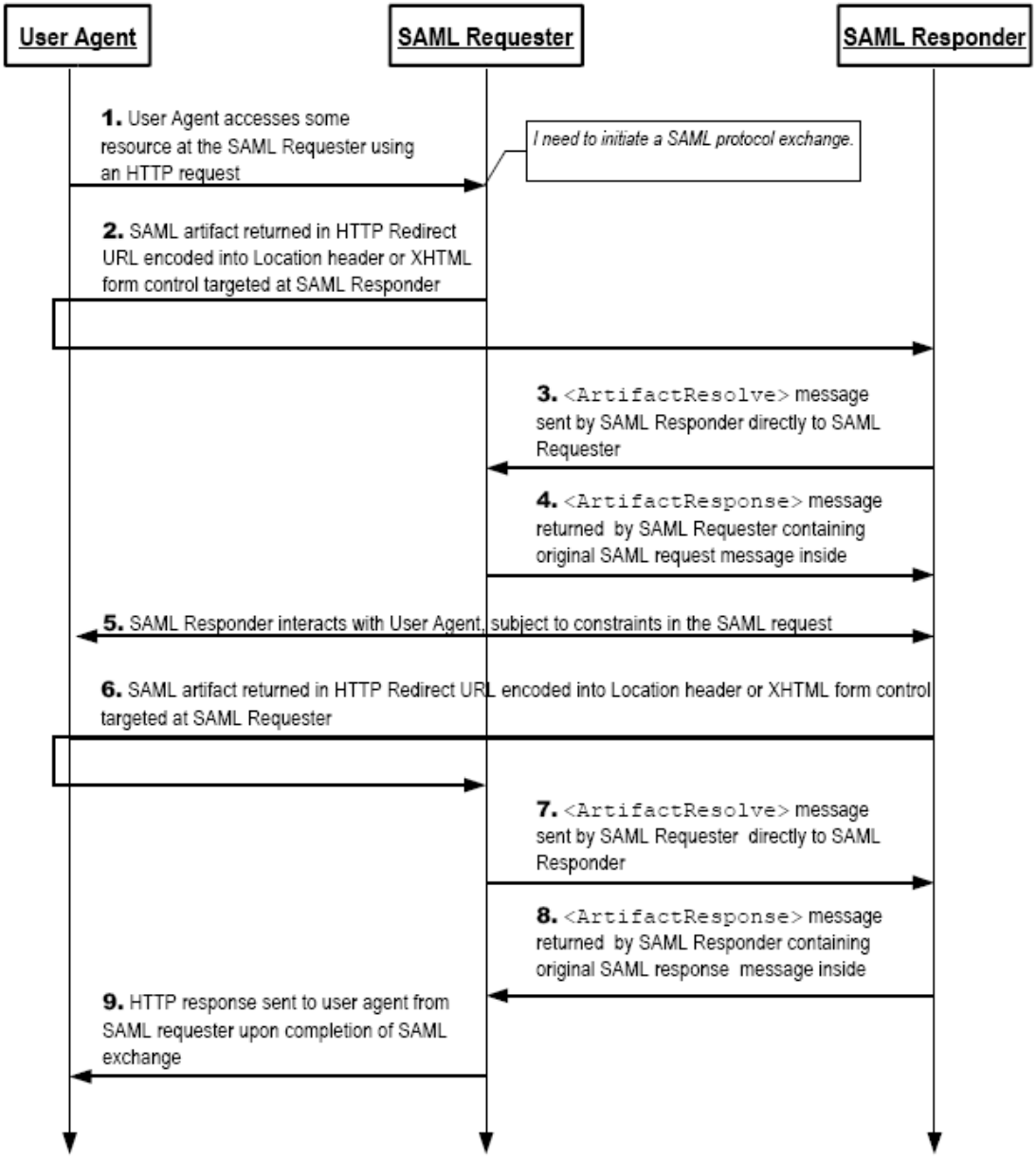

Figure 28: Artifact HTTP Binding

Initially in step 1 the user agent makes an arbitrary HTTP request to a system entity. In the course of processing the request, the system entity decides to initiate a SAML protocol exchange. In step 2 the system entity acting as a SAML requester responds to an HTTP request from the user agent by returning an artifact representing a SAML request. Next, in step 3, the SAML responder determines the SAML requester by examining the artifact, and issues a request to resolve the artifact to the SAML requester using a direct SAML binding, temporarily reversing roles. In step 4, assuming the necessary conditions are met, the SAML requester returns a response containing the original SAML request message it wishes the

SAML responder to process. Afterwards in step 5 the SAML responder responds to the SAML request by immediately returning a SAML artefact. Next, in step 6, the responder returns a SAML artifact to the user agent to be returned to the SAML requester. The SAML response artifact is returned in the same fashion as described for the SAML request artifact in step 2.The SAML requester determines the SAML responder by examining the artifact, and issues a request to resolve the artifact to the SAML responder using a direct SAML binding, as in step 3. In step 7, assuming the necessary conditions are met, the SAML responder returns a response containing the SAML response message it wishes the requester to process, as in step 4. Finally in step 8 the SAML requester returns an arbitrary HTTP response to the user agent.

5.3.2.6.SAML URI Binding

URIs are a protocol-independent means of referring to a resource. This binding is not a general SAML request/response binding, but rather supports the encapsulation of a <samlp:AssertionIDRequest> message with a single <saml:AssertionIDRef> into the resolution of a URI. The result of a successful request is a SAML <saml:Assertion> element, but not a complete SAML response. Like SOAP, URI resolution can occur over multiple underlying transports. This binding has to use HTTP either with SSL 3.0 or TLS 1.0.

5.3.3. Protocols

SAML 2 defines six protocols where different request and response messages are gathered depending on their "functionalities" similarly to the ID-FF specifications. Some of these messages are inherited from SAML 1 while some others are new for the new functionalities. The defined protocols are the following:

- **Assertion Query and Request Protocol:** defines messages and processing rules for requesting existing assertions by reference or querying for assertions by subject and statement type.
- **Authentication Request Protocol:** defines messages and processing rules for obtaining assertions containing authentication statements to establish a security context at one or more relying parties.

- **Artifact Resolution Protocol:** provides a mechanism by which SAML protocol messages can be transported in a SAML binding by reference instead of by value. Both requests and responses can be obtained by reference using this specialized protocol. Regardless of the protocol message obtained, the result of resolving an artifact is treated exactly as if the message so obtained had been sent originally in place of the artifact.
- **Name Identifier Management Protocol:** defines messages and processing rules by which an identity provider or a service provider, after establishing a name identifier for a principal, may change the value and/or format of the name identifier or it may indicate that a name identifier will no longer be used to refer to the principal. This protocol is typically not used with "transient" name identifiers, since their value is not intended to be managed on a long term basis.
- **Single Logout Protocol:** provides a message exchange protocol by which all sessions provided by a particular session authority are near-simultaneously terminated. The single logout protocol is used either when a principal logs out at a session participant or when the principal logs out directly at the session authority. This protocol may also be used to log out a principal due to a timeout.
- **Name Identifier Mapping Protocol:** defines messages and processing rules by which an entity that shares an identifier for a principal with an identity provider may obtain a name identifier for the same principal in a particular format or federation namespace.

The **Assertion Query and Request Protocol** basically gathers <AuthnQuery>, <AttributeQuery> and <AttributeQuery> elements to form request messages. These resulting messages are nearly the same as the AuthenticationQuery, AttributeQuery and AuthorizationDecisionQuery SAML 1 messages respectively. In response to one of these SAML-defined query messages, assertions returned by a SAML authority contain a <saml:Subject> element that strongly matches the <saml:Subject> element found in the query.

The **Authentication Request Protocol** allows a requester to ask to an identity provider for an assertion with an authentication statement, by sending it it an <AuthnRequest> message that describes the properties that the resulting assertion needs to have to satisfy its purpose. The requester might not be the same as the subject being authenticated.

When using the <Scoping> element the requester specifies the identity providers trusted to authenticate the subject, as well as limitations and context

related to proxying of the <AuthnRequest> message to subsequent identity providers by the responder. If an identity provider that receives an <AuthnRequest> has not yet authenticated or cannot directly authenticate the presenter, but believes that the he has already been authenticated to another identity provider, it may respond to the request by issuing a new <AuthnRequest> on its own behalf to be presented to the other identity provider. The original identity provider is termed the proxying identity provider. Upon the successful return of a <Response> to the proxying provider, the enclosed assertion equivalent is used to authenticate the presenter so that the proxying provider can issue an assertion of its own in response to the original <AuthnRequest>, completing the overall message exchange

In the **Artifact Resolution Protocol** the <ArtifactResolve> message is used to request a SAML protocol message be returned in an <ArtifactResponse> message by specifying an artifact that represents the SAML protocol message.

The original transmission of the artifact is governed by the specific protocol binding that is being used. The <ArtifactResolve> message is normally signed or otherwise authenticated and integrity protected by the protocol binding used to deliver the message. It contents an <Artifact> element that has the artifact value that the requester received and now wishes to translate into the protocol message it represents.

In the **Name Identifier Management Protocol** a provider sends a <ManageNameIDRequest> message with the current name identifier for a subject to inform the recipient of a changed in that name or to indicate the termination of the use of it. The <ManageNameIDRequest> message is normally signed or otherwise authenticated and integrity protected by the protocol binding used to deliver the message. The recipient of a <ManageNameIDRequest> message responds with a<ManageNameIDResponse> message. If the current name identifier sent in the request is not recognized by recipient the responding provider responds with an error message. This response message is also normally signed or otherwise authenticated and integrity protected by the protocol binding used to deliver the message.

With the **Single Logout protocol** a principal invokes the single logout process at a session participant or at a session authority. If it is invoked in a session participant then, the session participant sends a <LogoutRequest> message to the session authority that provided the assertion containing the authentication statement related to that session at the session participant. Then the session authoriry sends a <LogoutRequest> message to each session participant to which it provided assertions containing authentication statements under its current session

with the principal, with the exception of the session participant that sent the <LogoutRequest> message to the session authority if it is the case. At receiving a <LogoutRequest> each session participant invalidates the principal session as well as the session authoriry. After successfully terminating the principal sessions, the session participants respond to the session authority with a <LogoutResponse> message. Both the <LogoutRequest> and <LogoutResponse> messages are normally signed or otherwise authenticated and integrity protected by the protocol binding used to deliver the message.

The **Name Identifier Mapping Protocol** allows requesting an alternate name identifier for a principal from an identity provider sending a <NameIDMappingRequest> message. This message indicates the identifier and associated descriptive data that specify the principal as currently recognized by the requester and the responder and the requirements regarding the format and optional name qualifier for the identifier to be returned. This message is normally signed or otherwise authenticated and integrity protected by the protocol binding used to deliver the message. After receiving a <NameIDMappingRequest> message an identity provider responds with a <NameIDMappingResponse> message with the identifier and associated attributes that specify the principal in the manner requested, usually in encrypted form. This message too is normally signed or otherwise authenticated and integrity protected by the protocol binding used to deliver the message. If the identity provider does not recognize the principal identified in the request, it responds with an error.

5.3.4. Profiles

5.3.4.1. SSO Profiles

SAML 2.0 defines five SSO profiles:

- Web Broser SSO Profile:
- Enhanced Client or Proxy (ECP) Profile:
- Identity Provider Discovery Profile:
- Single Logout Profile:
- Name Identifier Management Profile:

5.3.4.1.1. Web Browser SSO Profile

The Web Browser SSO Profile comprises the two profiles defined in SAML 1: the Browser/Artifact Profile and the Browser/Post Profile. In the scenario supported by the SAML 2 Web Browser SSO Profile, a web user either accesses a resource at a service provider, or accesses an identity provider such that the service provider and desired resource are understood or implicit. The web user authenticates (or has already authenticated) to the identity provider, which then produces an authentication assertion or an artifact. Then the service provider consumes the assertion to establish a security context for the web user or, in the case of having received an artifact, dereferences such artifact and gets the assertion. During this process, a name identifier might also be established between the providers for the principal, subject to the parameters of the interaction and the consent of the parties. To implement this scenario, a profile of the SAML Authentication Request protocol is used, in conjunction with the HTTP Redirect, HTTP POST and HTTP Artifact bindings.

5.3.4.1.2. Enhanced Client or Proxy (ECP) Profile

The ECP profile specifies interactions between enhanced clients or proxies and service providers and identity providers. An ECP is a client or an HTTP proxy that satisfies the following two conditions:

- It has, or knows how to obtain, information about the identity provider that the principal associated with the ECP wishes to use, in the context of an interaction with a service provider. This allows a service provider to make an authentication request to the ECP without the need to know or discover the appropriate identity provider
- It is able to use a reverse SOAP (PAOS) binding as profiled here for an authentication request and response. This enables a service provider to obtain an authentication assertion via an ECP that is not otherwise (i.e. outside of the context of the immediate interaction) necessarily directly addressable nor continuously available. It also leverages the benefits of SOAP while using a well-defined exchange pattern and profile to enable interoperability. The ECP may be viewed as a SOAP intermediary between the service provider and the identity provider.

Figure 29 shows the processing flow in the ECP profile.

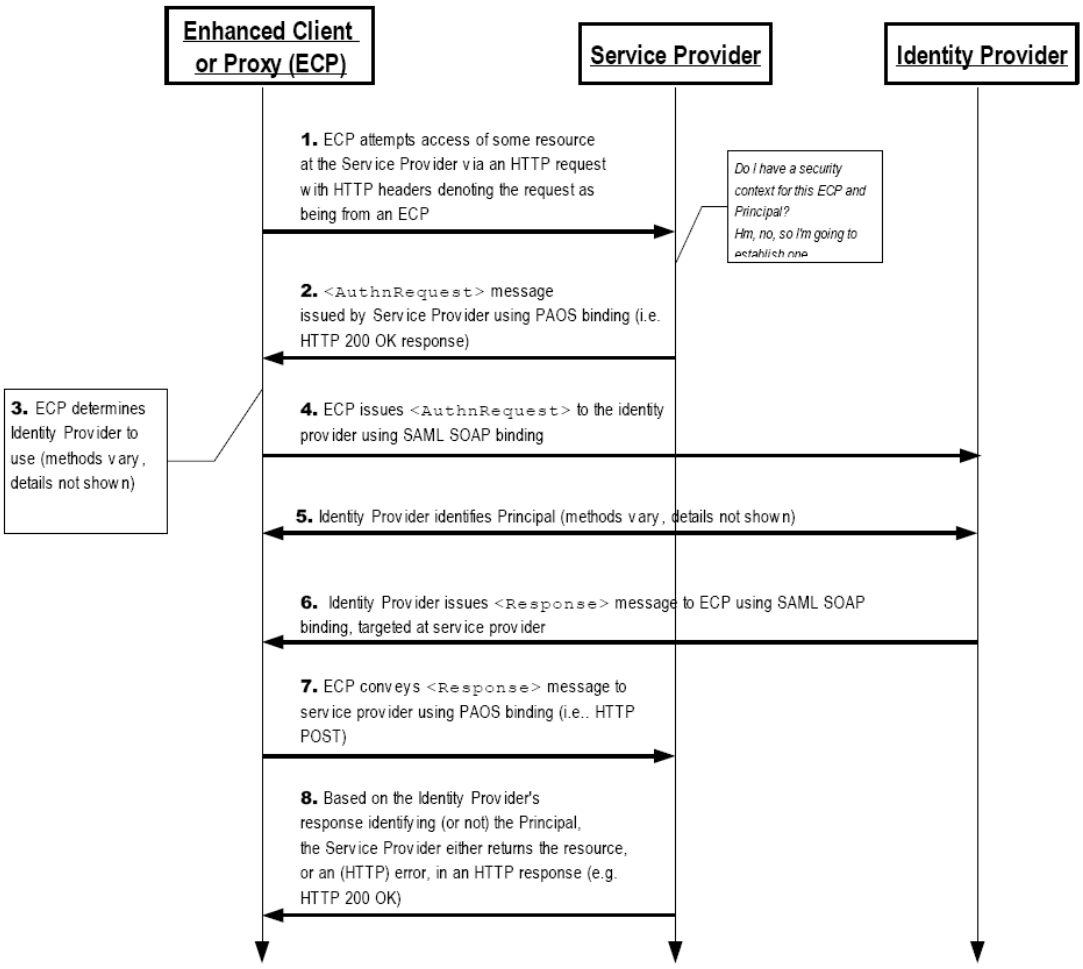

Figure 29: Enhanced Client or Proxy Profile

In step 1, the Principal, via an ECP, makes an HTTP request for a secured resource at a service provider, where the service provider does not have an established security context for the ECP and Principal. Next, in step 2, the service provider issues an <AuthnRequest> message to the ECP, which is to be delivered by the ECP to the appropriate identity provider. In step 3, the ECP obtains the location of an endpoint at an identity provider for the authentication request protocol that supports its preferred binding. The means by which this is accomplished is implementation-dependent and so no specified in SAML 2 though it may use the SAML identity provider discovery profile described in Section 4.3. Afterwards, in step 4, the ECP conveys the <AuthnRequest> to the identity provider identified in step 3 using a the SAML SOAP binding. In step 5, the Principal is identified by the identity provider. In step 6, the identity provider issues a <Response> message, using the SAML SOAP binding, to be delivered by the ECP to the service provider. The message either indicates an error, or includesan authentication assertion or more. Next, in step 7, the ECP conveys the <Response>

message to the service provider using the PAOS binding. Finally**,** in step 8, having received the <Response> message from the identity provider, the service provider either establishes its own security context for the principal and return the requested resource, or responds to the principal's ECP with an error.

5.3.4.1.3. Identity Provider Discovery Profile

The Identity Provider Discovery Profile allows a service provider to discover which identity providers a principal is using with the Web Browser SSO profile. The discovery profile relies on a cookie that is written in a domain that is common between identity providers and service providers in a deployment. The domain that the deployment predetermines is known as the common domain in this profile, and the cookie containing the list of identity providers is known as the common domain cookie.

The name of the cookie used is "_saml_idp" and its format value is a set of one or more base-64 encoded URI values separated by a single space character.

5.3.4.1.4. Single Logout Profile

Once a principal has authenticated to an identity provider, the authenticating entity may establish a session with the principal. The identity provider may subsequently issue assertions to service providers or other relying parties, based on this authentication event; a relying party may use this to establish *its own* session with the principal. In such a situation, the identity provider can act as a session authority and the relying parties as session participants. At some later time, the principal may wish to terminate all his established sessions (with the IdP and all the SPs he had established one). This process can be initiated either at the session authority, or at an individual session participant. The profile allows the protocol to be combined with a synchronous binding, such as the SOAP binding, or with asynchronous "front-channel" bindings, such as the HTTP Redirect, POST, or Artifact bindings.

Figure 30 illustrates the basic template for achieving single logout:

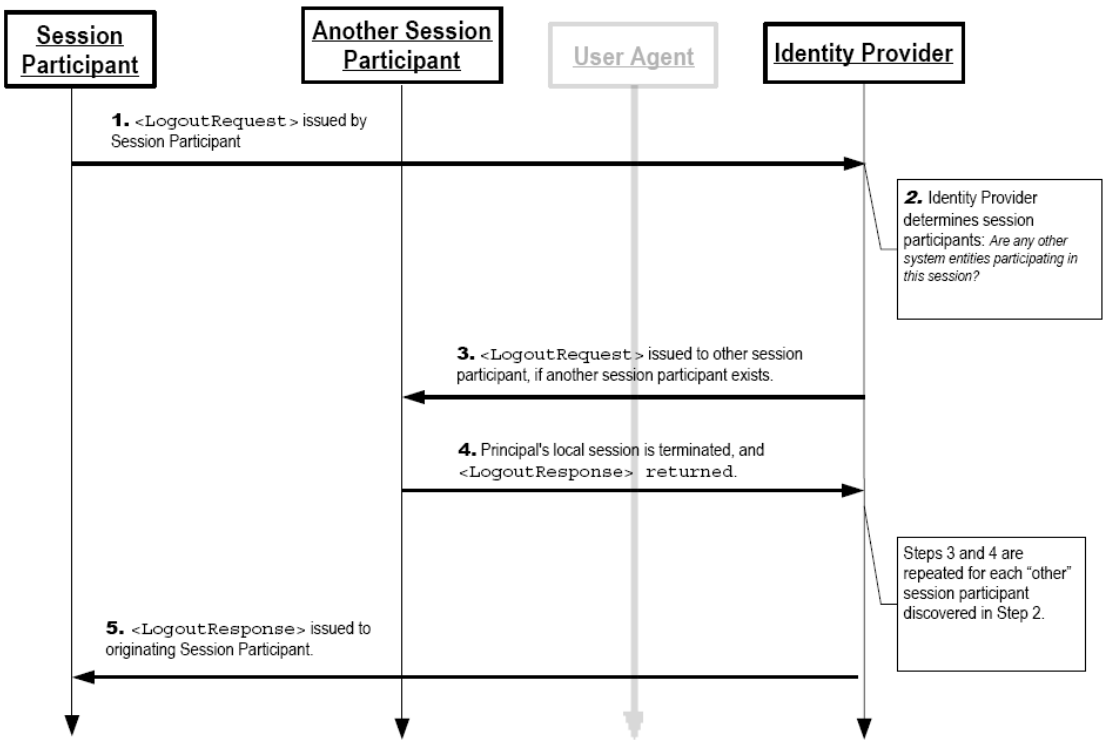

Figure 30: Single Logout Profile

The grayed-out user agent illustrates that the message exchange may pass through the user agent or may be a direct exchange between system entities, depending on the SAML binding used to implement the profile.

In step 1, the session participant initiates single logout and terminates a principal's session(s) by sending a <LogoutRequest> message to the identity provider from whom it received the corresponding authentication assertion. The request may be sent directly to the identity provider or sent indirectly through the user agent. It is recommended that, if the message is sent by a session participant, to use an asynchronous binding, such as the HTTP Redirect, POST, or Artifact bindings, to send the request to the identity provider through the user agent so the identity provider then may propagate any required logout messages to additional session participants using either a synchronous or asynchronous binding. The use of an asynchronous binding for the original request is preferred because it gives the identity provider the best chance of successfully propagating the logout to the other session participants during step 3. Otherwise, if the the session participant uses a synchronous binding, such as the SOAP binding, to send the request directly to the identity provider then the identity provider propagates any required logout messages to additional session participants using a synchronous binding. If the HTTP POST or Redirect binding are used then the <LogoutRequest> message is

always signed. If the SAML SOAP binding is used the requester has to authenticate itself to the identity provider, either by signing the <LogoutRequest> or using any other binding-supported mechanism.

In step 2, the identity provider uses the contents of the <LogoutRequest> message (or if initiating logout itself, some other mechanism) to determine the session(s) being terminated. If there are no other session participants, the profile proceeds with step 5. Otherwise, steps 3 and 4 are repeated for each session participant identified. Next, in step 3, the identity provider issues a <LogoutRequest> message to a session participant or session authority related to one or more of the session(s) being terminated. The request may be sent directly to the entity or sent indirectly through the user agent (if consistent with the form of the request in step 1). Afterwards in step 4, a session participant or session authority terminates the principal's session(s) as directed by the request (if possible) and returns a <LogoutResponse> to the identity provider. The response may be returned directly to the identity provider or indirectly through the user agent (if consistent with the form of the request in step 3). If the identity provider used a synchronous binding the response is returned directly to complete the synchronous communication. Otherwise, if the identity provider used an asynchronous binding then the <LogoutResponse> (or artifact) may be returned through the user agent to the identity provider's single logout service response endpoint or using a syncronous binding. Finally in step 5, the identity provider issues a <LogoutResponse> message to the original requesting session participant. The response may be returned directly to the session participant or indirectly through the user agent (if consistent with the form of the request in step 1).

5.3.4.1.5. Name Identifier Management Profile

In the scenario supported by the Name Identifier Management profile, an identity provider has exchanged some form of persistent long-term identifier for a principal with a service provider, allowing them to share a common identifier for some length of time. Subsequently, the identity provider may wish to notify the service provider of a change in the format and/or value that it will use to identify the same principal in the future. Finally, one of the providers may wish to inform the other that it will no longer issue or accept messages using a particular identifier. To implement these scenarios, a profile of the SAML Name Identifier Management protocol is used.

The profile allows the protocol to be combined with a synchronous binding, such as the SOAP binding, or with asynchronous "front-channel" bindings, such as

the HTTP Redirect, POST, or Artifact bindings. A front-channel binding may be required, for example, in cases in which direct interaction between the user agent and the responding provider is required in order to effect the change.

Figure 31 shows the processing flow in the Name Identifier Management profile

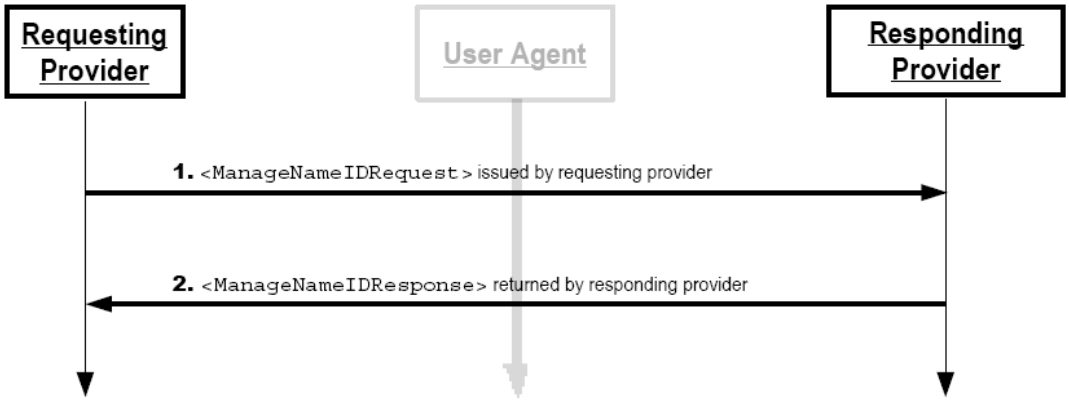

Figure 31: Name Identifier Management Profile.

The grayed-out user agent illustrates that the message exchange may pass through the user agent or may be a direct exchange between system entities, depending on the SAML binding used to implement the profile.

In step 1, an identity or service provider initiates the profile by sending a <ManageNameIDRequest> message to another provider that it wishes to inform of a change. The request may be sent directly to the responding provider using the SAML SOAP binding or sent indirectly through the user agent using the HTTP Redirect, POST or Artifact Binding. In step 2, the responding provider (after processing the request) issues a <ManageNameIDResponse> message to the original requesting provider. The response may be returned directly to the requesting provider or indirectly through the user agent the same way as in step 1 and, of course, consistent with the form of the request in step 1.

This SAML 2.0 profile includes the ID-FF Name Identifier Mapping Profile and the ID-FF Identity Termination Notification Profile.

5.3.4.2. Artifact Resolution Profile

This profile describes the use of the Artifact Resolution protocol with a synchronous binding, such as the SOAP binding.

Figure 32 illustrates the basic template for the artifact resolution profile.

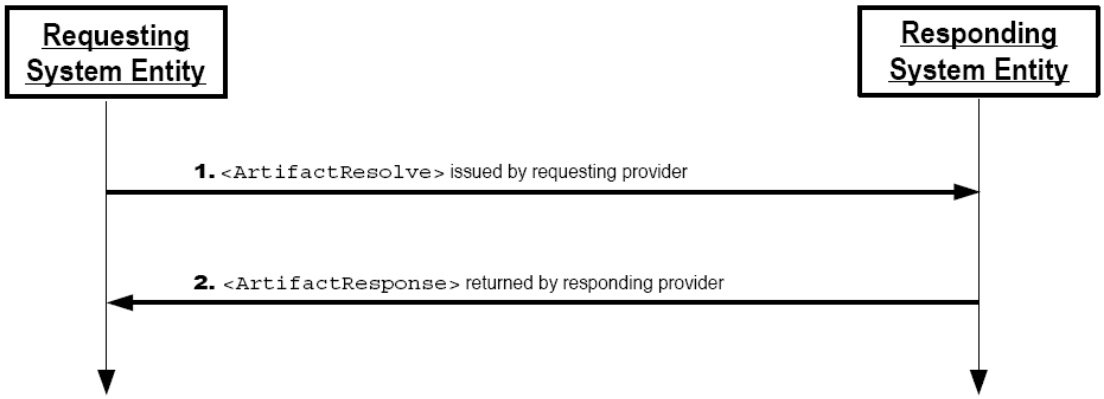

Figure 32: Artifact Resolution Profile

In step 1, a requester initiates the profile by sending an <ArtifactResolve> message to an artifact issuer. Next, in step 2, the responder (after processing the request) issues an <ArtifactResponse> message to the requester.

5.3.4.3. Assertion Query/Request Profile

This profile describes the use of the Assertion Query and Request Protocol with a synchronous binding, such as the SOAP binding.

Figure 33 illustrates the basic template for the query/request profile.

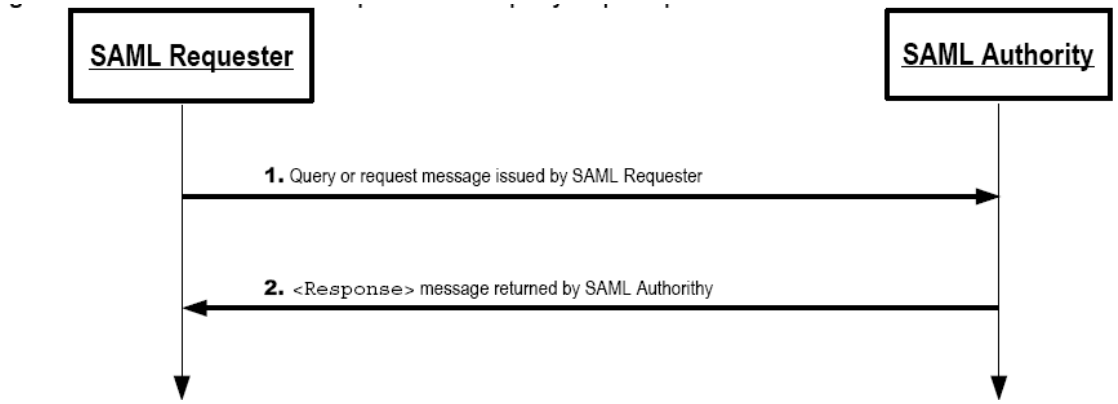

Figure 33: Assertion Query/Request Profile.

In step 1, a SAML requester initiates the profile by sending an <AssertionIDRequest>, <SubjectQuery>, <AuthnQuery>, <AttributeQuery>, or <AuthzDecisionQuery> message to a SAML authority.The SAML requester MUST use a synchronous binding, such as the SOAP binding to send the request directly to the identity provider. It is recommended that the requester authenticates itself

to the SAML authority either by signing the message or using any other binding-supported mechanism. In step 2, the responding SAML authority (after processing the query or request) issues a <Response> message to the SAML requester.

5.3.4.4. Name Identifier Mapping Profile

This profile describes the use of the Name Identifier Mapping protocol with a synchronous binding, such as the SOAP binding, and additional guidelines for protecting the privacy of the principal with encryption and limiting the use of the mapped identifier.

Figure 34 illustrates the basic template for the name identifier mapping profile.

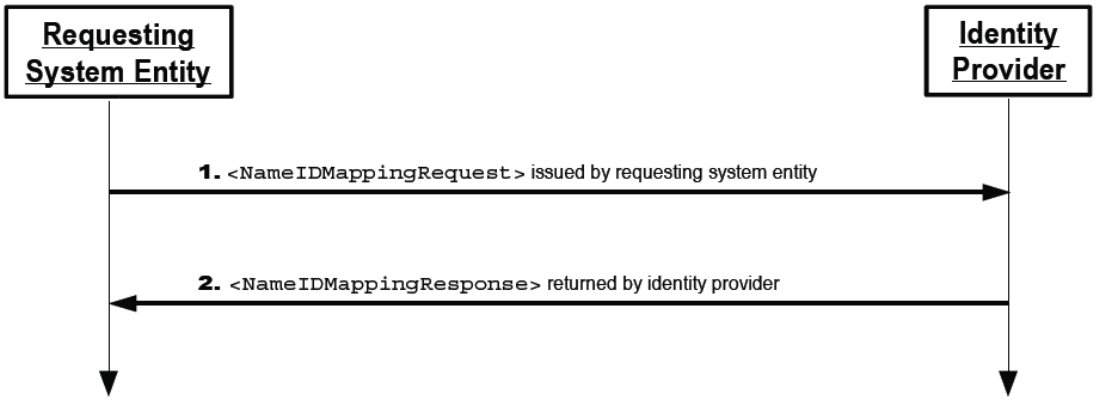

Figure 34: Name Identifier Mapping Profile

In step 1 a requester issues a <NameIDMappingRequest> message to an identity provider's name identifier mapping service endpoint. The requester uses a synchronous binding, such as the SOAP binding, to send the request directly to the identity provider. The requester authenticates itself to the identity provider, either by signing the request or using any other binding-supported mechanism. Then the identity provider processes the <ManageNameIDRequest> message and afterwards returns a <NameIDMappingResponse> message containing an appropriate status code to the requester to complete the SAML protocol exchange.The responder authenticates itself to the requester too.

5.3.4.5. SAML Attribute Profile

SAML 2.0 defines four different Attribute profiles:

- Basic Attribute Profile
- UUID Attribute Profile
- DCE PAC Attribute Profile
- XACML Attribute Profile

5.3.4.5.1. Basic Attribute profile

The Basic attribute profile specifies simplified, but non-unique, naming of SAML attributes together with attribute values based on the built-in XML Schema data types, eliminating the need for extension schemas to validate syntax.

5.3.4.5.2. UUID Attribute Profile

The UUID attribute profile standardizes the expression of UUID values as SAML attribute names and values. It is applicable when the attribute's source system is one that identifies an attribute or its value with a UUID.

UUIDs (Universally Unique Identifiers), also known as GUIDs (Globally Unique Identifiers), are used to define objects and subjects such that they are guaranteed uniqueness across space and time. UUIDs were originally used in the Network Computing System (NCS), and then used in the Open Software Foundation's (OSF) Distributed Computing Environment (DCE). Recently GUIDs have been used in Microsoft's COM and Active Directory/Windows 2000/2003 platform.

A UUID is a 128 bit number, generated such that it should never be duplicated within the domain of interest. UUIDs are used to represent a wide range of objects including, but not limited to, subjects/users, groups of users and node names.

5.3.4.5.3. DCE PAC Attribute Profile

The DCE PAC attribute profile defines the expression of DCE PAC information as SAML attribute names and values. It is used to standardize a mapping between the primary information that makes up a DCE principal's identity and a set of SAML attributes. This profile builds on the UUID attribute profile defined above.

5.3.4.5.4. XACML Attribute Profile

SAML attribute assertions may be used as input to authorization decisions made according to the OASIS eXtensible Access Control Markup Language (XACML) standard specification. Since the SAML attribute format differs from the XACML attribute format, there is a mapping that must be performed. The XACML attribute profile facilitates this mapping by standardizing naming, value syntax, and additional attribute metadata. SAML attributes generated in conformance with this profile can be mapped automatically into XACML attributes and used as input to XACML authorization decisions.

5.3.5. SAML 2 Metadata

Like in ID-FF, SAML profiles require agreements between system entities regarding identifiers, binding support and endpoints, certificates and keys, and so forth. A metadata specification is useful for describing this information in a standardized way. SAML metadata defines an extensible metadata format for SAML system entities, organized by roles that reflect SAML profiles. Such roles include that of SSO Identity Provider, SSO Service Provider, Affiliation, Attribute Authority, Attribute Requester, and Policy Decision Point.

5.4. WS-Federation

WS-Federation is composed of three functional parts:

- **Web Services Federation Language:** defines how different security realms broker identities, user attributes and authentication between Web services.
- **Passive Requestor Profile:** describes how federation helps provide identity services to HTTP 1.1-based Web browsers, Web-enabled cell phones and devices. The Web browser mechanisms describe how the WS-* messages (e.g. WS-Trust's RST and RSTR) are encoded in HTTP messages such that they can be passed between resources and Identity Provider/ Security Token Service parties by way of a Web browser client.
- **Active Requestor Profile:** describes how federation helps provide identity services to applications based on SOAP and other smart clients. The Web service requestors are assumed to understand the WS-Security and

WS-Trust mechanisms and be capable of interacting directly with Web service providers.

While SAML and Liberty specifications have as security tokens SAML assertions, WS-Federation uses the security tokens defined in WS-Security to assert claims. WS-Federation specification does not use the same terminology as SAML and Liberty. The concepts protocol, binding and profile are not used or are used with different meaning. Also the term security token service (STS) is inherited from the WS-Security and WS-Trust specifications. A security token service is defined as a Web service that issues security tokens. That is, it makes assertions based on evidence that it trusts, to whoever trusts it. In the WS-Federation an identity provider is seen as an extension of a security token service and sometimes referenced to as IdP/STS. The part of the service provider that manages the security tokens received is called the resource security token.

5.4.1. Security tokens

To assert claims WS-Federation uses WS-Security security tokens. WS-Security defines four types of general tokens:

- **UserName Token:** The <wsse:UsernameToken> element is introduced as a way of providing a username.
- **Binary Security Token:** Binary security tokens (e.g., X.509 certificates and Kerberos tickets) or other non-XML formats require a special encoding format for inclusion. The <wsse:BinarySecurityToken> element defines two attributes that are used to interpret it. The ValueType attribute indicates what the security token is, for example, a Kerberos ticket. The EncodingType tells how the security token is encoded, for example Base64Binary.
- **EncryptedData Token:** In certain cases it is desirable that the token included in the <wsse:Security> header be encrypted for the recipient processing role. In such a case the <xenc:EncryptedData> element is used to contain a security token and included in the <wsse:Security> header. That is this specification defines the usage of <xenc:EncryptedData> to encrypt security tokens contained in <wsse:Security> header.
- **XML Token:** For security tokens based on XML, the extensibility of the <wsse:Security> header allows for these security tokens to be directly inserted into the header.

In some cases, however, it is more efficient to not return the token, but return a handle to the token along with the proof information. Requestors can then send messages to services secured with the proof token but only passing the token reference, just as SAML and Liberty Artifact.

5.4.2. WS-Trust

WS-Trust defines, among other things, what in SAML and in Liberty would be called a protocol. This "protocol" defines the request and response messages and how they need to be processed for acquiring security tokens.

A requestor sends a request, and if the policy permits and the recipient's requirements are met, then the requestor receives a security token response. This process uses the <wst:RequestSecurityToken> (RST) and <wst:RequestSecurityTokenResponse> (RSTR) elements respectively. Both messages are normally signed. These elements are passed as the payload to specific WSDL ports that are implemented by security token services. In general, the requestor is not required to parse the returned token.

When requesting and returning security tokens additional parameters can be included in requests, or provided in responses to indicate server-determined (or used) values. If a requestor specifies a specific value that isn't supported by the recipient, then the recipient can either fault with a wst:InvalidRequest (or a more specific fault code), or it can return a token with their chosen parameters that the requestor may then choose to discard because it doesn't meet their needs.

The requesting and returning of security tokens can be used for a variety of purposes. In WS-Trust bindings define how this framework is used for specific usage patterns and has no relation with SAML or Liberty bindings. Some of the bindings that are useful for WS-Federation are:

- **Issuance Binding:** defines specific options and processing rules for issuing new security tokens.
- **Renewal Binding:** defines specific options and processing rules for renewing a security token sent in the request message. That is, a previously issued token with expiration is presented and the same token is returned with new expiration semantics.

- **Validation Binding:** defines specific options and processing rules for the evaluation of a specific security token validity.

WS-Federation does not define authentication contexts like SAML 2 and Liberty does. Nevertheless the WS-Trust specification defines the wst:AuthenticationType parameter to indicate a desired type of authentication. However, neither pre-defined values nor authentication classes are specified.

5.4.2.1. WS-Trust Extension

One important extension in WS-Trust is the specification of Referent Tokens. In some cases it is more efficient to not return the token, but return a handle to the token along with the proof information. Requestors can then send messages to services secured with the proof token but only passing the token reference. The recipient is then responsible for obtaining the actual token. Obviously Referent Tokens are equivalent to SAML and Liberty Artifacts. They are defined with the <fed:ReferenceToken ...> element.

5.4.3. WS-Federation Services

WS-Federation describes four services that enable federation operations such as single sign-out, the use of pseudonyms etc. These services are very similar to its corresponding SAML 2.0 and ID-FF 1.1 and 1.2 Profiles.

5.4.3.1. Sign-Out

In the typical use case, federated sign-out messages will be generated by the principal terminating a session, either at the IdP/STS or at one of the resource providers (or its STS) accessed during the session. There are two primary flows for these messages. In one case they are effectively chained through all the STSs involved in the session; that is, a mechanism is used (if available) by the IP/ STS to send sign-out messages to all the other STSs in a sequential manner by causing each message to cause the next message to occur in sequence resulting in a message back to itself either on completion or at each step to orchestrate the process. The second approach is to require the IP/ STS to send sign-out messages

to all the other token services and target services in parallel (those that it knows about).

The chained (sequential) approach has been found to be fragile since if one of the messages fails to complete its local processing and does not pass the sign-out message on the sign-out, notification does not reach all the involved parties. For this reason, is recommended that implementations employ the parallel approach. If the session is terminated at a resource provider, it cleans up any local state and then sends a sign-out message to the IP/STS. The latter sends parallel sign-out messages to all the other STSs.

All the sign out messages are normally signed by the requestor to prevent tampering and to prevent unauthorized sign-out messages.

5.4.3.2. Attribute Service

Web services often need to be able to obtain additional data related to service requestors to provide the requestor with a richer experience. This is achieved having an attribute service that requesters and services use to access this additional information. In many cases, the release of this information about a service requestor is subject to authorization and privacy rules and access to this data is only granted to authorize services for any given attribute. Attribute stores most likely exist in some form already in service environments using service-specific protocols (e.g. such as LDAP). An attribute service provides the interface to this attribute store. **Figure 35** below illustrates the conceptual namespace of an attribute service. WS-Federation specification makes no proposals or requirements on the organization of the data.

Principals represent any kind of resource, not just people. Consequently, the attribute mechanisms can be used to associate attributes with any resource, not just with identities. Principals and resources may have specific policies that are required when accessing and managing their attributes.

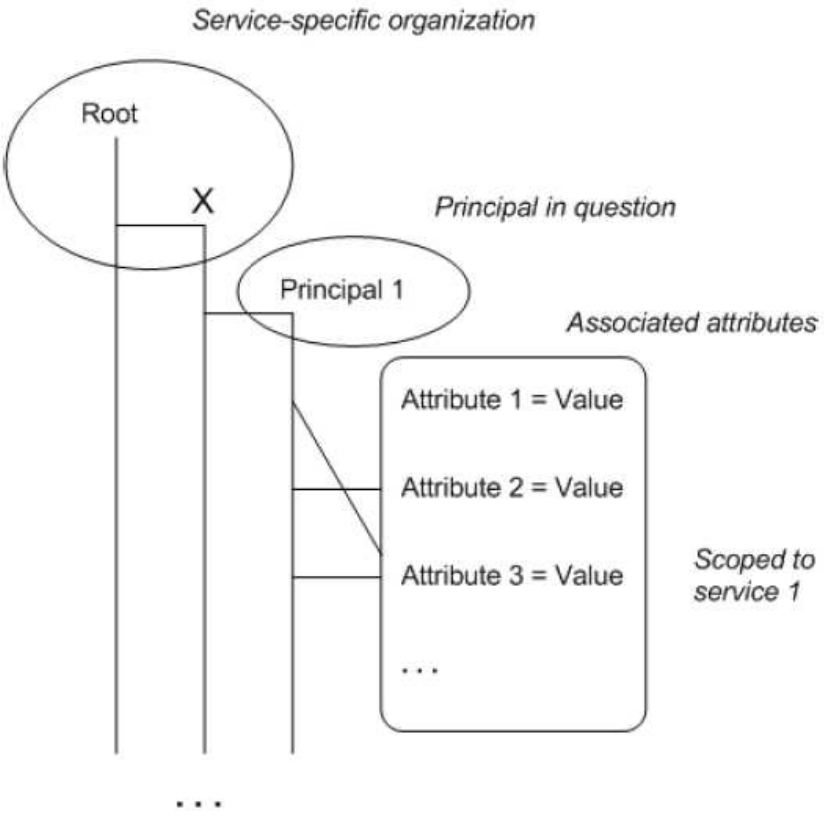

Figure 35: Attribute Service organization

At a high-level, attribute processing uses the same mechanisms defined for security token service requests and responses. That is, redirection is used to issue requests to attribute services and subsequent redirection returns the results of the attribute operations. All communication occurs with the standard HTTP GET and POST methods using redirects to automate the communication.

5.4.3.3. Pseudonym service

In WS-Federation the pseudonym service is seen as a special type of attribute service which maintains alternate identity information and optionally associated tokens for principals.

A pseudonym can have zero or more associated security tokens that allow an IdP to directly return the appropriate token for specified scopes. For example, when Fred.Jones requested a token for Fabrikam123.com, his IdP could have returned the Freddo identity directly allowing the requestor to pass this to

Fabrikam123. When performing operations on a pseudonym store it is highly recommended to filter the scope of the operation. This can be accomplished using the <fed:FilterPseudonyms> element or with some WS-ResourceTransfer extensions to WS-Transfer.

Pseudonyms are requested from a pseudonym service using the GET WS-Transfer method. This method is uded for fetching one-time snapshot of the representation of a resource. To return a pseudonym this one is embedded in the body of the GET response message in a <fed:Pseudonym> element.

To update a pseudonym in a pseudonym service the PUT WS-Transfer is used. This method is used for updating a resource by providing a replacement representation. The update operation allows one or more pseudonyms to be added in the same request specifying a filter. If a filter is not specified, then the PUT impacts the full pseudonym set.

Pseudonyms are created in a pseudonym service using the WS-Resource create operation. As well as the updating operation, this one also allows one or more pseudonyms to be added with one request with the help of a filter. If a filter is not specified, then the CREATE affects the full pseudonym set.

Pseudonyms are deleted in a pseudonym service using the WS-Transfer PUT operation. In this operation filters are used to restrict the scope of the PUT to only remove pseudonym information corresponding to the filter. If a filter is not specified, then the PUT impacts the full pseudonym set.

5.4.3.4. Authorization Service

An authorization service is an STS that operates in a decision brokering process. That is, it receives a request for a token to access another service. Such a service can be separated from the target service or it can be co-located. The authorization service determines if the requested party can access the indicated service and, if it can, issues a token with the allowed rights at the specified service. In order to make the authorization decision, the authorization service must ensure that the requestor has presented and proven the claims required to access the target service indicated in the request. Thus the authorization service constructs a table of name/value pairs representing the claims required by the target service.

5.4.4. Passive Requestor Profile

The federation model described in WS-Federation builds on fountains established by WS-Security and WS-Trust. Typical Web client requestors can not perform the message security and token operations defined in WS-Federation. Consequently a Passive Requestor Profile was defined, describing the mechanisms for requesting, exchanging and issuing security tokens within the context of a Web requestor.

This Profile could be compared to the Browser Redirect and the Browser Post SAML 2.0 Bindings, since both are also used here. WS-Federation defines a variety of parameters, some optional, some obligatory, for both HTTP 1.1 GET and POST methods. For example the *wa* parameter is required and it specifies the action to be performed. Another one is the *wreply* and it is an optional parameter and it contains the URL to which responses are directed.

WS-Federation describes a Home Realm Discovery Service to determine the location of the IdP/STS that manages the identity of a requestor. This Service is explained in section 5.4.4.1. Another way to know the IdP/STS of the requestor is through the *whr* parameter. This parameter is used to indicate the IdP/STS address for the requestor and it may be specified directly as a URL or indirectly as an identifier. In the case of an identifier the recipient is expected to know how to translate this (or get it translated) to a URL. When the *whr* parameter is used, the resource, or its local IdP/STS, typically removes the parameter and writes a cookie to the client browser to remember this setting for future requests. Then, the request proceeds in the same way as if it had not been provided. Note that this serves roughly the same purpose as federation metadata for discovering IdP/STS locations previously discussed.

The return of security tokens is done by passing an HTTP from. To return the tokens, this profile embeds a <wst:RequestSecurityTokenResponse> element as specified in the WS-Trust specification.

In many cases the IdP/STS to whom the request is being made, will prompt the requestor for information or for confirmation of the receipt of the token. As a result, the IdP/STS can return an HTTP form to the requestor who then submits the form using an HTTP POST method. This allows the IdP/STS to return security token request responses in the body rather than embedded in the limited URL query string. However, in some circumstances interaction with the requestor may not be required (e.g. cached information). In these circumstances the IdP/STS have several options:

- Use a form anyway to confirm the action

- Return a form with script to automate and instructions for the requestor in the event that scripting has been disabled
- Use HTTP GET and return a pointer to the token request response (unless it is small enough to fit inside the query string)

WS-Federation recommends using the POST method as the GET method requires additional state to be maintained and complicates the cleanup process whereas the POST method carries the state inside the method.

One additional parameter defined in security tokens responses is the *wresultptr*. This parameter specifies a URL to which an HTTP GET can be issued. The result is a document that contains the issuance result. This parameter serves roughly the same purpose as the WS-ReferenceToken mechanism explained in section 5.4.2.1.

Single Sign-In, Single Sign-Out, Attribute Requests and Pseudonymous requests may be used in the Passive Requestor Profile. Its fluxes diagrams are nearly the same as SMAL 2.0 and Liberty diagrams.

5.4.4.1. Home Realm Discovery

WS-Federation does not specify a normative way of discovering the *home realm* of the requestor; however, the following mechanisms are mentioned:

- **Fixed:** The home realm is fixed or known
- **Requestor IP:** The home realm is determined using the requestor's IP address
- **Prompt:** The user is prompted (typically using a Web page)
- **Discovery Service:** A service is used to determine the home realm
- **Shared Cookie:** A shared cookie from a shared domain is used (out of scope)

The *Home* Realm Discovery Service is a Web-based service that, through implementation-specific methods may be able to determine a requestor's home realm without user interaction. A resource or resource IdP/STS can redirect to a discovery service to attempt to determine the home realm without prompting the user. The discovery service redirects back to the URL specified by the *wreply* parameter. If the discovery service was able to determine the home realm, it is returned using the *whr* parameter. This value can be used to lookup the URL for the user's IdP/STS for properly redirecting the token request. If the discovery service is

unable to determine the home realm then the *whr* parameter is not specified and the home realm must be discovered by other means.

5.4.5. Active Requestor Profile

An active requestor is an application (possibly a Web browser) that is capable of issuing (and receiving) SOAP messages such as those described in WS-Security and WS-Trust. The primary goal of the Active Requestor Profile is to define mechanisms for federation of identity, authentication, and authorization information as applied to active requestors.

Security tokens are requested and returned using the <RequestSecurityToken> and < RequestSecurityTokenResponse> messages respectively defined in the WS-Trust specification.

Neither Single Sign-Out nor Single Sign-Out are typical actions done by active requestors.

5.4.5.1. Single Sign On

Active requestors can obtain the policy ahead of time or via error responses from services. In general, requestors are required to obtain a security token (or tokens) from their Identity Provider (or STS) when they authenticate themselves. The IdP/STS generates a security token for use by the federated party. This is done using mechanisms defined in WS-Trust. In some scenarios, the target service acts as its own IdP/STS so communication with an additional service isn't required. Otherwise the requestor may be required to obtain additional security tokens from service-specific or service required identity providers or security token services.

Figure 36: Single Sign On Active Requestor Profile below illustrates one possible flow.

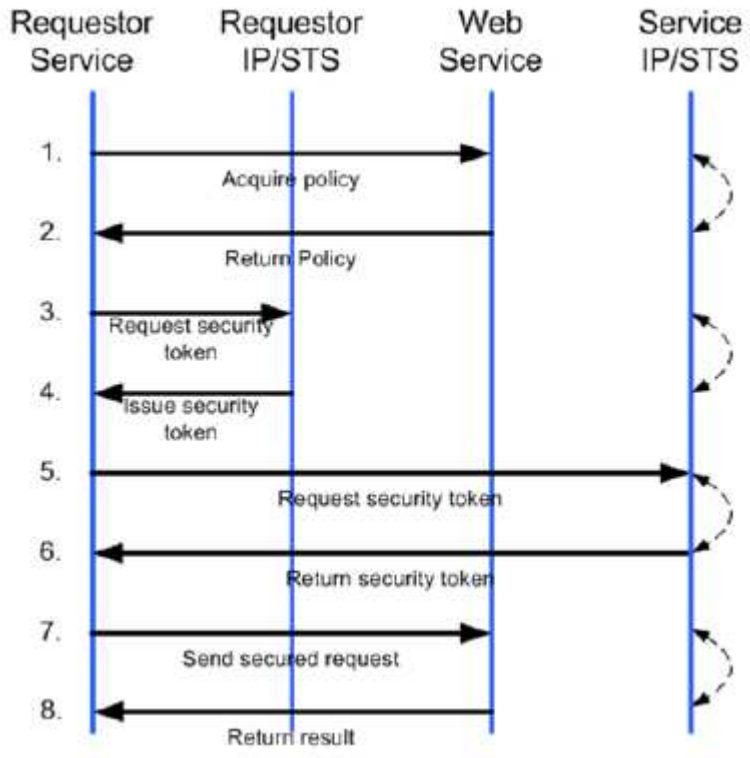

Figure 36: Single Sign On Active Requestor Profile

In step one the requestor service acquires the Service policy using the mechanisms defined in WS-MetadataExchange. Next, in step two the web service returns its policy using also WS-MetadataExchange mechanisms. In step three the requestor requests a security token from its IdP/STS using a <RequestSecurityToken> message. Afterwards, in step four, the IdP/STS returns a security token using a <RequestSecurityTokenResponse> message. In step five the requestor requests a security token from the Web services IdP/STS for the target Web service using also <RequestSecurityToken>. After that, in step 6 the Web service's IdP/STS returns a token using the (<RequestSecurityTokenResponse>) message. In step 7 the requestor sends the request to the service attaching and securing the message using the issued tokens as described in WS-Security. Finally, in step eight, the web service issues a secured reply using its security token.

5.4.5.2. Sign-Out

WS-Federation Single Sign-Out in active requestors does not introduce any important difference to the SAML 2.0 or ID-FF Single Sign-Out flow. In situations where federated sign-out messages are desirable with active requestors, the

requestor's IdP/STS needs to keep track of the realms to which it has issued. When the sign-out is received at the requestor's IP/STS, the requestor's IdP/STS is responsible for issuing federated sign-out messages to interested and authorized parties. The exact mechanism by which this occurs is up to the IP/STS, but WS-Federation specification strongly advises to use the single sign out messages defined in its specification. These messages are identified for having the element <SignOut>. When a federated sign-out message is received at a realm, the realm needs to cleanup any cached information and delete any associated state with the requestor.

5.4.5.3. Attributes and Pseudonyms

For active requestors, attribute and pseudonyms services are identified via WS-Policy as described in WS-Federation. Attributes are requested and updated using messages specific to the attribute services as explained in the Attribute Service section. This specification doesn't mandate a specific attribute store technology.

Pseudonyms are requested and updated using the messages and mechanisms described in the Pseudonymous Service.

5.4.6. WS-Federation Metadata

Like in SAML 2.0 and ID-FF specifications, in WS-Federation participation in a federation requires knowledge of metadata such as policies and potentially even WSDLs and schemas for the services within the federation. Additionally, in many cases mechanisms are needed to identify the Identity Provider, security token services, and attribute/pseudonym services for the target of a given policy. Therefore, WS-Federation metadata serves the same purpose as SAML 2.0 and ID-FF metadata.

5.5. OpenID

OpenID is structured completely different than any of the previously explained specifications. It does not define bindings, or protocols or profiles. Compared to SAML, Liberty or WS-Federation, OpenID is a lightweight specification that defines a simple and small federation model.

OpenID uses different terminology than the other specifications. The SP is always referred as Relying Party and the Identity Provider as the OpenID Provider (OP).

The last version of OpenID is composed of three specifications:

- OpenID Authentication 2.0
- OpenID Attribute Exchange 1.0
- OpenID Simple Registration Extension 1.0

The OpenID Simple Registration Extension is an extension to the OpenID Authentication protocol that allows for very light-weight profile exchange. This extension is not going to be explained further in this project since it is a very reduced and simplified version of the authentication specification.

5.5.1. OpenID Authentication 2.0

The OpenID Authentication specification specifically and concretely addresses Web Single Sign-On use cases. It is a single monolithic specification binding together the specification of message formats, protocol initiation, identity provider discovery protocol, user identifier definition, and SSO protocol definition. OpenID 2.0 specifies a concrete web SSO protocol, OP discovery protocol, user identifier format, and an extensibility mechanism, for example for attribute exchange, security considerations, and backwards compatibility in a single draft specification.

5.5.1.1. Security Assertions

OpenID security assertions are comprised of a set of key-value pairs, without explicit message-independent delineation. OpenID does not define an explicitly delineated security assertion object, thus limiting reusability in other protocol contexts. The keys and values permit the full Unicode Character Set. When the keys and values need to be converted to/from bytes, they are encoded using UTF-8.

5.5.1.1.1. HTTP Encoding

When a message is sent to an HTTP server, it is encoded using a form encoding specified in Section 17.13.4 of HTML 4.0.1. Specification.

All of the keys in the request message have the prefix "openid.". This prefix prevents interference with other parameters that are passed along with the OpenID

Authentication message. The "openid.mode" parameter allows the recipient of the message to know what kind of message it is processing. If "openid.mode" is absent, the party processing the message assumes that the request is not an OpenID message.

5.5.1.2. Communication Types

5.5.1.2.1. Direct Communication

Direct communication is initiated by a Relying Party to an OP endpoint URL. It is used for establishing associations and verifying authentication assertions.

5.5.1.2.2. Indirect Communication

In indirect communication, messages are passed through the User-Agent. This can be initiated by either the Relying Party or the OP. Indirect communication is used for authentication requests and authentication responses.

There are two methods for indirect communication: HTTP redirects and HTML form submission. Both require that the sender know a recipient URL and that the recipient URL expect indirect messages. The initiator of the communication chooses which method of indirect communication is appropriate depending on capabilities, message size, or other external factors.

In **HTTP Redirect** data can be transferred by issuing a 302, 303, or 307 HTTP Redirect to the end user's User-Agent. In **HTML Form Redirection**

A mapping of keys to values can be transferred by returning an HTML page to the User-Agent that contains an HTML form element whose "action" attribute value is the URL of the receiver.

In the case of a malformed request, or one that contains invalid arguments, the OpenID Provider redirects the User-Agent to the "openid.return_to" URL value if the value is present and it is a valid URL.

5.5.1.3. Initiation and Discovery

OP/IDP discovery is concretely specified in the OpenID specification. It is based directly on the notion that OpenID Identifiers are normalizable into either URLs or XRIs.

XRIs are dereferenciable into either an XRDS which in turn points to the user's OP/IDP or a web page containing an HTML <Link> tag pointing to the user's OP/IDP. XRDS is an XML format for discovery of metadata about a resource. XRDS are basically the OpenID metadata together with the established associations explained in the following section.

Figure 37 the flow for the OpenID Initiation and Discovery process

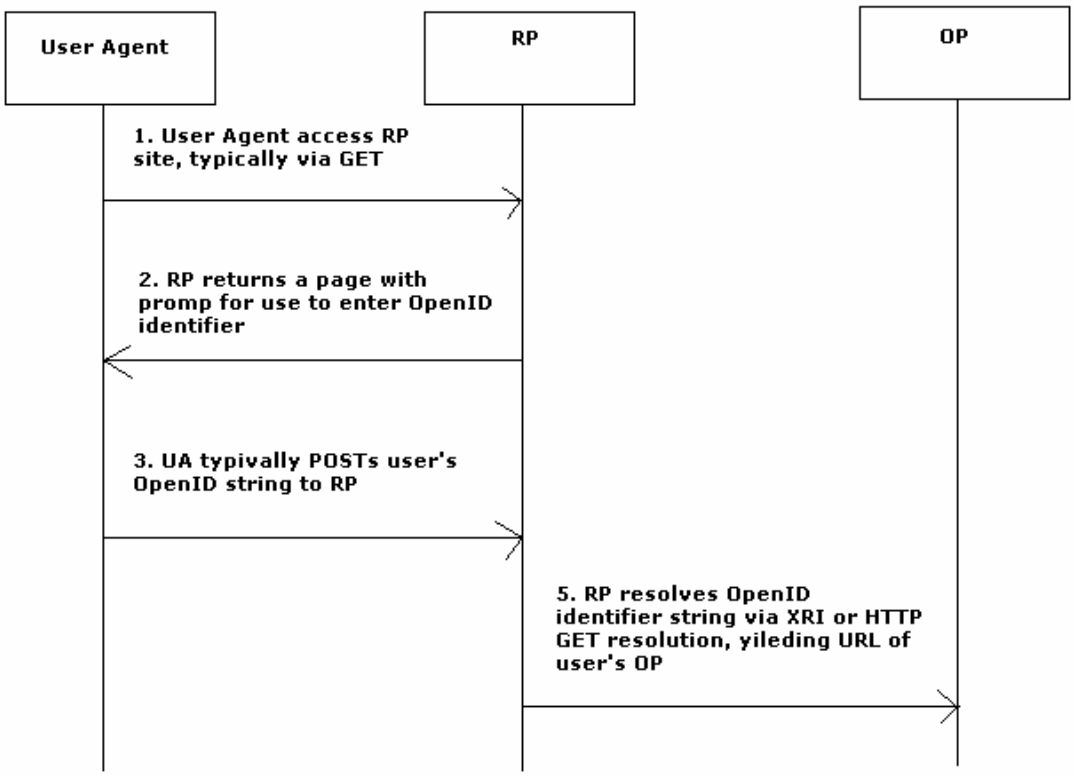

Figure 37: OpenID OP Discovery

5.5.1.4. Establishing associations

OpenID relies upon establishing associations for exchanging keying material between an RP and an OP/IDP. An association session is initiated by a direct request from a Relying Party to an OP.

The "association establishment sub-protocol" is based directly on Diffie-Hellman (DH) key exchange to securely transmit shared secrets and is actively performed between the RP and the OP/IDP. Diffie-Hellman key exchange is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an over an insecure communications channel. It yields a symmetric session key, which is subsequently used to sign messages exchanged between the parties.

This eases deployment- and run-time considerations in that no mutual beforehand configuration is required between a relying party and an OP in order to exchange mutually-verifiable signed messages.

5.5.1.5. Authentication

Once the Relying Party has successfully performed discovery and, in some cases, created an association with the discovered OP, it can send an authentication request to the OP to obtain an assertion. An authentication request is an indirect request.

When an authentication request comes via indirect communication, the OP understands that an authorized end user wishes to complete the authentication. In case of authentication success the OP sends a positive assertion to the Relying Party. Otherwise, if the OP is unable to identify the end user or the end user does not or cannot approve the authentication request, the OP sends a negative assertion to the Relying Party as an indirect response.

If there was no established association between the RP and the OP before the authentication process direct signature verification with the OP is done. To have the signature verification performed by the OP, the Relying Party sends a direct request to the OP. To verify the signature, the OP uses a private association that was generated when it issued the authentication response.

5.5.2. OpenID Attribute Exchange 1.0

OpenID Attribute Exchange is an OpenID service extension for exchanging identity information between endpoints. It defines two message types for transferring attributes: fetch and store. **Fetch** retrieves attribute information from an OpenID Provider, while **store** saves or updates attribute information on the OpenID Provider. Both messages originate from the Relying Party and are passed to the OpenID Provider via the user agent as defined in the OpenID authentication 2.0 specifications as indirect communication.

All attributes are identified by an URI, which is used for referring to property values, and its values are in UTF8. Any individual site or consortium of sites may define their own attribute types with agreements on the syntax and semantics of their associated attribute values.

5.5.2.1. Fetch

A fetch message is identified with the element "openid.ax.mode=fetch_request". With a fetch message a Relaying Party can ask for both required and optional (only if those are available in the OP) attributes as well as the number of values that it wishes to receive for specific attributes.

The fetch response message supplies the information requested in the fetch request if this is available. This response is identified with the element openid.ax.mode="fetch_response".

5.5.2.2. Store

A store message is identified with the element "openid.ax.mode=store_request". With a store message a Relaying Party can send a collection of attributes with its new values or the number of values that are going to be sent for each attribute.

The store response informs if the operation has been succeed. In case of success the response message will have the openid.ax.mode element with the value "store_response_success". Otherwise it will contain "store_response_failure".

6. Federation implementations

Once studied the current federation specifications some software product had to be chosen. One of the premises in e-Catalunya platform is that all software used must be open source. That narrowed the research of all the possible available implementations. To choose one, the following aspects were taken into consideration:

- Is the implementation an API or a software product?
- What specifications are being implemented?
- How many and what kind of federations use this implementation?
- What organization is behind this implementation?
- How good is its documentation?

In this chapter there is a small overview of the main characteristics of the most important studied products as well as the final chosen product.

6.1. Lasso and Authentic

Lasso is a Library that manages Liberty Alliance exchanges and that can be used to add support for Liberty Alliance protocols to a service. Lasso is a library written in C Language and has complete bindings for the C, C++, C #, ColdFusion, Java, PHP and Python languages. It is build on the top of powerful XML libraries (libxml2, XMLSec).

Authentic is a Liberty Alliance identity provider aiming to address a broad range of needs, from simple to complex setups. It provides Single Sign-on (SSO), Single Logout (SLO) and attribute exchange. Authentic is commonly run inside Apache web server. It is released under the GNU GPL license. Liberty Alliance conformance: support of ID-FF 1.2, ID-WSF 1.1 (2.0 work in progress) and SAML 2.0. Support of different users databases: LDAP V3 (Active Directoy), Postgresql, MySQL etc.

Both Lasso and Authentic are developed by Entr'ouvert, a free software company whose commercial activity was built around eGovernment and eVoting solutions. Entr'ouvert is a member of Libre-entreprise, a corporate network, and benefits from several years of experience concerning eGovernment.

Lasso is used in some well known organizations and projects in Francesuch as "Carte de Vie Quotidienne" and "Services de Vie Quotidienne" projects; the Lasso Python bindings are used to run the identity provider

(macommune.identification.svq.fr) and two service providers (macommune.formulaires.svq.fr and macommune.consultation.svq.fr).

Entr'ouvert offers support thought its mailing lists, thought they do not seem to be used daily.

Lasso has good documentation and easy to find and use, although some of the documents are only in French.

6.2. ZXID

ZXID is a C library that implements the full SAML 2.0 stack and aims to implement in the near future all popular federated ID management protocols such as Liberty ID-FF 1.2 and WS-Federation. It is based on schema based code generation, resulting in an accurate implementation. SWIG is used to offer scripting language interfaces such as Perl, PHP, and Python, as well as Java. It can act as SP, IdP, WSC, and WSP. ZXID author is Sampo Kellomäki.

ZXID project has currently six outputs:

- **Libzxid**: A C library for supporting SAML 2.0, including federated Single Sign-On (SSO)
- **Zxid**: A C program that implements a SAML Service Provider (SP) as a CGI script
- **Net::SAML**: Perl module wrapping libzxid. Also zxid.pl, that implements SP in mod_perl environment, is supplied.
- **php_zxid**: A PHP extension that wraps libzxid. Also supplied: zxid.php that implements SP in mod_php environment.
- **libzxidjni.so**: A Java JNI extension that wraps libzxid. Also supplied: zxid.java that implements SP as a CGI script. zxidhlo.java demonstrates use under servlet engine, e.g. Tomcat.
- **mod_auth_saml**: An Apache httpd auth module that does SAML SSO. No programmatic integration required, just alter your Apache httpd.conf

There is no information about current federations being implemented with ZXID, although it seems very popular on the internet. Nevertheless it does not look like is a solution adopted by big companies or organizations and the fact that there is no organization behind it makes it seem less reliable than other solutions.

Currently there are no mailing lists to offer support and the only means to getting help is by mailing the author.

6.3. Jboss Federated SSO Framework

The JBoss SSO Framework is a collection of components that can be integrated within existing web applications to create a federation of trusted web sites. The framework has support for SAML assertions but it does not implement SAML specification per se. The system consists of the following components:

- **Federation Server:** A Federation Server is used for securely propagating the Federation Token across web applications located in different security domains. This is the component that enables the federation use cases.
- **Token Marshalling Framework:** A flexible/pluggable Java API to marshal/unmarshal a Federation Token. It is in this component that SAML assertions may be received or requested. Moreover it ships with other tokens compliant marshallers such as Kerberos.
- **Identity Connector Framework:** A flexible/pluggable Java API to connect to central identity stores. The system ships with a Provider to connect to LDAP based Identity Stores.

JBoss is a division of Red Hat. It specializes in open-source middleware software. The company profits from a service-based business model. JBoss pioneered the Professional Open Source business model where the core developers of projects make a living and offer their services.

There is not much information about current federations implemented by JBOSS, but the support tools offered, such as mailing lists and forums are quiet used and updated daily. Thus seems to point that JBOSS federated SSO is broadly used.

The documentation found is scarce and not comprehensive enough. It lacks general information to understand easily the whole framework and how its components interact between them.

6.4. Higgins

Higgins is framework that enables users and other systems to integrate identity, profile, and relationship information across multiple heterogeneous systems. Higgins is not centred in a federation model per se and tries to reach further. Higgins main aspect is its user centric approach. In a general idea, Higgins allows people to store their digital identities and profile information in places of their choice and to share the stored information with companies and other parties in a controlled fashion. Higgins unifies all identity interactions (regardless of protocol/format) under a common user interface metaphor called i-cards. Each i-card represents a different digital identity, and is based in the physical cards one would have, such a credit bank card, a library card etc. That way users dispose of more control over personal information distributed across external information silos. The i-cards manager has to be installed in the client side and Higgins offers different pug-ins for web-browsers. That implies that the user will have to install something in the client machine in order to use this identity management, which it may be an inconvenience, but it reduces the risk of phishing or other attacks.

In the federation aspect Higgins provides two identity providers web services. The first is a Security Token Service (STS) that supports WS-Trust. The second supports SAML2. Higgins also includes the service provider libraries necessary to enable websites and systems to request and accept information cards. Developers can incorporate this relying party code into their applications and web sites to make it easier for users to login. Underneath the selector apps and web services just mentioned lies an identity management abstraction layer. This layer consists of a software framework that can be extended using plugins. The lowest layer of this framework is the Identity Attribute Service (IdAS) that provides an interoperability and portability across "silos" of identity data. IdAS provides read/write access to a wide variety of data sources including LDAP directories and can be extended using plugins called "Context Providers". The IdAS service maps identity and social network data into the Higgins Context Data Model. IdAS makes it possible to mash up identity and social network data across highly heterogeneous data sources including directories, relational databases, and social networks.

The initial code for the Higgins Project was written by Paul Trevithick and later became part of SocialPhysics.org, a collaboration between Paul and Mary Ruddy, of Parity Communications, Inc., and John Clippinger, a senior fellow at the Berkman Center of the Harvard Law School. Currently Higgins, is part of the Eclipse Foundation. Mary and Paul are the project co-leads. IBM and Novell's participation

in the project was announced in early 2006. Higgins has received technology contributions from IBM and Novell as well as from several other firms and individuals.

Higgins is more centred in the user-centric aspects than in the federation model, thought it supports it. Higgins reaches further than the other implementations giving more control to the user. Nevertheless the user centric feature was out of scope in this project and it was not one of the objectives to be achieved.

Higgins documentation lacks in technical aspects, whereas to find general and business information about it is really easy. As a technical support Higgins offers mailing lists.

6.5. OpenSSO

The Open Web Single Sign-On Project, referred to as OpenSSO, is based on the source code for Sun Java System Access Manager and Sun Java System Federation Manager, two commercial identity and federation products offered by Sun Microsystems, Inc.

Sun Microsystems, Inc is a multinational vendor of computers, computer components, computer software, and information technology services and is a proponent of open systems in general and Unix in particular and a major contributor of open source software

OpenSSO consists of Open Federation Library and Service Provider Interfaces implementations using OpenSSO APIs. The Open Federation Library supports ID-FF 1.1 and 1.2, ID-WSF 1.0 and 1.1, SAML 1.0, 1.1 and 2.0.

To achieve extensibility and customizability, a list of Service Provider Interfaces are provided in each standard implementation to satisfy different deployment use cases. A set of common Service Provider Interfaces used by all components are also defined to integrate with existing authentication, configuration, session, logging and data store infrastructure.

OpenSSO offers implementations of the framework in a war format deployable on any J2EE web container as well as easy to setup samples for getting the system up and running.

OpenSSO is a very complete project with a big community of people working on it. Support is offered by an IRC Channel, mailing lists, wikis and forums.

This solution is used in quite a large number of enterprises that would rather use open source, such as "Medavie Blue Cross”.

The main problem with OpenSSO is the vast number of documents and information offered that makes the understanding of the project vary confusing. To understand the whole project one needs to dedicate a great effort before doing anything just reading documentation which half of it will not be useful later on.

6.6. Shibboleth

Shibboleth is not only an implementation of a federation solution but a new specification based completely on SAML adding to it some extensions. Initially Shibboleth was built on SAML 1.1 and 1.2. As seen in chapter 4 these versions of SAML have some limitations compared to later specifications. Thus Shibboleth extended SAML 1.1 and 1.2 specifications adding new functionalities that were required. Some of them are the specification of an optional *WAYF (Where Are You From) service or the extension of the* Browser/POST Profile and Browser/Artifact Profile to allow them to be service provider initiated. The WAYF is used by the SP to determine the user's preferred IdP, with or without user interaction. The WAYF is essentially a proxy for the authentication request passed from the SP to the SSO service at the IdP.

SAML 2.0 covers all these Shibboleth extensions and adds much more functionalities. Hence the last version of Shibboleth (Shibboleth 2.0 released as a stable this same year) is built primarily on SAML 2.0 adding some new features.

Shibboleth 2.0 implementation consists of several individual components: the identity provider (IdP), service provider (SP), and discovery service (DS).

Shibboleth is a project of Internet2. Internet2 is a non-profit consortium which develops and deploys advanced network applications and technologies, for education and high-speed data transfer purposes. It is led by over 200 universities and partners with many affiliate members and corporate members drawn from companies in the publishing, networking and other technology industries.

There are lots of federations implemented using Shibboleth, most of them inside the university or the government context. It is easy to find information about them and some documents explaining how Shibboleth was installed and configured in these organizations are available. Some of these organizations are UK Federation, In Common, SWITCHaai etc.

Internet2 offers a good and complete documentation, both for the Shibboleth specification and the Shibboleth software implementation. Also it offers mailing lists for user support that are daily used and consulted.

6.7. PAPI

PAPI structure and components are based on Shibboleth specification simplifying it. PAPI consists of three independent elements: the authentication server (AS) and the point of access (PoA) and the -wide Point of Access (GPoA). The AS has the same role as and IdP and the PoA the same as an SP. The GPoA is used for combining a group of PoAs with similar access policies and it intends to simplify AS-PoA interactions.

PAPI is a product of RedIRIS. RedIRIS is the national research and education network (NREN) for Spain. It is part of Red.es, which also administers and oversees the .es national Top-level domain. Most Spanish universities and research centres are interconnected through RedIRIS, currently totalling about 260 institutions. RedIRIS also acts as an Internet Service Provider for affiliated institutions, through links with Telia and Global Crossing. As a national NREN, RedIRIS is connected to the high-speed European GÉANT2 backbone, similar to the US-based Abilene Network.

PAPI is used to protect resources at the RedIRIS Web services, and it is also used by different institutions inside the Spanish Academic Network. A couple of library consortia (each of them grouping several universities in two different regions) are using it for accessing their digital contents, using a PoA for each one of the data sources they are offering, controlled by a consortium GPoA. These (G)PoAs trust the AS of the universities inside each consortium. Pilot PAPI-enabled access services are also offered at several universities for controlling the access of students and professors to digital teaching materials. Several other National Research Networks are using or evaluating the use of PAPI for applications such as a single point authentication service to access digital content and services for inter-institutional access to data and learning contents, such as and. New authentication methods are being experimented and we have demonstrated the applicability of PAPI to several commercial Web-based information access systems from digital content providers.

RedIRIS offers a good set of documentation about PAPI. Documents are written in either English or Spanish.

For user support mailing lists are used. However, the content of these mailing lists can be consulted and it seems that they are hardly used.

6.8. Chosen product

All products are good solutions in their own way and the selection of one depends on the criteria of the search. In the end, in this project, Shibboleth 2.0 was chosen. The main reason for selecting Shibboleth 2.0 is the big community that lies behind it and its popularity. Shibboleth is a software broadly used in important and public organizations, thus making it a very reliable solution. Moreover, it is impressive the huge amount of useful information that may be found on the internet about it.

Furthermore, Shibboleth 2.0 is based on SAML 2.0 specifications which makes it a robust solution and interoperable with other ones. The fact that it is a software application instead of an API has its advantages and its inconveniences. An API gives the developer complete freedom and control in its own implementation and the final product can be more suited and easy adaptable to the end application that is going to be "federated". However, when working in the identity management field, where security is an important aspect, it is good to use software developed by team expertises whose experience is most surely going to assure the presence of security holes. Moreover, the great community that is using Shibboleth makes the detection of bugs or other possible problems easier.

7. Prefederating e-Catalunya

A great part of the work needed to do when building a federation needs to be done before its building. The most important part is the agreements and the circle of trust that are going to be used between the different organizations.

Another aspect to consider is, in case of federating applications already developed and working, if the applications will have to suffer any changes and what kind of changes.

The first part of this chapter does a general overview of some of the aspects that should be considerate before attempting a federation deployment. The second part will focus on how the federation model will be integrated inside the e-Catalunya platform.

7.1. Trust scope

The process of creating a federated authentication trust is much more that what trust protocols and specifications are going to be used. There are a number of legal, business and technical issues that must be clarified before implementation between all the organizations that are going to form the federation.

In the future, if e-Catalunya ends building a federation with some applications of the Generalitat of Catalunya, both responsible organizations will have to discuss and agree wit a particular circle of trust. It is usually recommended to develop a trust scope document with all these agreements and requirements.

The following points are some examples of what normally needs to be defined before deploying a federated model:

- Participants of the federation.
- Identity registration requirements for each party.
    - Are background checks required?
    - What pieces of identification are required?
- Identity provisioning process for each party.
    - How are users provisioned?
        - What is the time frame for provisioning?
    - How are identities archived?
- Identity authentication requirements for each party.
    - Assertions that are going to be used for the SSO.

- What attribute assertions are going to be used?
  - Roles, titles, credentials etc.
  - What is the privacy policy for each of these attributes?
- Authorization requirements for each party.
- Use cases scenarios for each party.
  - What is the role of each party inside the federation model?
- Authentication trust management and security
  - What kind of certificates are going to be used?
  - What Certification Authorities are accepted inside the federation?
  - What messages are going to be signed?
  - What messages are going to be encrypted?

7.2. Changes in e-Catalunya

As explained in the second chapter, e-Catalunya 1.6 is composed of two applications, the platform itself and the phpBB2 forums. Currently, to achieve SSO between these two applications, Java Open Single Sign-On (JOSSO) is used. JOSSO is a single sign-on solution for web applications in the same domain. It is an open source J2EE based software for user authentication and authorization. JOSSO architecture is very similar to the federation models proposed by the different specifications. There is a SSO Gateway, an SSO Agent and a partner application. The **SSO Gateway** represents the SSO server or Identity Provider and provides authentication services to users who need authentication with partner applications. The **SSO Agent** represents the client of the SSO Gateway to which authentication requests are delegated on user protected resource access, managing the whole user authentication flow. It validates SSO sessions and obtains associated user information against the SSO Gateway web services using the SOAP protocol. The SSO Agent corresponds to the Service Provider explained in the federation model. Finally the **partner application** represents a web application or Service Provider that uses SSO Gateway services to authenticate users.

Figure 38 is an overview of the JOSSO architecture.

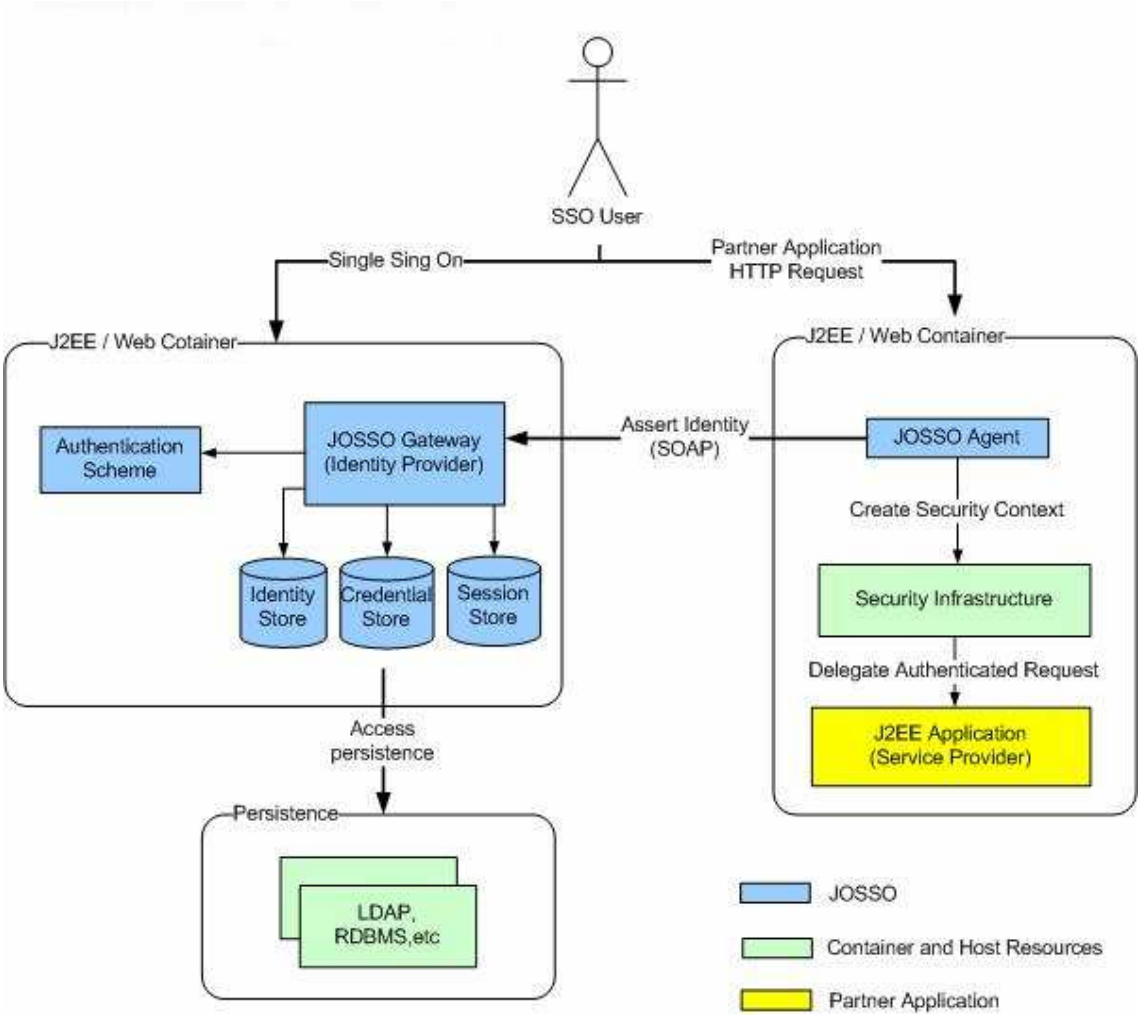

Figure 38: JOSSO architecture.

Based on **Figure 38**, **Figure 39** shows the current situation of JOSSO in the e-Catalunya platform. The JOSSO Agent on the Tomcat Server is implemented as a Valve while the JOSSO Agent controlling the phpBB2 is a php component that intercepts all apache petitions that go to protected areas.

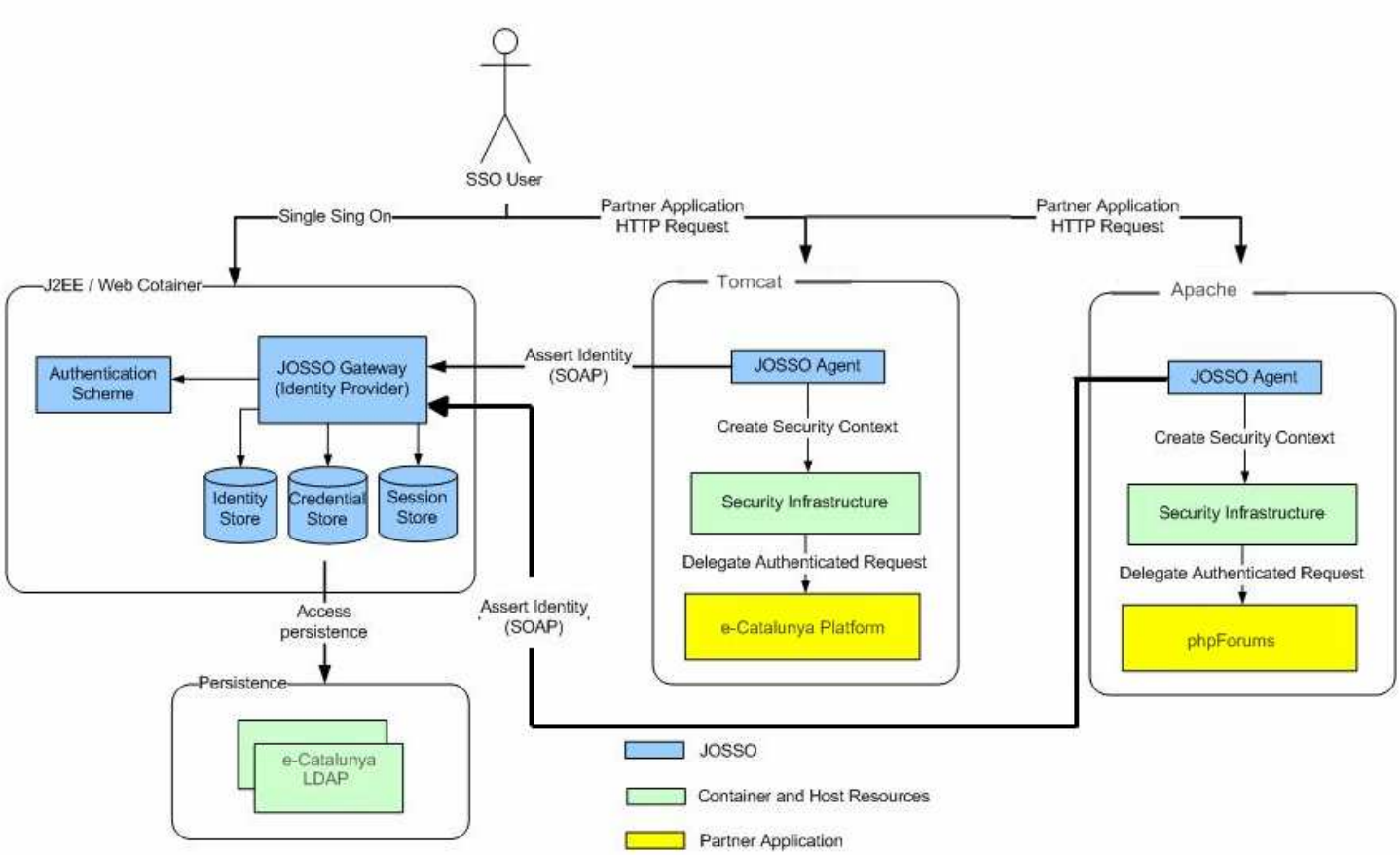

Figure 39: JOSSO in the e-Catalunya platform

In e-Catalunya JOSSO is only responsible for authenticating the users. The authorization access is managed by the platform itself due to the complex system that e-Catalunya presents with private and public areas. As seen in chapter 1 the right of a user to access to a particular area may depend on the membership of this user in the portal/group, the visibility of the portal/group/tool or the user role inside this portal/group.

To do this authorization control the platform needs to know, in case of a user that has already been authenticated, the user identification. In e-Catalunya users are identified by their emails. JOSSO Agents populate the HTTP header REMOTE_USER with the user email so the platform may have access to it and create a user session.

Since Shibboleth may enable SSO between different applications inside the same domain too, it was decided to substitute JOSSO for Shibboleth in the pilot test. Therefore, Shibboleth would not only be used to enable a federation model with e-Catalunya but it would provide the Single Sign-On between the platform and

the phpBB2. Except for deleting the JOSSO system, e-Catalunya would not suffer any other changes.

All the effort of building a federated e-Catalunya would be focused on the Shibboleth 2.0 installation and configuration, but not on modifying the platform itself.

8. Shibboleth and its deployment

Gilead then cut Ephraim off from the fords of the Jordan, and whenever Ephraimite fugitives said, 'Let me cross,' the men of Gilead would ask, 'Are you an Ephraimite?' If he said, 'No,' they then said, 'Very well, say Shibboleth.' If anyone said, 'Sibboleth', because he could not pronounce it, then they would seize him and kill him by the fords of the Jordan. Forty-two thousand Ephraimites fell on this occasion.

Judges 12:5-6, NJB

Shibboleth 2.0 is a complex system with a large number of components. This chapter does not intend to explain all options Shibboleth 2.0 offers nor all of its components. This chapter will do a general overview of Shibboleth focusing on the main aspects that were required to deploy it on the e-Catalunya platform. As Shibboleth is being explained the solutions chosen in this project will be described too.

This chapter is structured in two main parts. When deploying Shibboleth in e-Catalunya two configurations were tested. The first one was incorrect in the platform context, as it is going to be explained, but it was easier to configure than the configuration desired. So to do the initial testing this first configuration was tried. Once it seemed that both the IdP and the SP worked correctly and communicated without problems the second configuration was configure. Therefore, the first and second parts of this chapter will explain the first and second configuration respectively.

8.1. Pilot test

Before starting the detailed explanation of Shibboleth 2.0 deployment it is important to understand the structure of the pilot test.

To simulate a federation at least to parties are required. Depending of the federation objectives both parties may play the roles of Identity Provider and Service Provider against each other or one party may play the role of the Identity Provider while the other plays for the Service Provide. It all depends on the interactions defined in that specific federation.

In this project the latter case was the one that was going to be simulated since there was only one organization playing at all.

To do so the platform was logically "divided" as two different entities. On one side there was the LDAP where all the user data required for the authentication process was stored and in the other there were the rest components that were the responsible for providing services.

Therefore the IdP installation would work with the LDAP to authenticate users while the SP protected the platform content.

8.2. Installation and first configuration

8.2.1. Introduction

Shibboleth is a complex system with a large number of different possible configurations. During its installation and especially during its configuration lots of errors were bound to happen. For this reason the recommended way to do a new deployment of Shibboleth is to rely on some Shibboleth entity (IdP or SP) already installed and working correctly. In case that this scenario could not be available, Shibboleth developer team offers a test service called TestShib for new installations and basic configurations. Thus, if the new installation is an IdP, the best way to do the initial testing could be federating it with some already functional SP or with the test SP offered by TestShib. Otherwise, if the new installation is an SP, the initial testing should be done federating it with an already functional IdP or with the test IdP offered by TestShib. This way it could be known where all errors found were coming from, if the IdP or the SP.

In case that none of above options were available and both SP and IdP had to be installed, configured and tested against each other, patience and precaution are advised.

That fateful situation was met in this thesis. Due to not having any Shibboleth entity already installed and nicely configured to test the newly installed SP and IdP and the firewall configuration that made communication with TestShib impossible both installations had to be tested against each other.

This is the reason why this first configuration was tried before the final one. To try to install and configure both IdP and SP and test them in the end would be really dangerous and complicated. To identify the errors and discover if they were caused by a wrong installation or a wrong configuration could be complicated. Thus it was decide to add an intermediate step where only the installation and a basic configuration could be tested.

In this intermediate configuration the Single Sign-On between the phpBB2 software and the e-Catalunya platform was not going to be implemented. The objectives to achieve were to deploy user authentication in the IdP and a SAML 2.0 Web Browser Single Sign-On (Redirect or Post) between the IdP and the SP. Once this was tested Single Sign-On in the phpBB2 and authorization in the e-Catalunya platform would be deployed and configured.

To deploy both providers it could be recommended to deploy each of them in separate machines to avoid conflict between the configurations of the servers (Apache, Tomcat etc.). Due to some circumstances it was impossible to do so in this project[1]. This final situation did the deployment more complicated than what initially was thought.

To create the virtual machine the VMware Workstation was used. VMware Workstation is a virtual machine software suite for x86 and x86-64 computers from VMware, previously a division of EMC Corporation. This software suite allows users to set up multiple x86 and x86-64 virtual computers and to use one or more of these virtual machines simultaneously with the hosting operating system [wiki].

Our virtual machine had 512 MB of RAM and a hard disk of 8 GB. The operating system installed was openSuse 10.2 to simulate the daniol development environment.

Before installing the SP and IdP, the e-Catalunya platform had to be installed. To do so the ECAT Platform 1.4 install package developed by Albert Miró, member of the e-Catalunya developer team, was used.

8.2.2. IdP installation and first configuration

The Shibboleth Identity Provider 2.0 is a Java web application composed of fairly loosely coupled components. The components are loaded, managed, and inter-connected by Spring framework.

The Spring Framework is an open source application framework for the Java platform. Although the Spring Framework does not enforce any specific programming model, it has become popular in the Java community as an alternative or even an addition to the Enterprise JavaBean (J2EE) model. By design,

[1] The IdP and SP could not be deployed in different machines because each personal computer had only one network interface needed for other related work. The second shot was trying to run two virtual machines in the same computer. But, again, that was not possible since the IdP machine required a minim of 512 MB of RAM and we only disposed of 1 GB of RAM. So when we tried to run one virtual machine of 512 MB and one of 256 MB simultaneously the system performance was not good enough to allow proper working. In consequence both the SP and the IdP were deployed in the same virtual machine.

the framework offers a lot of freedom to Java developers yet provides well documented and easy-to-use solutions for common practices in the industry.

The highest components within the IdP are known as services and they may use each other. Services are built up of smaller components. These smaller components within a service may use each other but may not use components from another service. **Figure 40** attempts to show this.

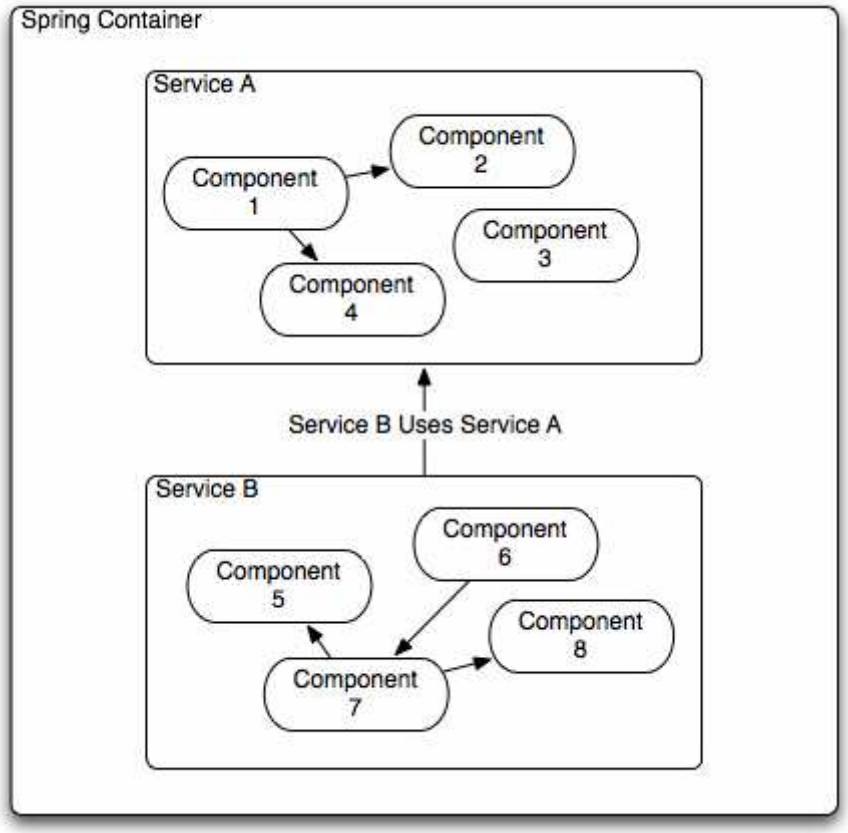

Figure 40: IdP Service-Components structure

Services within the IdP are defined in the *service.xml* file located inside the IDP_HOME/conf directory, in which IDP_HOME is the directory where the IdP has been installed. In this directory is where all configuration files are, except for the metadata file. There are seven services that compose the IdP:

- **Attribute Resolver Service**: responsible for retrieving attributes from data store and then combining, transforming, or otherwise mutating that data into a set of attributes about a user. The usual intent is that all or some of these attributes would be sent back to the IdP client. The configuration file for this service is attribute-resolver.xml.

- **Attribute Filter Service**: filters a collected set of attributes based on a loaded set of policies. These policies reflect which attributes and attribute values a client may retrieve. The configuration file for this service is attribute-filter.xml.

- **SAML 1 Attribute Authority Service:** relies on an attribute resolver, and optionally a filter, service takes a collection of attributes and encodes them into a SAML 1 attribute statement. In addition, if the requesters SAML metadata contains information about which attributes it needs/wants the service will also perform a filtering function so that only those attributes are released. If either the filter service or the attribute information in the metadata is not given then those filtering steps are simply skipped.

- **SAML 2 Attribute Authority Service:** nearly identical to the SAML 1 Attribute Authority, except that it produces SAML 2 attribute statements and also offers a third filtering pass based on attributes requested within the SAML 2 attribute request.

- **Relying Party Configuration Service:** responsible for managing the configurations for various relying party or groups of relying party. This includes the default and anonymous relying party configurations. The relying party configurations also include the configurations for the communication profiles supported by the IdP. The configuration file for this service is relying-party.xml.

- **Handler Manager Service:** manages the various IdP endpoints that may receive messages there is one endpoint per incoming binding and each endpoint may support multiple outgoing bindings. This service also holds the configuration for the error and login handlers which endpoints used by the IdP internally. The configuration file for this service is handler.xml.

- **Servlet Context Attribute Exporter Service:** it does not have an associated Spring application context or any associated components. Instead it simply binds other services into the Servlet context so that they may be retrieved from Servlets, filters, and JSP pages.

In our deployment the SAML 1 Attribute Authority Service was not used since we only used SAML 2.0 assertions.

The IdP had to use a different Tomcat and a different java machine than e-Catalunya since it requires Tomcat 5.5 or greater and J2SE SDK 1.5 or greater.

The IdP can be installed either using an Apache and a Tomcat installation or using a standalone Tomcat installation. In this case the latter was used. The first step to deploy the IdP is to configure Tomcat and the jsdk. The IdP requires Tomcat to have two secured ports. One of the secure ports is for enabling the direct communication with the SP, as opposed to sending messages via the user's browser, during certain operations such as Attribute Query, Artifact Resolution, and Logout. The other is to secure the IdP HTTP connections.

After configuring the Tomcat server the IdP was build using the apache ant tool and afterwards the .war was copied inside the Tomcat webapps directory. After the building process the following information had been generated:

- The IdP's entity ID
- A key pair and self-signed certificate used for signing and encryption (these are not use for HTTP connections to your server)
- The IdP's metadata
- A basic set of IdP configuration files based on this information

Every Shibboleth entity is identified by a unique *entity ID*, sometimes also referred to as a *provider ID*. The entity ID is used by entities to refer to themselves during communications, and to look up metadata for other entities. It is highly desirable that entities are referred to by the same name in every Shibboleth federation of which they are a member, both to simplify configuration and to allow free movement of entities from one federation to another.

By default the entityID generated for the IdP has the following schema: https://*host-name*/idp/shibboleth. Hence in our case the eintityID was https://tokyo.fib.upc.edu/idp/shibboleth.

For each IdP must be two <EntityDescriptor> element in the metadata file: the <IDPSSODescriptor> element and the <AttributeAuthorityDescriptor>. Figure x shows the metadata file generated by the building process with some modifications done such as changing the port number of some of the bindings to direct those services to the correct Tomcat. Some other modifications are going to be explained below.

The certificate included in the metadata file is the one generated by the building process. It is not necessary to use this certificate and it is recommended to

change it for a valid certificate signed by a recognized CA in the production environment. Since this was just a pilot test the generated certificates were used. **Figure 41** shows the generated metadata file with the entityIDs modified.

```
<EntityDescriptor entityID="https://tokyo.fib.upc.edu/idp/shibboleth"
                  xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
                  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                  xmlns:shibmd="urn:mace:shibboleth:metadata:1.0"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <IDPSSODescriptor protocolSupportEnumeration="urn:mace:shibboleth:1.0
urn:oasis:names:tc:SAML:1.1:protocol urn:oasis:names:tc:SAML:2.0:protocol">

        <Extensions>
            <shibmd:Scope regexp="false">upc.edu</shibmd:Scope>
        </Extensions>

        <KeyDescriptor>
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate>
MIIDLzCCAhegAwIBAgIUYgisg0PRxpBGwseMA7aqVtZvWAwwDQYJKoZIhvcNAQEF
BQAwHDEaMBgGA1UEAxMRdG9reW8uZmliLnVwYy5lZHUwHhcNMDgwNjAzMDk0ODI4
WhcNMjgwNjAzMDk0ODI4WjAcMRowGAYDVQQDExF0b2t5by5maWIudXBjLmVkdTCC
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJa6uZ3lyWBy273E4Lwa1L2j
+KmEtJlOKzLL6izn5uHQnpyftUbo3QOLChsDqc456m71kFecmJqfJW1oHNf3w2PT
sLV37xVlRnzBTs51CSYDy4KY2TX+4CdQHLCxNL7Ag1xe3FUkBx5KtqMnJqTEyI88
r/BjIu26nZPD+YlXeILm2nBeUEMYN5R1hofp1UhfeIcJO7CtDzT1uBYT5ynic6dM
zk8TLakoGTckUqG2zR4Hz8NLtqKSrOcYBLG+4a6xlR9le6JhOIYBvI6T8MR+SKw2
DOgjszia3xDBaOFFEsmHH/m9fJzGn2pBWMIB2s8zDt+j9L1+4zGeP3EOiBpI+psC
AwEAAaNpMGcwRgYDVR0RBD8wPYIRdG9reW8uZmliLnVwYy5lZHWGKGh0dHBzOi8v
dG9reW8uZmliLnVwYy5lZHUvaWRwL3NoaWJib2xldGgwHQYDVR0OBBYEFEP51zik

0qpgH3/QfmNL18gcCQzTMA0GCSqGSIb3DQEBBQUAA4IBAQBamVgVwTJqKMpPblZm
dycUL3WAFPlikoNH+THX1CFOh2kWbKTL1W8sS86HvC5yf+b8mP0YJuk183/wOI93
H8sskbLjkJGYvY96F3tG0xCwSAAndJIwmYJrQjA66NIt/Mmhjg8LnlAWk1j3KCi6
6ny3D1Ucr+BxqnJs8RjvulrRN0NgbyHKd26N64PVkh6K3tFSTwTeL76aLCiSAV+7
opMp6kaxZw8/CjThQusD95qOAqUJKDbtIOz8qsKBJMEbsN2g9Ph2mNJDIF2pS3Yy
rCyVvGi61irBIDllfYiL0nAwqHuPnRNd3Tj5r5XBICxf92jgLasB0Lclyq3ogo2U
9364

                    </ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>

        </KeyDescriptor>

        <ArtifactResolutionService
Binding="urn:oasis:names:tc:SAML:1.0:bindings:SOAP-binding"

Location="https://tokyo.fib.upc.edu:8444/idp/profile/SAML1/SOAP/ArtifactResolution"
                                    index="1"/>

        <ArtifactResolutionService
Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"

Location="https://tokyo.fib.upc.edu:8444/idp/profile/SAML2/SOAP/ArtifactResolution"
                                    index="2"/>
```

```
        <NameIDFormat>urn:mace:shibboleth:1.0:nameIdentifier</NameIDFormat>
        <NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified</NameIDFormat>
        <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-
format:transient</NameIDFormat>


        <SingleSignOnService
Binding="urn:mace:shibboleth:1.0:profiles:AuthnRequest"

Location="https://tokyo.fib.upc.edu:8445/idp/profile/Shibboleth/SSO" />


        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST"

Location="https://tokyo.fib.upc.edu:8445/idp/profile/SAML2/POST/SSO" />

        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST-SimpleSign"

Location="https://tokyo.fib.upc.edu:8445/idp/profile/SAML2/POST-SimpleSign/SSO" />

        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect"

Location="https://tokyo.fib.upc.edu:8445/idp/profile/SAML2/Redirect/SSO" />
    </IDPSSODescriptor>

    <AttributeAuthorityDescriptor
protocolSupportEnumeration="urn:oasis:names:tc:SAML:1.1:protocol
urn:oasis:names:tc:SAML:2.0:protocol">

        <Extensions>
            <shibmd:Scope regexp="false">upc.edu</shibmd:Scope>
        </Extensions>

        <KeyDescriptor>
            <ds:KeyInfo>
                <ds:X509Data>

                    <ds:X509Certificate>
MIIDLzCCAhegAwIBAgIUYgisg0PRxpBGwseMA7aqVtZvWAwwDQYJKoZIhvcNAQEF
BQAwHDEaMBgGA1UEAxMRdG9reW8uZmliLnVwYy5lZHUwHhcNMDgwNjAzMDk0ODI4
WhcNMjgwNjAzMDk0ODI4WjAcMRowGAYDVQQDExF0b2t5by5maWIudXBjLmVkdTCC
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJa6uZ3lyWBy273E4Lwa1L2j
+KmEtJlOKzLL6izn5uHQnpyftUbo3QOLChsDqc456m71kFecmJqfJW1oHNf3w2PT
sLV37xVlRnzBTs51CSYDy4KY2TX+4CdQHLCxNL7Ag1xe3FUkBx5KtqMnJqTEyI88
r/BjIu26nZPD+YlXeILm2nBeUEMYN5R1hofp1UhfeIcJO7CtDzT1uBYT5ynic6dM
zk8TLakoGTckUqG2zR4Hz8NLtqKSrOcYBLG+4a6xlR9le6JhOIYBvI6T8MR+SKw2
DOgjszia3xDBaOFFEsmHH/m9fJzGn2pBWMIB2s8zDt+j9L1+4zGeP3EOiBpI+psC
AwEAAaNpMGcwRgYDVR0RBD8wPYIRdG9reW8uZmliLnVwYy5lZHWGKGh0dHBzOi8v
dG9reW8uZmliLnVwYy5lZHUvaWRwL3NoaWJib2xldGgwHQYDVR0OBBYEFEP51zik
0qpgH3/QfmNL18gcCQzTMA0GCSqGSIb3DQEBBQUAA4IBAQBamVgVwTJqKMpPblZm
dycUL3WAFPlikoNH+THX1CFOh2kWbKTL1W8sS86HvC5yf+b8mP0YJuk183/wOI93
H8sskbLjkJGYvY96F3tG0xCwSAAndJIwmYJrQjA66NIt/Mmhjg8LnlAWk1j3KCi6
6ny3D1Ucr+BxqnJs8RjvulrRN0NgbyHKd26N64PVkh6K3tFSTwTeL76aLCiSAV+7
opMp6kaxZw8/CjThQusD95qOAqUJKDbtIOz8qsKBJMEbsN2g9Ph2mNJDIF2pS3Yy
rCyVvGi61irBIDllfYiL0nAwqHuPnRNd3Tj5r5XBICxf92jgLasB0Lclyq3ogo2U
9364
```

```
                    </ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </KeyDescriptor>

        <AttributeService Binding="urn:oasis:names:tc:SAML:1.0:bindings:SOAP-
binding"

Location="https://tokyo.fib.upc.edu:8444/idp/profile/SAML1/SOAP/AttributeQuery" />

        <AttributeService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"

Location="https://tokyo.fib.upc.edu:8444/idp/profile/SAML2/SOAP/AttributeQuery" />

        <NameIDFormat>urn:mace:shibboleth:1.0:nameIdentifier</NameIDFormat>
        <NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified</NameIDFormat>

      <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-
format:emailAddress</NameIDFormat>
    </AttributeAuthorityDescriptor>

</EntityDescriptor>
```

Figure 41: IdP Metadata file.

8.2.2.1. Authentication mechanism

The first step done to configure the IdP was to define the authentication mechanism that was going to be used. Currently there are four defined mechanisms in Shibboleth 2.0, but it is known that some more will be added in the near future. These four mechanisms are Remote User, Username/Password, IP Address and Previous Session.

Remote User sets the principal name in the IdP to REMOTE_USER as determined by the web server or container's authentication. **Username/Password** presents the user with an authentication page and then checks the entered username and password against an LDAP directory or Kerberos 5 domain. **IP Address** checks the user's IP address against a given range to identify the user as a pre-determined principal. **Previous Session** is called when the user's existing session is used as the authentication mechanism (i.e. when the IdP is performing an SSO based sign-on). The Previous Session mechanism doesn't need to be configured nor modified.

In our case the authentication required was Username/Password through the e-Catalunya LDAP for the user authentication and Previous Session for the SSO. To

enable the Username/Password mechanism its definition in the *handler.xml* file had to be uncommented:

```
<LoginHandler xsi:type="UsernamePassword"
   jaasConfigurationLocation="file:///opt/shibboleth-idp-
2.0.0/conf/login.config">
        <AuthenticationMethod>urn:oasis:names:tc:SAML:2.0:ac:classes:
PasswordProtectedTransport</AuthenticationMethod>
</LoginHandler>
```

Figure 42: Authentication methods definition in handler.xml file

Afterwards the mechanism had to be configured in the *login.config* file as indicated in the definition above:

```
ShibUserPassAuth {

  edu.vt.middleware.ldap.jaas.LdapLoginModule required
      host="daniol.fib.upc.edu"
      port="9389"
      base="ou=users,dc=exoplatform,dc=com"
      serviceUser="cn=admin,dc=exoplatform,dc=com"
      serviceCredential="exo"
      userField="uid";

};
```

Figure 43: Username/Password authentication configuration in login.config file

8.2.2.2. Defining a new SP

The only thing remaining for the initial configuration was to define new the SP with which this IdP could be communicating. To accomplish this, a new relaying party was defined in the *relying-party.xml* file:

```
<RelyingParty id="https://tokyo.fib.upc.edu/shibboleth"
              provider="https://tokyo.fib.upc.edu/idp/shibboleth"
              defaultSigningCredentialRef="IdPCredential">

 <ProfileConfiguration xsi:type="saml:ShibbolethSSOProfile" />
 <ProfileConfiguration xsi:type="saml:SAML1AttributeQueryProfile" />
```

```
 <ProfileConfiguration xsi:type="saml:SAML2SSOProfile" />
 <ProfileConfiguration xsi:type="saml:SAML2AttributeQueryProfile" />
 <ProfileConfiguration xsi:type="saml:SAML2ArtifactResolutionProfile"
/>

</RelyingParty>
```

Figure 44: SP definition in the relaying-party.xml file

The id field is the entityID of our SP, which by default is https://*HostName*/shibboleth, and the provider field is the URI of the IdP when answering requests to this relying party or group. The list of <ProfileConfiguration> elements defines the profiles supported by the relying party.

Finally, the SP metadata had to be loaded by the IdP. To create the SP metadata using the default metadata file provided by the Shibboleth team the SP had to be installed. The installation of the SP and its configuration will be explained below in section 8.1.3.

A definition of a metadata provider is needed for each metadata file that the IdP must know load. Metadata Providers are defined with a MetadataProvider element in the *relying-party.xml* file. Metadata Providers must have an id attribute which assigns a unique id to the provider. There are different types of metadata providers supported, such as File Backed HTTP Metadata Provider, Resource backed Metadata Provider etc.

Here the Chaining Metadata Provider and the Filesystem Metadata Provider were used. The Chaining Metadata Provider provides a means of logically combining other Metadata Provider definitions. When the IdP requests metadata from this type of provider the provider returns the response of the first metadata provider, in the chain, that provides a response. For example, if providers A, B, and C are defined in a chain and both B and C contain an entity descriptor for *urn:example.org:idp1*, which the IdP is searching for, only the entity descriptor from B will be returned as it will be the first provider to return a non-empty response.

The Filesystem Metadata Provider reads SAML 2 metadata from a file on the file system. Metadata is cached in memory for a period of time in order to improve performance. The metadata provider also monitors the file for changes and will reload the file upon detecting an update. The metadataFile attribute shows the file system path to the metadata file.

```
<MetadataProvider id="ShibbolethMetadata"
xsi:type="ChainingMetadataProvider"
xmlns="urn:mace:shibboleth:2.0:metadata">

 <!-- MetadataProvider reading metadata from the filesystem -->
 <!--SP Metadata-->

 <MetadataProvider id="FSMD" xsi:type="FilesystemMetadataProvider"
  xmlns="urn:mace:shibboleth:2.0:metadata"
  metadataFile="/home/alba/Desktop/Metadata.xml"
  maintainExpiredMetadata="true">
 </MetadataProvider>


 <!-- MetadataProvider reading metadata from the filesystem -->
 <!--IDP Metadata-->

 <MetadataProvider id="IDPMD" xsi:type="FilesystemMetadataProvider"
  xmlns="urn:mace:shibboleth:2.0:metadata"
  metadataFile="/opt/shibboleth-idp-2.0.0/metadata/idp-metadata.xml"
  maintainExpiredMetadata="true">
 </MetadataProvider>

</MetadataProvider>
```

Figure 45: Metadata providers definition in the relying-party.xml file.

8.2.2.3. Trust Engine

The IdP has two types of trust engines. **Signature trust engines** which are used to validate a digital signature and ensure that the credentials used to create this signature are trusted and **credential trust engines** which are used to ensure that the credentials used by a service provider to connect to the IdP are valid and trusted. All trust engines are defined in the IdP's *relying-party.xml* configuration file after any credentials and before any defined security policies.

Since credential trust engines are still being developed in Shibboleth 2.0 they are not going to be explained any deeper.

Signature trust engines are subdivided into two groups. **Static engines** which evaluate a signature against static information explicitly provided at configuration time and **metadata engines** which use SAML metadata, which may change over time, to lookup or derive the needed security information.

In this project the metadata engine was used so any credentials loaded by a metadata provider could be trusted.

There are two types of metadata engines, the explicit key metadata signature and the PKIX metadata signature trusted engine. The **explicit key**

metadata based signature trust engine evaluates trustworthiness by looking to an entity's role metadata. If there is a defined key that validates the signature then the signature is trusted. The **PKIX metadata trust engine** is similar to the explicit key metadata trust engine except that it also performs PKIX validation using information located within the metadata.

In both metadata engines there is a reference, metadataProviderRef, which points to the metadata provider that can be trusted.

By default Shibboleth 2.0 offers an already defined signature trust engine in which the explicit key metadata signature and the PKIX metadata trust engine are defined. In this project the default configuration was used to trust the SP credentials and so the metadataProviderRef was modified so it could point to the metadata provider ShibbolethMetadata previously defined.

```
<security:TrustEngine id="shibboleth.SignatureTrustEngine"
xsi:type="security:SignatureChaining">
        <security:TrustEngine
id="shibboleth.SignatureMetadataExplicitKeyTrustEngine"
xsi:type="security:MetadataExplicitKeySignature"
                                metadataProviderRef="ShibbolethMetadata"
/>
        <security:TrustEngine
id="shibboleth.SignatureMetadataPKIXTrustEngine"
xsi:type="security:MetadataPKIXSignature"
                                metadataProviderRef="ShibbolethMetadata"
/>
    </security:TrustEngine>
```

Figure 46: Metadata trust engine configured to trust the SP credentials in relying-party.xml

8.2.3. SP installation and first configuration

To deploy the SP some software was required:

- **Xerces-C:** Validating XML parser designed to give an application the ability to read and write XML data. Provides a shared library for parsing, generating, manipulating, and validating XML documents. Shibboleth uses this to work with XML files.
- **Log4shib:** Forked version of log4cpp logging library that has been created for the Shibboleth project.

- **XML-Security-C:** Library developed by the Apache project with the aim of work with XML in a secure way. Shibboleth use this to manage secure XML files.
- **XML Tooling-C:** lower-level library that provides a higher level interface to XML processing, particularly in light of signing and encryption.
- **Libcurl:** Client-side URL transfer library.
- **Opensaml:** *I*s a set of open-source libraries in Java and C++ which can be used to build, transport, and parse SAML messages. OpenSAML is able to store the individual information fields that make up a SAML message, build the correct XML representation, and parse XML back into the individual fields before handing it off to a recipient. OpenSAML supports the SOAP binding for the exchange of SAML request and response objects (C++ supports requesting only). Shibboleth use this to support the SAML assertions.

After installing all the components above only remained to compile and install the Shibboleth SP.

The SP is basically composed by two components: a module that must be loaded in the web server, in this case apache, and a daemon. The SP includes a cache of session information and can provide cookie-based session semantics to applications independently of the application's own language or framework. Most web servers today run as a pool of processes, even if this pool is small, and the simplest way to provide such state for a single server is to store it in a separate process. Therefore the Shibboleth daemon is used.

Moreover access to private keys can be restricted to the daemon process and not granted to the web server, while still allowing applications on the server to receive encrypted data; the system stability is improved by limiting the code that needs to run inside web server processes and many significant libraries need not be linked into the web server, minimizing conflicts and speeding up process creation.

Figure 47 shows the SP with both of its components together with some of the e-Catalunya architecture.

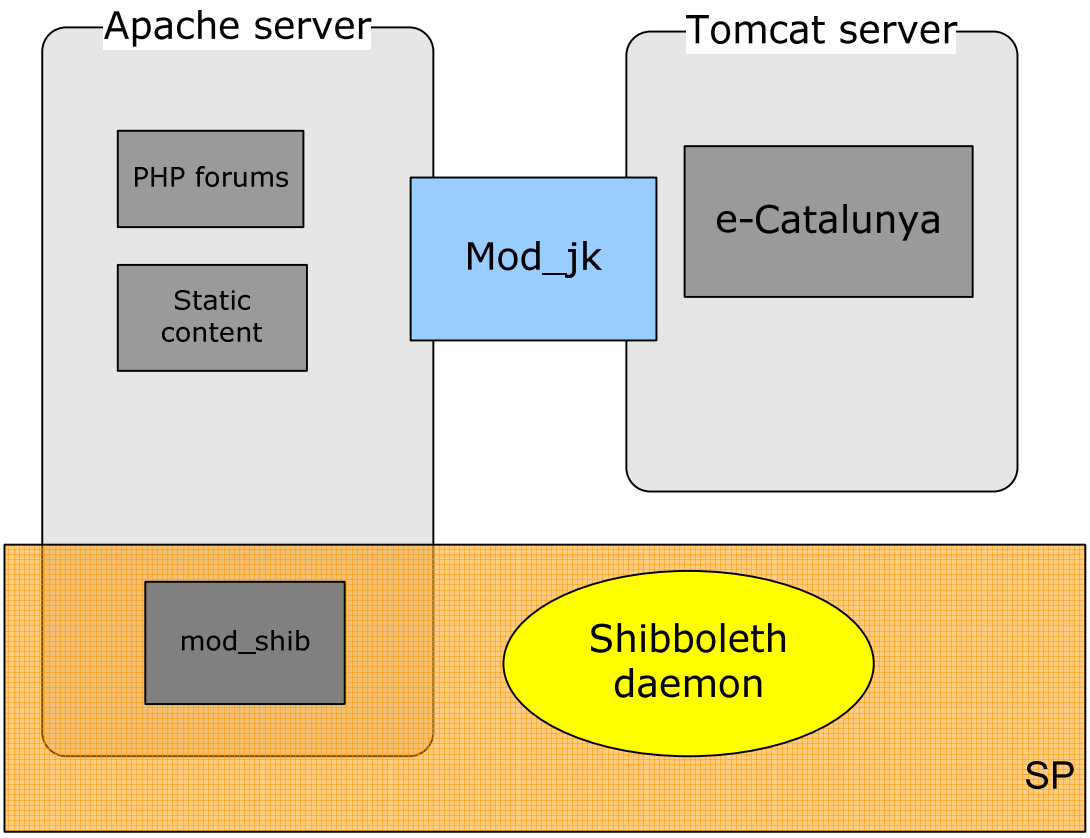

Figure 47: SP components in the e-Catalunya platform.

After building the SP a default metadata file was created, the same way as in the IdP installation. This SP metadata file is known by the IdP as explained above thanks to the configuration of the *relying-party.xml* IdP file.

By default the entityID generated for the SP has the following schema: https://*host-name*/shibboleth. Hence in our case the eintityID was https://tokyo.fib.upc.edu/shibboleth.

For each SP definition must be one <EntityDescriptor> element in the metadata file: the <SPPSSODescriptor>.

Figure 48 shows the SP metadata file with the modified entityID.

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
entityID="https://tokyo.fib.upc.edu/shibboleth">

  <md:SPSSODescriptor
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol
urn:oasis:names:tc:SAML:1.1:protocol urn:oasis:names:tc:SAML:1.0:protocol">
    <md:KeyDescriptor use="signing">
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName>linux-b47g</ds:KeyName>
        <ds:X509Data>
```

```
          <ds:X509SubjectName>CN=linux-b47g</ds:X509SubjectName>
          <ds:X509IssuerSerial>
            <ds:X509IssuerName>CN=linux-b47g</ds:X509IssuerName>
            <ds:X509SerialNumber>13368210379452187093</ds:X509SerialNumber>
          </ds:X509IssuerSerial>

<ds:X509Certificate>MIIC5TCCAc2gAwIBAgIJALmFbIMl8AHVMA0GCSqGSIb3DQEBBQUAMBUxEzARBgNV
BAMTCmxpbnV4LWI0N2cwHhcNMDgwNTI3MTk0MjM5WhcNMTgwNTI1MTk0MjM5WjAV
MRMwEQYDVQQDEwpsaW51eC1iNDdnMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAqTi7YJgEIVrBgA3APyATwakxhWni7jQ2n9cXHuXn+S5lpLQZDWkIUNkg
o2A4Sx0UqZBUth7SUQk0b1oX2CbXW+yepzf9JeKlyptFyRMMYLT0ZhIwblACwpco
q3zfhgbCJ2HsMPZinezEBhxKu9o8MZMsEOTNIJrw+EGPXwhLeMG1jI4VaP85Tq5P
nujtAEL5VWdTluY3g7O/PRLjTEcqHvbzVwML4yekBtkhuGUPv68cVKm8Uq9bcFDh
sFMesnfO2PX+l1/rNYcx+NcxJ2gn5FCoORKvpyfEh/XrBRUH3BW9mm+CrNn5e/zg
0KlodQYkWrVxtp3gopX3MT7P1p8ZIQIDAQABozgwNjAVBgNVHREEDjAMggpsaW51
eC1iNDdnMB0GA1UdDgQWBBS5Vq9WttaP5lwYxdkX7umx3ITmoTANBgkqhkiG9w0B
AQUFAAOCAQEAYnApnMNEd+MQlpV9Vyc6AnH5P1UyxfcRXvNu1i0Xnle7V70FXRod
yyD4Vp8fDCsZ5b5ZReyXFWGd0Uui16R6S1Y4Hm1uF8ZxwUv+K9ikEthI3Wf1k+qu
AY86TPUsjHWTGKxh+rAwZjYXumLXPK4Y7htvrX7dQAyy+Cqy3tLvC0exKo0+Te1o
oa7xbGfdAsqFg6wLgxjLZpw155swappk6gddZZFJCtjizizlSwedHcBZGOc4ch09
WmkcLwVpkfXbw3LVo+FP/I6XICgt1r0h30npoiVr3cEYgKl6zaD3bOHMj9aLzoht
z2WYxAINfcyfdBFcfglOhi0l/Slm0EO3ZA==
</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </md:KeyDescriptor>
    <md:KeyDescriptor use="encryption">
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName>linux-b47g</ds:KeyName>
        <ds:X509Data>
          <ds:X509SubjectName>CN=linux-b47g</ds:X509SubjectName>
          <ds:X509IssuerSerial>
            <ds:X509IssuerName>CN=linux-b47g</ds:X509IssuerName>
            <ds:X509SerialNumber>13368210379452187093</ds:X509SerialNumber>
          </ds:X509IssuerSerial>

<ds:X509Certificate>MIIC5TCCAc2gAwIBAgIJALmFbIMl8AHVMA0GCSqGSIb3DQEBBQUAMBUxEzARBgNV
BAMTCmxpbnV4LWI0N2cwHhcNMDgwNTI3MTk0MjM5WhcNMTgwNTI1MTk0MjM5WjAV
MRMwEQYDVQQDEwpsaW51eC1iNDdnMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAqTi7YJgEIVrBgA3APyATwakxhWni7jQ2n9cXHuXn+S5lpLQZDWkIUNkg
o2A4Sx0UqZBUth7SUQk0b1oX2CbXW+yepzf9JeKlyptFyRMMYLT0ZhIwblACwpco
q3zfhgbCJ2HsMPZinezEBhxKu9o8MZMsEOTNIJrw+EGPXwhLeMG1jI4VaP85Tq5P
nujtAEL5VWdTluY3g7O/PRLjTEcqHvbzVwML4yekBtkhuGUPv68cVKm8Uq9bcFDh
sFMesnfO2PX+l1/rNYcx+NcxJ2gn5FCoORKvpyfEh/XrBRUH3BW9mm+CrNn5e/zg
0KlodQYkWrVxtp3gopX3MT7P1p8ZIQIDAQABozgwNjAVBgNVHREEDjAMggpsaW51
eC1iNDdnMB0GA1UdDgQWBBS5Vq9WttaP5lwYxdkX7umx3ITmoTANBgkqhkiG9w0B


AQUFAAOCAQEAYnApnMNEd+MQlpV9Vyc6AnH5P1UyxfcRXvNu1i0Xnle7V70FXRod
yyD4Vp8fDCsZ5b5ZReyXFWGd0Uui16R6S1Y4Hm1uF8ZxwUv+K9ikEthI3Wf1k+qu
AY86TPUsjHWTGKxh+rAwZjYXumLXPK4Y7htvrX7dQAyy+Cqy3tLvC0exKo0+Te1o
oa7xbGfdAsqFg6wLgxjLZpw155swappk6gddZZFJCtjizizlSwedHcBZGOc4ch09
WmkcLwVpkfXbw3LVo+FP/I6XICgt1r0h30npoiVr3cEYgKl6zaD3bOHMj9aLzoht
z2WYxAINfcyfdBFcfglOhi0l/Slm0EO3ZA==
</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </md:KeyDescriptor>

    <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SLO/SOAP"/>
```

```
    <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect" Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SLO/Redirect"/>
    <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SLO/POST"/>
    <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Artifact" Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SLO/Artifact"/>
    <md:ManageNameIDService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
Location="https://tokyo.fib.upc.edu/Shibboleth.sso/NIM/SOAP"/>
    <md:ManageNameIDService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect" Location="https://tokyo.fib.upc.edu/Shibboleth.sso/NIM/Redirect"/>
    <md:ManageNameIDService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location="https://tokyo.fib.upc.edu/Shibboleth.sso/NIM/POST"/>
    <md:ManageNameIDService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Artifact" Location="https://tokyo.fib.upc.edu/Shibboleth.sso/NIM/Artifact"/>
    <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST" Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SAML2/POST" index="1"/>
    <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST-SimpleSign" Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SAML2/POST-
SimpleSign" index="2"/>
    <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Artifact" Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SAML2/Artifact"
index="3"/>
    <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:PAOS"
Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SAML2/ECP" index="4"/>
    <md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:1.0:profiles:browser-post"
Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SAML/POST" index="5"/>
    <md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:1.0:profiles:artifact-01"
Location="https://tokyo.fib.upc.edu/Shibboleth.sso/SAML/Artifact" index="6"/>
  </md:SPSSODescriptor>

</md:EntityDescriptor>
```

Figure 48: SP Metadata file.

The most important SP configuration files are *shibboleth2.xml, attribute-map.xml, attribute-policy.xml* and the metadata file. The *shibboleth2.xml* Most of the native service provider's configuration options are found in shibboleth2.xml, located in the SP's main configuration directory. This is the first file that needs to be modified to enable communication with other parties, in our case the IdP already installed.

First of all the SP has to be able to find the IdP metadata. This is configured exactly the same way as the IdP and its configuration is in the *shibboleth2.xml* file. So in our case the following lines were added:

```
<MetadataProvider type="Chaining">

<MetadataProvider type="XML" file="/opt/shibboleth-idp-
 2.0.0/metadata/idp-metadata.xml" />
 </MetadataProvider>
```

Figure 49: Metadata provider definition in shibboleth2.xml file.

Next, the runtime behaviour of the software when processing SAML assertions and protocols had to be defined. That is done with the <ApplicationDefaults> element. The usual pattern is to define most of this information once, in the outer element, and then supply a limited set of overridden information for each application you define that needs exceptional behaviour with a <ApplicationOverride> element.

The homeURL is where the SP redirects the client to when there is nothing else that can be done with a request and should be set to a standard home page or index page. In this case it redirects to the main page of e-Catalunya platform.

```
<ApplicationDefaults id="default" policyId="default"
      entityID="https://tokyo.fib.upc.edu/shibboleth"
      homeURL="http://tokyo.fib.upc.edu/portal/index.jsp"

    >
```

Figure 50: Application definition in the shibboleth2.xml file.

8.2.3.1. Session Initiator

The <SessionInitiator> element is used to configure handlers that are responsible for initiating the process of authentication to the SP and establishing a session with it. This represents what single sign-on architecture refers to as an "SP-initiated" flow, where a browser starting at the application end needs to be referred to the IdP to login and then return with the appropriate information to login.

Broadly speaking, there are two kinds of SessionInitiators: **protocol handlers** and **discovery handlers**.

A protocol handler requires that the entityID of an IdP be supplied to the handler so that its metadata can be obtained to determine whether and where it supports a particular protocol. The entityID can be supplied in a number of ways,

including via query string, a hard coded <SessionInitiator> attribute, or via a content setting applied to the resource. The absence of the entityID or the metadata file where this entityID is defined causes a warning to be logged and the handler otherwise ignores the request.

By contrast, a discovery handler requires that the IdP not be known, and is responsible for interacting with the browser in some way to determine the IdP to use. It can do this in any way it wants to, such as examining cookies, interacting with the user, or by redirecting the browser to some other server. In this case no entityID can be known or the handler will silently ignore the request, since discovery would serve no purpose.

Shibboleth 2.0 offers two Protocol Handler SessionInitiators (SAML2 SessionInitiator and SHIB1 SessionInitiator) and four Discovery Handler Sessioninitiators (SAMLDS SessionInitiator, WAYF SessionInitiator, Cookie SessionInitiator and Form SessionInitiator).

SAML2 SessionInitiator is indicated by type="SAML2" and it supports SAML 2.0 authentication requests. As a protocol handler, an entityID must be specified/known, which is then used to check for metadata with an <md:IDPSSODescriptor> role supporting SAML 2.0.

SHIB1 SessionInitiator is indicated by type="SHIB1" and supports Shibboleth 1.x authentication requests, an extension of the SAML 1.1 standard. As a protocol handler, an entityID must be specified/known, which is then used to check for metadata with an <md:IDPSSODescriptor> role supporting Shibboleth 1.x.

SAMLDS SessionInitiator is indicated by type="SAMLDS" and refers the browser to a discovery service (DS) supporting the proposed SAML 2.0 IdP Discovery Protocol. The design of this protocol results in a redirect back to the SP with a single entityID selected, if one is chosen by the user. The handler implementation causes this redirect to be returned to itself in a manner that allows the handler to be configured in a "chain" with one or more protocol handlers. Used alone, the SAMLDS handler will simply display an error when the result is returned because it cannot itself cause an authentication request to be generated.

WAYF SessionInitiator is indicated by type="WAYF" and it refers the browser to a Shibboleth WAYF service. The design of this protocol results in the browser leaving the SP until a successful authentication response is returned. This is a legacy technique that limits the ability of an SP to effectively support newer protocols and more advanced features.

The **Cookie SessionInitiator** is indicated by type="Cookie", checks for a cookie maintained as part of the SP's IdP history feature and uses it to obtain the

entityID to use for later SessionInitiator handlers in a chain. This handler doesn't actually cause a response to the browser, but it generally runs first in a chain, and allows the entityID to be set before other handlers run.

Finally the **Form SessionInitiator** is indicated by type="Form", displays an HTML template containing a form to prompt the user for the entityID to use. This is a simple substitute for referring the user to another site, which is generally incapable of addressing scenarios involving multiple sets of unrelated IdPs. This handler can be combined with the Transform SessionInitiator to enable the user's input to be turned from something simpler into an entityID.

Shibboleth 2.0 also defines a **Chaining SessionInitiator** (as the Chaining MetadataProvider defined previously). It is identified by type="Chaining" and it wraps a sequence of SessionInitiator handlers so that they run in series. The series ends when a handler indicates that a response to the browser was returned.

For the WAYF (Where Are You From) and the DS (Discovery Service) a DS Service has to be installed. The Discovery Service automatically selects between "WAYF-Mode" and the full SAML 2.0 Discovery Service Protocol depending if the specification being used is SAML 1.x or SAML 2.0.

The WAYF service was developed to make up for the lack of a discovery service in SAML 1.x. It is an intermediary service between the SP and the IdP and it works very similar to the Discovery Service defined by SAML 2.0. The first time the client accesses an SP, the WAYF might present the user with a form containing a list of all available IdPs. The user selects an IdP from the list and submits the form, after which the WAYF redirects the client to the selected IdP. At the same time, the client stores a reference to the IdP in a client-side cookie such that subsequent visits to the same SP are redirected straight through to the corresponding IdP by the WAYF.

In this project the SAML2 SessionInitiator was used due to the existence of only one IdP. In case of having more than one IdP the SAMLDS SessionInitiator would be used.

```
          <SessionInitiator type="Chaining" Location="/Login"
isDefault="true" id="Intranet" acsByIndex="false"
                relayState="cookie"
entityID="https://tokyo.fib.upc.edu/idp/shibboleth">
        <SessionInitiator type="SAML2" defaultACSIndex="1"
template="bindingTemplate.html"/>
 </SessionInitiator>
```

Figure 51: Session Initiator definition in the shibboleth2.xml file.

8.2.3.2. Access control

At this moment the basic configurations to enable communication between the SP and the IdP were done. Now, the next step was to protect the e-Catlunya platform with the SP. At this point the JOSSO authentication and authorization control were deleted.

The protection of content with the SP can be done in different ways. Each application is a different world and depending on how the authentication and authorization are performed the "shibollizing" of this application can be more or less difficult.

The three ways to protect content with Shibboleth are:

- **XML access control:** Authorization decisions are controlled by the SP and the access rules are defined in the shibboleth2.xml file. Shibboleth can use <RequestMapper type="XML"> with a set of <Host> and <Path> elements to represent part of your webspace. When a request comes in that matches the hosts and paths, a set of <Rule>s can be applied, optionally using boolean <AND>, <OR>, or <NOT> logic elements.
- **Web server-based access control:** Authorization decisions are done by the web server. In Apache case the access rules can be defined in the httpd.conf file or in some .htaccess file.
- **Application access control:** Authorization decisions are the application responsibility. All the attributes that Shibboleth acquires in a trusted way can be supplied to the application to allow the application to make the access control decisions. This is the preferred approach, because the application can display the most helpful errors and refusals.

In e-Catalunya all authorization decisions are done by the application itself. These decisions are done based on the user id (the user email) that e-Catalunya gets from the REMOTE_USER variable that has been populated by JOSSO after a user has logged in.

Therefore the SP Shibboleth had to perform as JOSSO did to avoid any need of modifications in the platform.

However it was decided to configure first the web server-based control since it seemed easier to do, to see if the communication between the already installed IdP and SP worked correctly. After confirming that, the web server based control could be replaced for the application access control.

That was done because, as said above, both the IdP and the SP were being deployed simultaneously. Hence it was wiser to ensure first that the communication between these two was performed correctly before trying to populate correctly the REMOTE_USER variable in Apache.

To protect content through Apache the following lines were added in the apache configuration file, typically *httpd.conf*. Here they were added in the *tomcat.conf* file, included in the *httpd.conf* file, where all the configurations of the e-Catalunya platform are.

```
<Location /portal>
  AuthType shibboleth
  ShibRequireSession On
   require valid-user
</Location>
```

Figure 52: Protecting the platform through Apache.

With those lines added every access to the platform could need a Shibboleth authentication, no matter if the access was to a public or a private zone.

To configure the component used by the SP to map incoming requests to the set of configuration options that should be applied the <RequestMapper> element in the shibboleth2.xml file had to be configured. Every request to the web server is associated with an application by the request mapper component. An application represents distinct collections of resources within the server's document tree that share the same general configuration requirements, although access control can be enforced on a finer-grained basis.

These application boundaries may or may not align with the notion of an application as understood in other contexts, but usually do. A very common strategy is for each virtual host on a server to map to its own application.

When a resource protected by the SP needs distinct behavior in these kinds of areas, a new application needs to be defined here and referenced by the request mapper using the applicationId property.

Shibboleth 2.0 defines two Request Mappers: the Native Mapper and XML Request Mapper.

The **Native Request Mapper**, identified by type="Native", integrates native web server content configuration features with the portable syntax supported by

the XML request mapper. For most deployments, this is the type to use. It is a hybrid that allows you to combine Apache commands in .htaccess files with XML-based configuration and seamlessly switch back and forth. The native commands override any XML-based attributes. The

The **XML Request Mapper**, identified by type="XML", provides a portable XML syntax for configuring settings based on mappings assigned to URL host, path, and query string information. It does not permit settings to be defined in the web server's configuration. The only reason for using this type in favour of the "Native" type above would be to prevent developers with access to content from using .htaccess commands. Since Apache itself can prevent this already, it's largely superfluous, but there might be very small efficiency gains from using it in such cases.

In this project there was only one application to protect and therefore only one default Request Mapper was defined. As the first configuration intended was the web server access control the Native Request Mapper was used with the requireSession set to true. This forces any connection to the protected area (in this case all the application) to have an authenticated session. If none exists, the SP will try to automatically establish one using the default SessionInitiator. In **Figure 53** is the portion of the shibboleth2.xml file that configures the request mapper:

```
<RequestMapper type="Native">
        <RequestMap applicationId="default">
            <Host name="tokyo.fib.upc.edu">
                    <Path name="/portal" authType="shibboleth"
requireSession="true"/>
                </Host>

      </RequestMap>
  </RequestMapper>
```

Figure 53: Request mapper configuration in the shibboleth2.xml file

At this point the IdP and the SP could already communicate between each other. By default the RedirectSSO SAML 2.0 profile was used to implement Single Sign On. Now, when trying to connect to any place of the e-Catalunya platform the SP redirected the user to the IdP. There the user authenticated himself against the LDAP configured in the IdP. The IdP recollected the username and the password using an HTML form. Once the user was authenticated he was automatically redirected to the SP and afterwards to the e-Catalunya platform where now the user had access.

Since the e-Catalunya manages its own authorization accesses using the email of the users, with this basic configuration users could only have access to the public areas. Moreover the phpBB2 authorization access was still unresolved. So, having both installations and their main configurations tested, the next step was to disable the web server access control and give all authorization responsibility to the application.

8.3. Second configuration

There are a hundred different ways to integrate a Shibboleth SP with an application. Some are very elegant, and some are really not elegant. Some are simple and fast, and others are complicated and will take a long time. The key is to deliver the information Shibboleth supplies to the application however the application expects it, or to modify the application.

Many applications have control over authentication of users. Single sign-on is the art of taking control from the application, and teaching it to rely on authentication (and often attributes) from the surrounding environment. However, sometimes you don't have this much control over the application, in which case single sign-on becomes the art of taking attribute and authentication information and placing it where and how the application wants it.

To shibbolize e-Catalunya two questions had to be considered:

- How to pass the information needed, in this case the user email.
- When to create the Shibboleth session.

The former question is developed in section 8.2.1 while the latter is developed in section 8.2.2.

8.3.1. Transmitting user identification

The Shibboleth SP places authentication and attribute information in the web environment as HTTP headers or environment variables. Any application that has been enabled to rely on standard HTTP environment information such as REMOTE_USER or uses web server commands for content protection has already been prepared for use with Shibboleth.

As explained in chapter 7, in the e-Catalunya JOSSO is the responsible of authenticating the users while the platform itself manages the authorization access. JOSSO populates the REMOTE_USER variable with the email of the authenticated

user allowing the platform to get the user identification. With this information e-Catalunya knows which places a user have access to and which actions he is allowed to do.

Hence the objective was to do the same with Shibboleth. Shibboleth had to populate the REMOTE_USER variable as JOSSO did, so no changes could be needed in the platform.

In the AJP Coyote connector configuration part of *server.xml* of the e-Catalunya tomcat, tomcatAuthentication="false" was added to allow the REMOTE_USER header to be communicated from Apache mod_jk to the AJP connector.

```
      <!-- Define a Coyote/JK2 AJP 1.3 Connector on port 8009 -->
          <Connector port="8009"
                     enableLookups="false"         redirectPort="8443"
debug="0"
                     protocol="AJP/1.3"
                     tomcatAuthentication="false"/>
```

Figure 54: Connector definition in the server.xml file of Tomcat.

In e-Catalunya the Apache server and the Tomcat Server are connected using the mod_jk modul. In case that they were connected with mod_proxy the line "ShibUseHeaders On" would be needed in the apache configuration file, in this case *tomcat.conf,* because environment variables are not passed by mod_proxy_ajp unless they have AJP_ prefixes.

```
<Location /portal>
  AuthType shibboleth
  ShibRequireSession On
  ShibUseHeaders On
   require valid-user
</Location>
```

Figure 55: Apache configuration to enable passing variables to Tomcat when using mod_proxy modul

Next an attribute had to be defined, both in the IdP and the SP, to sent the email of the authenticated user from the IdP to the SP and afterwards populate the REMOTE_USER HTTP header.

There were two ways of communicating the email to the SP. One was defining an attribute and the other was using the SAML 2.0 NameID. In SAML 1 the IdP was the sole arbiter of which name identifier was used for a user at any given point. SAML 2 however introduced not only SAML metadata, which allows entities to indicate which formats they support, but also added the ability for an SP to require a particular format using he NameID element.

For this reason NameID was chosen over an attribute in this deployment. Nevertheless in the following section a brief overview of attribute definition and release is going to be exposed.

8.3.1.1. Attribute definition and release in the IdP

An attribute passes through four main steps from source system to release to an SP: it's pulled from the system of record, massaged within the provider, given a set of protocol-specific encoders, and then filtered for release. Every attribute has a unique attribute ID which is used to refer to it consistently through this process.

Shibboleth doesn't store any attributes about users itself; it relies on external data stores to supply user information to be released. These attributes are generally pulled from data sources using data connectors. Data connectors may produce more than one attribute and each attribute may have many values.

Shibboleth 2.0 defines four Data Connectors to extract attributes from data storages. The **static data connector** is used to add statically attributes and values to every person served by the identity provider. An example usage of this connector would be to add an entitlement attribute that everyone in your organization receives. The **stored ID data connector** is used to construct and persist identifiers by means of a database. The **relational database connector** is used to pull attributes from a relational database by executing some configured SQL. The **LDAP data connector** is used to pull attributes from an LDAP directory by executing an LDAP filter on a specific branch.

In this project no attribute was going to be extracted from any data storage since the user email is the username used to identify the user. So when a user authenticates himself against the IdP he is already providing his email and thus, if the authentication is correct, the attribute is already known.

Once the attribute is obtained there are two rounds of transformation each attribute may undergo as it passes through the IdP. The first set of changes allow the IdP to transforms attributes (merge, split, reformat, etc.) using other attributes to get a complete data definition. The second set of changes transform the attribute from the internal representation to one appropriate for the protocol the IdP will be communicating with, a process known as attribute encoding. The former is

configured with the definition of the attribute and the latter with the definition of the attribute encoding.

The attribute definition gives an ID and value to the attribute. Shibboleth defines eleven attribute definitions.

The **simple attribute definition** is used to handle flat attributes with a simple name and values. The **scoped attribute definition** is used to handle flat attributes with a simple name and scoped values. The **prescoped attribute definition** creates scoped attribute values from a delimited string. The **principal name attribute definition** creates an attribute whose value is the user's principal name. The **principal authentication method attribute definition** creates an attribute whose value is the user's authentication method. **The regex based split attribute definition** creates an attribute whose values are the split of a set of input values. The **SAML 1 NameIdentifier attribute definition** creates an attribute whose values are SAML 1 NameIdentifiers. The **SAML 2 NameID attribute definiti**on creates an attribute whose values are SAML 2 NameIDs. The **script attribute definition** builds an attribute's values based on an ECMAScript script. The **mapped attribute definition** is used to map an attribute's values to different values using regular expressions. The **template attribute definition** uses the VelocityTemplateLanguage to construct values by combining other attribute values.

Once the attribute has the internal name and value desired, one more transformation has to take place to give it the right format on the wire. Each attribute needs one encoder per protocol it will be sent on. Shibboleth 2.0 defines the following eight encoders:

- **SAML 1 String Attribute Encoder** is used to encode an attribute as a SAML 1 <Attribute> with simple strings for values.
- **SAML 1 Scoped String Attribute Encoder** is used to encode an attribute as a SAML 1 <Attribute> with scoped strings for values.
- **SAML 1 Base64 Attribute Encoder** is used to encode an attribute as a SAML 1 <Attribute> with a byte array for a value.
- **SAML 1 XMLObject Attribute Encoder** is used to encode an attribute, with XMLObjects as values, into a SAML 1 <Attribute>.
- **SAML 2 String Attribute Encoder** is used to encode an attribute as a SAML 2 <Attribute> with simple strings for values.
- **SAML 2 Scoped String Attribute Encoder** is used to encode an attribute as a SAML 2 <Attribute> with scoped strings for values.
- **SAML 2 Base64 Attribute Encoder** is used to encode an attribute as a SAML 2 <Attribute> with a byte array for a value.

- **SAML 2 XMLObject Attribute Encoder** is used to encode an attribute, with XMLObjects as values, into a SAML 2 <Attribute>.

Newly defined attributes are not released to service providers until an attribute filter policy for that attribute hasn't been defined. Such policies describe which service providers, under which conditions, receive which attributes. These release rules are defined in the *attribute-filter.xml* file. An attribute rule is defined with the element <AttributeRule> with the attribute attributeID and a <PermitValueRule xsi:type="MATCHING_RULE_TYPE"> element. The attributeID is the ID of the attribute to which the rule applies, as assigned in the attribute resolver. The <PermitValueRule xsi:type="MATCHING_RULE_TYPE"> element describes which values are released. A value is released if the permit value rule evaluates to true.

Some of the MATCHING_RULE_TYPE defined by Shibboleth 2.0 are:

- **ANY**: Always evaluates to true
- **AND**: Evaluates to true if all contained rules are true
- **OR**: Evaluated to true if any contained rule is true
- **NOT**: Evaluates to true if the contained rule evaluates to false
- **AttributeRequesterString**: Evaluates to true if the attribute requester's entity ID matches a given string
- **AttributeIssuerString**: Evaluates to true if the attribute issuer's entity ID matches a given string
- **AuthenticationMethodString**: Evaluates to true if the method used to authenticate the user matches a given string
- **AttributeRequesterRegex**: Evaluates to true if the attribute requester's entity ID matches a given regular expression

8.3.1.2. NameID

Supporting a new name identifier within the identity provider is a three step process; configuring the IdP to produce name identifiers of the given type, configuring the IdP to accept these name identifiers, and expressing support for the new identifiers within metadata.

Since Shibboleth 2.0 regards NameIDs to be one specific type of attribute the values of name identifiers are simply attributes constructed through the normal attribute resolver and filter infrastructure. This allows the identity provider to support many different name formats but control, through attribute filter policies, which service providers get which name identifiers.

The IdP defines two name identifier attribute encoders: the SAML 1 String Name Identifier and the SAML 2 String NameID. The **SAML 1 String NameIdentifier** is used encode an attribute into a SAML 1 NameIdentifier with a simple string value. The **SAML 2 String NameID** is used encode an attribute into a SAML 2 NameID with a simple string value.

Obviously the encoder chosen was the SAML 2 String NameID for this project. Figure 56 shows the definition of the NameID in the e-Catalunya deployment in the *attribute-resolver.xml* file. In this case it was also defined that the format of the identifier was an email address.

```
 <resolver:AttributeDefinition id="principal" xsi:type="PrincipalName"
xmlns="urn:mace:shibboleth:2.0:resolver:ad">

          <resolver:AttributeEncoder xsi:type="SAML2StringNameID"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"

nameFormat="urn:oasis:names:tc:SAML:2.0:nameid-format:emailAddress" />
    </resolver:AttributeDefinition>
```

Figure 56: NameID definition in the attribute-resolver.xml file.

If desired, Shibboleth allows defining Transient NameID using the Transient ID attribute definition that generates an opaque ID and maintains a short-term association between this ID and the principal for which it was generated.

The second part of the configuration allows the IdP to acquire the a user's principal name from a name identifier used by a relying party. In other words, it maps the user name identifier used in the SP with the user identifier used in the IdP (principal ID). This configuration is also done attribute resolver through a new type of plugin known as a principal connector

The identity provider currently ships with two principals connectors: the Direct Principal Connector and the Transient Principal Connector. The **Direct Principal Connector** treats the value of the name identifier as the principal name. That is, no mapping between the name identifier and the principal name is provided. The **Transient Principal Connector** determines the principal associated with a name identifier by looking up the principal in the mapping established by the transient ID attribute definition.

In this deployment the Direct Principal Connector was used. **Figure 57** shows its configuration in the *attribute-resolver.xml* file. Note that the name format was defined here too.

```
<resolver:PrincipalConnector id="directPrincipalConnector"
xsi:type="pc:Direct"

nameIDFormat="urn:oasis:names:tc:SAML:2.0:nameid-
format:emailAddress"/>
```

Figure 57: NameID Direct Principal Connector definition in the attribute-resolver.xml file.

Just as another attribute, nameID needs to have a filter policy specified the same way. In this case all name identifiers had to be released and so the defined rule was ANY. As explained in the last section, the attribute ID had to match with the identifier used in the definition of the NameID.

```
<AttributeRule attributeID="principal">
                <PermitValueRule xsi:type="basic:ANY" />
</AttributeRule>
```

Figure 58: Attribute release policy definition for the attribute "principal" in the attribute-filrer.xml file.

The last step is to publish the availability of this new format in metadata. This is done by adding a new <NameIDFormat> to both the IdP's IDPSSODescriptor and AttributeAuthorityDescriptor roles. The value of this element has to match the name format as configured in your name identifier attribute encoder.

In Figure xxx there are the lines hat were added both in the IDPSSODescriptor and the AttributeAuthorityDescriptor.

```
<NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-
format:emailAddress</NameIDFormat>
```

Figure 59: NameID definition in the <IDPSSODescriptor> and <AttributeAuthorityDescriptor> elements in the IdP metadata file

8.3.1.3. Attribute receiving in the SP

To translate received attributes, typically from SAML assertions, Shibboleth SP uses the *attribute-map.xml* file. For every attribute there has to be a new <Attribute/> definition in that file. The name and id of the attribute has to be the same as the name used in the IdP.

To extract the desired content from the attribute the attribute decoders, defined as <AttributeDecoder> are used. The decoder plugin is responsible for instantiating an object that subclasses the required internal class interface, while capturing any data necessary to properly expose the extracted value or values to the SP and protected applications. Shibboleth 2.0 defines three decoders: the String AttributeDecoder, the ScopedAttribute Decoder and the NameID AttrributeDecoder. The **String AttributeDecoder** treats the SAML attribute's values as a simple string and therefore no additional processing is done. The **Scoped AttributeDecoder** treats the SAML attribute's values as a two-part relational construct consisting of a left-hand side (the "value") and a right-hand side (the "scope"). During processing, both halves are tracked independently and exposed either as a flattened string or individually, depending on how the object is being used. Typically, an attribute's scope gives an indication of the domain in which an attribute's value applies; for example, staff@example.org represents a staff member at Example Organization, and the scope is example.org. However, it may not be desirable to allow staff@osu.edu to be asserted by the Brown University IdP. The specialized processing of this decoder facilitates these kinds of distinctions. The **NameID AttributeDecoder** processes SAML attribute values that take the form of a <saml:NameIdentifier> or <saml2:NameID> element. Since SAML identifiers are complex XML types that include XML attributes as well as content, a formatting string must be supplied to instruct the system how to represent the value internally as a flattened string, when necessary.

Since the communication of the user email was being done using the SAML 2.0 NameID, the decoder used was the NameID AttributeDecoder. In the definition of the attribute the id had to be the same as the one specified in the IdP. To format the email correctly formatting string "$NameQualifier$SPNameQualifier$Name" was used to turn the XML content into a flat string

```
<Attribute                    name="urn:oasis:names:tc:SAML:2.0:nameid-
format:emailAddress" id="principal">
                <AttributeDecoder    xsi:type="NameIDAttributeDecoder"
formatter="$NameQualifier$SPNameQualifier$Name"/>
          </Attribute>
```

Figure 60: NameID attribute decoder definition in the attribute-map.xml file.

Just like in the IdP, it was necessary to publish the required NameID needed in the SP inside the <SPPSSODescriptor>.

```
      <md:NameIDFormat>
                urn:oasis:names:tc:SAML:2.0:nameid-format:emailAddress
          </md:NameIDFormat>
```

Figure 61: NameID definition in the <SPSSODescriptor> element in the SP metadata file.

Finally the SP had in its possession the authenticated user email. So now it was only needed top populate the REMOT_USER HTTP header with the email value. This is configured in the *shibboleth2.xml* file inside an application definition. Since in this project there was only one application defined in the SP it was configured in the Application defaults definition. The only thing that had to be done was to indicate that the REMOTE_USER would take the value of the attribute defined in the *attribute-map.xml* identified by the name "principal". **Figure 62** shows the new definition with the line REMOTE_USER="principal" added.

```
<ApplicationDefaults id="default" policyId="default"
      entityID="https://tokyo.fib.upc.edu/shibboleth"
      homeURL="http://tokyo.fib.upc.edu/portal/index.jsp"
      REMOTE_USER="principal"
        >
...
</ApplicationDefaults>
```

Figure 62: Application definition in the shibboleth2.xml file with the population of the REMOTE_USER HTTP header

8.3.2. Lazy Session

Another change that had to be done was that the user did not have to authenticate himself to access the platform but only when he clicked on the login link. Shibboleth allows applications to request that a session to be created on demand by redirecting a user to the URL of a <SessionInitiator>. This is called lazy session initiation.

This is the tricky step for integration. It's the position of the Shibboleth project that all web applications should require no specific programming for any specific authentication or authorization mechanism, relying instead on variables provisioned by the web environment. This includes Shibboleth, which has no external API; it only sets header/environment variables and sends the user to the application. For this the SessionInitiator mechanism does support a simple redirect protocol capable of triggering, and influencing, the creation of authentication requests.

This protocol supports a small set of query string parameters that correspond to identically named attributes that can be supplied either directly on a <SessionInitiator> element or as content settings on a per-resource basis.

The first change done was in the *tomcat.conf* file where the access control had been defined before. ShibRequireSession was set to off, require valid-user was deleted and require shibboleth was added. The require shibboleth deferred all decisions about everything to *shibboleth2.xml* <RequestMap> where the shibboleth authentication would be turned to off too.

```
<Location /portal>
  AuthType shibboleth
  require shibboleth
  ShibRequireSession Off
  ShibUseHeaders On
</Location>
```

Figure 63: Apache configuration with the required authenticated user option turned off in the tomcat.conf file.

In the Request Mapper the requireSession was set to false thus allowing non authenticated users to access to the platform.

```
    <RequestMapper type="Native">
        <RequestMap applicationId="default">
                <Host name="tokyo.fib.upc.edu"  authType="shibboleth"
requireSession="false"/>

      </RequestMap>
    </RequestMapper>
```

Figure 64: Request Mapper definition with the required authenticated user option turned off in the shibboleth2.xml file.

At this point the SP was completely configured to work with lazy session. The only thing remaining was to change the login link in the e-Catalunya platform. The new login URL had to point to the Session Initiator defined in the *shibboleth2.xml* file. This URL was the concatenation of the following elements:

- Host name= tokyo.fib.upc.edu
- handlerURL= /Shibboleth.sso
- Session Initiator Location=/Login (defined in the Session Initiator definition)

To indicate the URL to be redirected after the authentication in the IdP a protocol defines by IdP was used. At the end of the new formed URL was added "?target=DESIRED_URL". Thus the Login link had the following URL: https://tokyo.fib.upc.edu/Shibboleth.sso/Login?target=$url_back"

8.4. Single Logout issue

Although before releasing Shibboleth 2.0 it was announced that Single Logout (or Single Sign-Out) would be supported, currently the Single Logout profile it is not implemented. Nevertheless it seems that in future releases Single Logout is going to be supported.

The reason of this missing feature is due to the difficulties and problems that Single Logout presents both in the user experience and in the technical context.

8.4.1. User experience problems

One of the first issues with Single Logout (SLO) is communicating to the user what will occur if they click "logout". Users have already been taught that doing this will cause them to be logged out of the application that contained the logout button/link they used. They assume that all other applications will remain active. Therefore there must be some means to distinguish between application-level logout and SLO.

This issue may be solved by placing a branded, distinct, image/button in place of the application-level logout link/button. This approach however suffers from two problems. First it requires educating users about this UI component and what it is meant to convey. Second, it requires all SPs display the same UI component, which is unlikely to happen.

Another UI issue occurs during SP-initiated SLO. In this case it is the SPs responsibility to provide the final indication to the user about the success or failure of SLO. However, the amount of information that the SP gets back from the IdP is very limited; a URI that either indicates complete success or one that indicates some failure in the process. In the case where SLO fails the SP will have little information with which to tailor what it displays to the user since it will not know the cause of the failure. If a front-channel binding is used this problem can become worse. In the event that a SP is not responsive the user will be presented with an HTTP error. At this point neither the IdP nor the SP present information to the user.

8.4.2. Technical problems

Continuing with the last issue mentioned in section 7.3.1, another problem that may be found when an HTTP error is encountered during the use of a front-channel binding is, if this error is due to a not responsive SP, the process breaks down and the remaining SPs will not receive logout requests. So, in addition to not being able to tell the user what has happened there are also SPs, with active sessions, that are unaware that a failed logout attempt has occurred.

Finally, there is the problem that only SAML 2 supports single logout. If any entity supporting only SAML 1 is participating in the SSO session they will be left out of the logout process. Attempting to explain to a user which services support SLO and which do not is unlikely to be successful.

8.4.3. Deployed Logout

Shibboleth 2.0 defines the <LogoutInitiator> element used to configure handlers that are responsible for initiating a logout operation. The handler is responsible for performing protocol-specific tasks related to the logout, as well as terminating the session. Shibboleth specifies two logout connectors: local and global. **Local logout** means that the SP's session is removed, but no communication with the IdP or other SPs is involved. **Global logout** implies that the IdP is also informed of the logout operation. The Global Logout defined is the **SAML2 LogoutInitiator**. If the user's session was initiated with a protocol other than SAML 2, then the handler ignores the request. Otherwise, the initiating entityID is used to check for metadata with an <md:IDPSSODescriptor> role supporting SAML 2.0 and a compatible <md:SingleLogoutService> endpoint. The absence of either causes a warning to be logged and the handler otherwise ignores the request. SAML 2 LogoutInitiator does not implement, as already said, single Logout, since the other SPs that may have active sessions for this user are not going to be notified.

Since in this project the user session was configured to initiate with a SAML 2.0 protocol, the SAML2 LogoutInitiator could be used.

```
   <LogoutInitiator type="Chaining" Location="/Logout" r
relayState="cookie">

     <LogoutInitiator type="SAML2"/>

</LogoutInitiator>
```

Figure 65: SAML 2 Logout Initiator definition in the shibboleth2.xml file.

Like in the Lazy Session the Logout URL were changed by a new one that pointed to the LogoutInitiator. This URL was build just like in the SessionInitiator. The resulting URL was: https://tokyo.fib.upc.edu/Shibboleth.sso/Logout.

8.5. Final Result

To finish this chapter here is the general workflow of the login process of a user in the federated e-Catalunya without going into much detail.

First of all, a user accesses a public area of e-Catalunya. Then Shibboleth checks whether authentication is needed or not. First place to check is in the apache configuration file (here *tomcat.conf*). In case that in the apache configuration file the authentication is specified as not required Shibboleth checks in the *shibboleth2.xml* file the <RequestMap>.

Figure 66 shows the e-Catalunya platform with the Shibboleth login link framed in red. Until the user does not click into this link the SP will not do another thing than the action that has just been explained. When the user clicks into the Shibboleth login link the session initiator being referenced by this link figures out which IdP the user will authenticate with and what protocols to use (in this case the Authentication Request Protocol is used). So the user is redirected to the IdP issuing an authentication request and using the SSO Browser/Post Profile.

Then the IdP examines the request and decides that the user has to be authenticated using a user/password method. This decision is based on the configuration of the e-Catalunya SP in the *relaying-party.xml* file. The user is redirected to the right login handler, defined in the *login.config* file, authenticates through the selected method, and comes back to the profile handler with his username set. **Figure 67** shows the displayed page for the username/password authentication method.

Then the IdP consults the *attribute-resolver.xml* file to see if it has to send some attribute to the SP. In this case the IdP has to send the user name identifier. It checks again the *attribute-resolver.xml* file to see what kind of connector is used for this name identifier. Since it is used a direct connector the IdP does not have to do anything: the user name identifier that is going to be sent to the SP is going to be the same as the user name in the IdP.

After checking the *attribute-policy.xml* to see if the name identifier may be released to the specific SP the IdP sends the name identifier in a SAML 2.0 assertion. This assertion is signed with the IdP's key and encrypted with the SP's key for security and privacy. The assertion is placed into a message and the user is redirected to the SP carrying it along.

The user ends up at an assertion consumer service at the SP. It unpacks the message, decrypts the assertion, and performs several security checks. If

everything's in order, then it will extract the user name identifier from the message and translate it into the REMOTE_USER header variable.

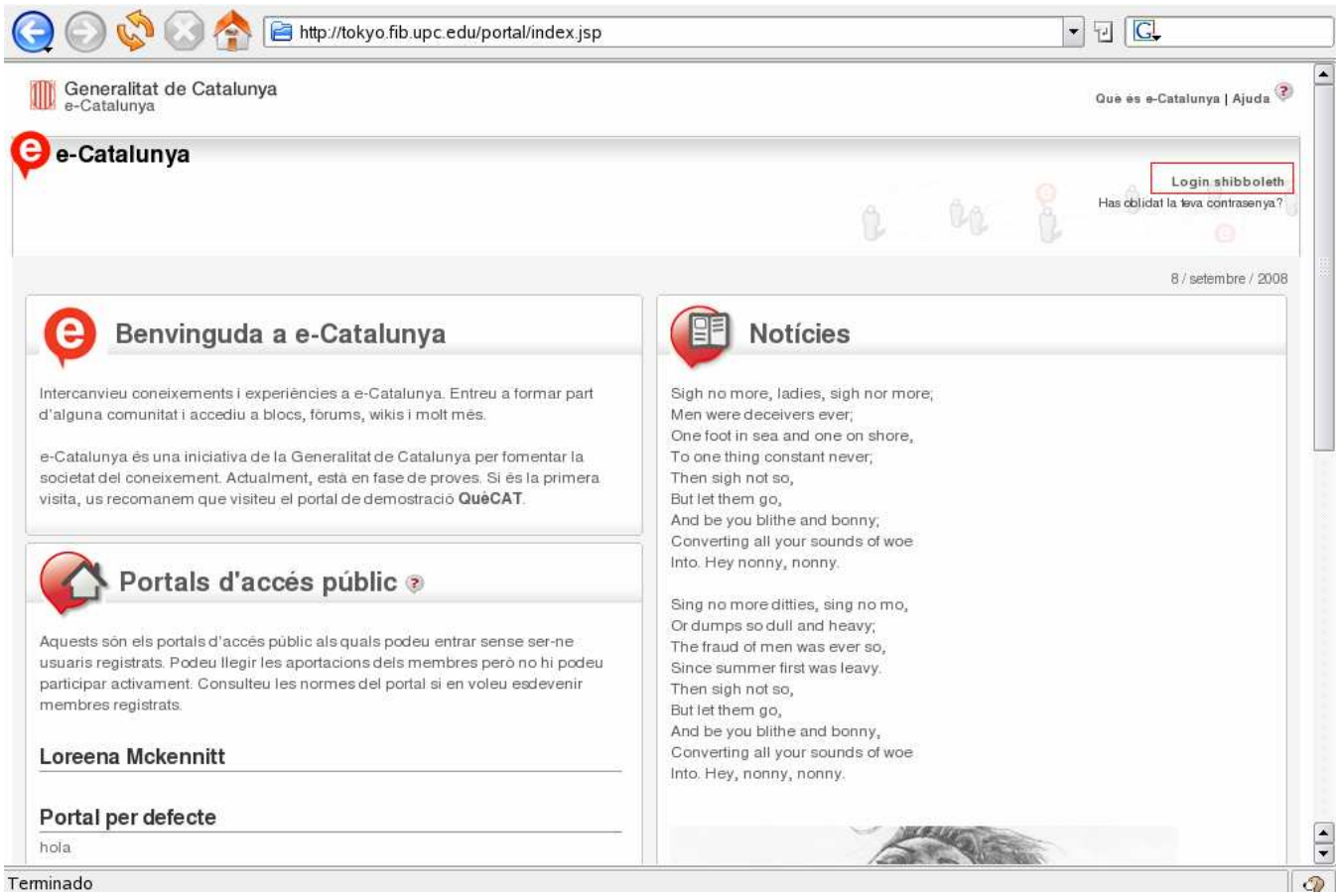

Figure 66: e-Catalunya portal index page with a link for Shibboleth 2.0 authentication

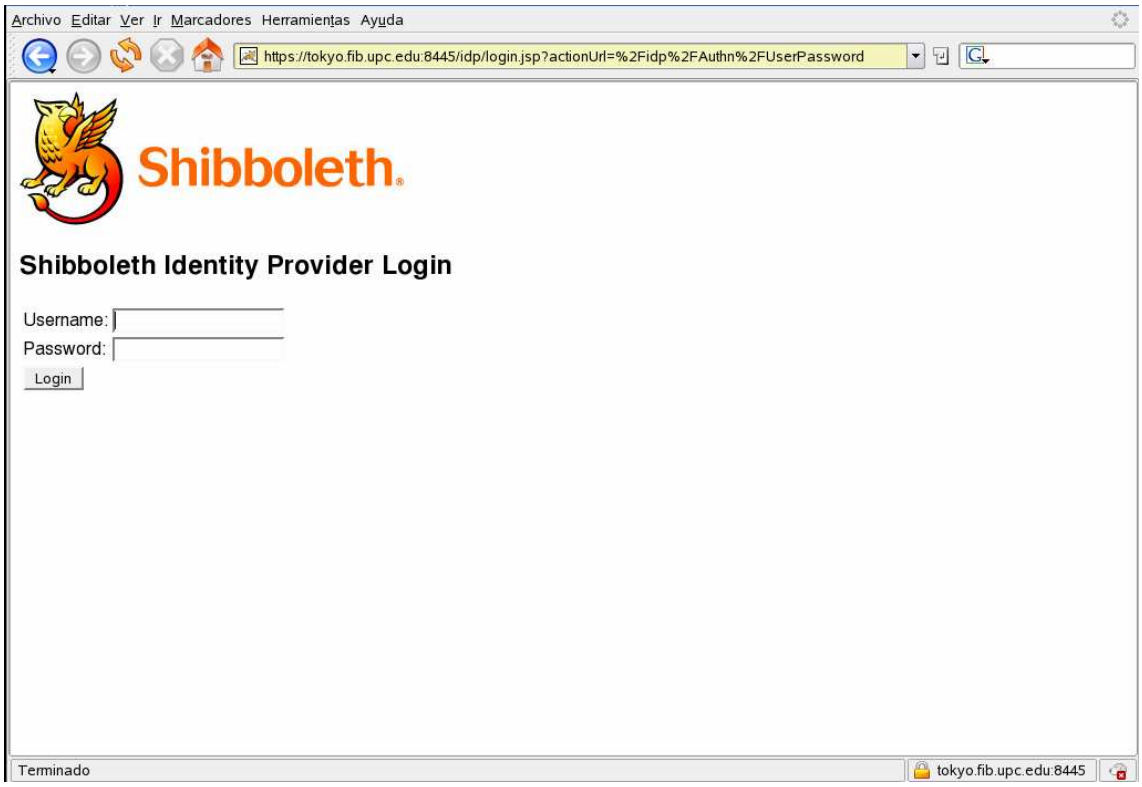

Figure 67: Shibboleth username/password authentication default page

9. Project Analysis and Conclusions

This chapter shows the project cost and its initial time planning compared to the final one. It also describes the possible future prospects for this project, the accomplished goals and finally, the personal conclusions of the author.

9.1. Project Cost

The cost of the project has been divided in human resources costs, hardware costs and software costs.

9.1.1. Human resources costs

The development of this project included two distinguished phases that required different skills.

One was the analysis which comprised the initial study, the specification research and comparison, the study of some of the current federation implementations, the choice of Shibboleth 2.0 and the decision of how Shibboleth 2.0 was going to be deployed inside the e-Catalunya.

The other was the development or the deploying of Shibboleth 2.0.

The calculation of the human resources costs has taken these two phases into account implying the need of a system analyst and a developer. **Table 1** displays the human resource costs detailing the professional profile of the worker, accumulated work time and estimated per hour salary.

	Hours	Price/Hour	Cost
System Analyst	500	35 €	**17500 €**
Developer	340	25 €	**8500 €**
Total			**26000 €**

Table 2: Human resources costs.

9.1.2. Hardware costs

For this project it was only required one personal computer in the hardware context. Table 3 displays the hardware cost for this project.

Hardware	Cost
Personal Computer	**1.000 €**
Total	**1.000 €**

Table 3: Hardware costs.

9.1.3. Software costs

Nearly all software used during the development of this project was free software, except for the Vmware Workstation used for the deployment of Shibboleth 2.0. Table 4 displays the software costs.

	Cost
Vmware Workstation Licence	**130 €**
Total	**130 €**

Table 4: Software costs.

9.1.4. Total Cost

Table 5 displays the total cost calculated with the human resources cost, the hardware cost and the software cost.

Resources	Cost
Human resources	**26.000 €**
Hardware	**1.000 €**
Software	**130 €**
Total	**27130 €**

Table 5: Total costs.

9.2. Project Planning

The planning of this project was very hard to do. At the beginning the terms of "federation" or "federated identity" were completely alien and there was no basic knowledge to know if they implied a large amount of information or not or the complexity of it.

9.2.1. First stage planning

Approximately two weeks were necessary to assess the magnitude of the federated identity management infrastructure. It was at that point when it was done an initial planning of the study of the federation model. Since it was not decided yet how the pilot test federation was going to be done (using an API or installing software product) in this fist stage was not planned the deployment except for estimating its starting and ending time. The following gantt chart, **Figure 68**, shows this initial first stage planning with the first two weeks included.

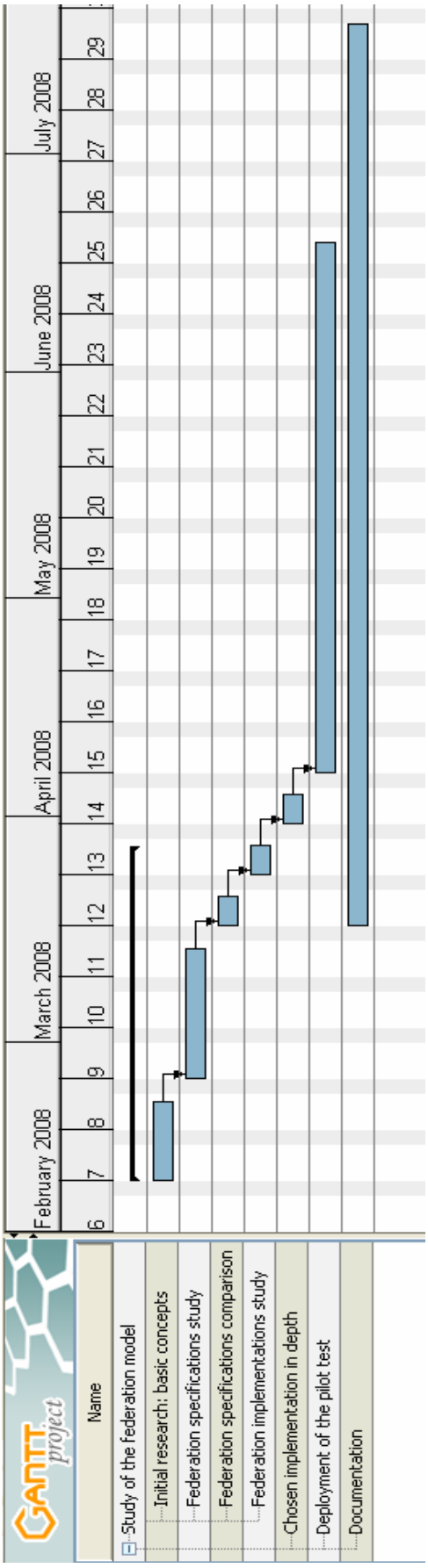

Figure 68: Gantt chart with the initial time panning for the first phase of the project .

The following sections summarize the tasks defined in the gannt chart.

9.2.1.1. Initial research: basic concepts

Initial study of the federated identity management concept. This phase includes the description of a federation and its motivations as well as the discovering of the current specifications and some of its implementations.

9.2.1.2. Federation specifications study

Study of the most important federation specifications. The specifications studies in this phase are SAML 1.0, 1.1 and 2.0, Liberty Alliance specifications, WS-Federation and OpenID.

The estimated time for this task was three weeks.

9.2.1.3. Federation specifications comparison

Comparison of the main aspects of the specifications overviewed in the previous task.

The estimated time for this task was one week.

9.2.1.4. Federation implementations study

Study of some of the most popular products that implement a federated identity management system. The study is most focused on the reliability of each product more than its technical features.

The estimated time for this task was one week.

9.2.2. First Stage planning revision

At the end of this first stage there had been to deviations in the initial time planning.

The first one was related with the comparison of the federation specifications. This task actually started during the previous task, the study of each specification. This was due to the complexity of the specifications being reviewed. It was easier to mark the main aspects of each one as well as its resemblances and differences as they were being studied.

The second deviation retarded the ending of this first stage one week. The implementations studied resulted to be more complicated than what was first thought due to the difficulty of finding useful information or information at all. Some of the implementations that were researched are not included in this document because of not finding any technical information.

Figure 69 is a gantt chart with the real time spent on each task.

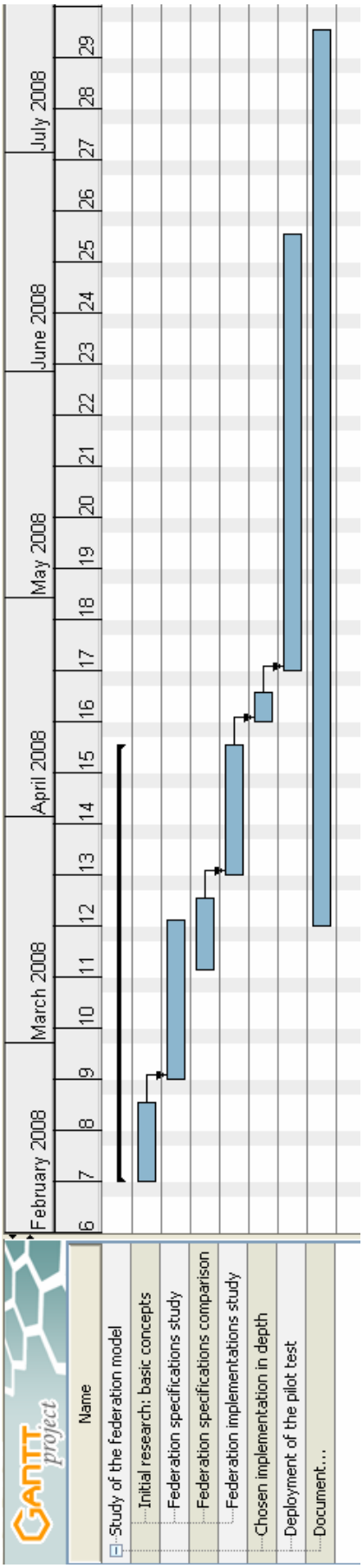

Figure 69: Gantt chart with the real time dedication per task of the first phase of the project.

9.2.3. Second Stage planning

The planning of this second stage was done once an implementation product was chosen. Figure 70 shows the initial planning and its tasks.

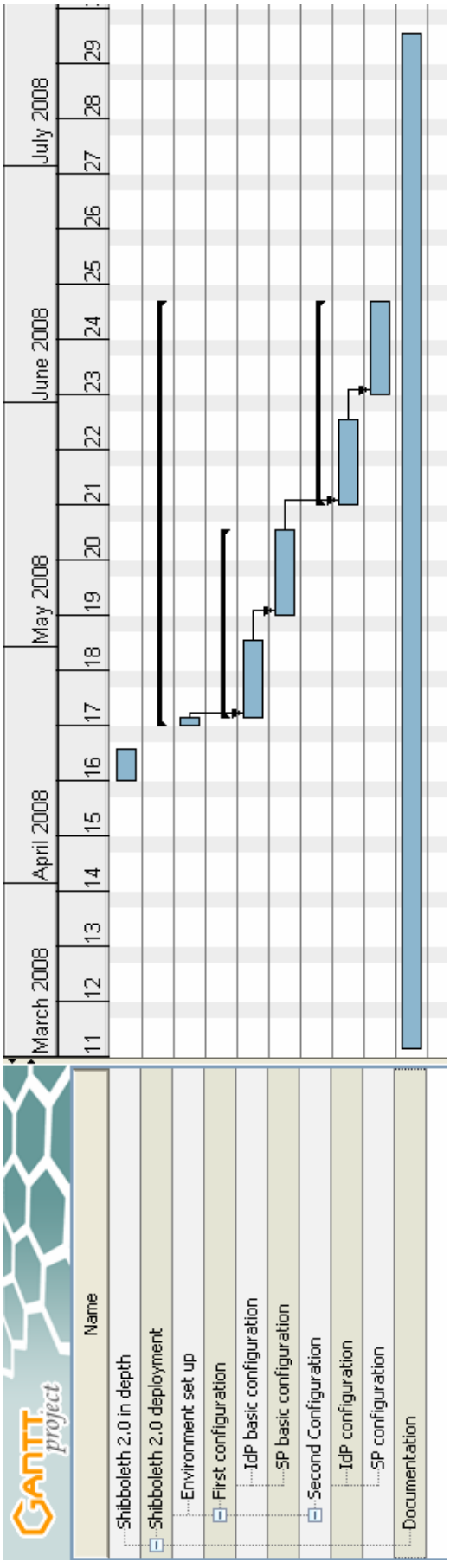

Figure 70: Gantt chart with the initial time panning for the second phase of the project

The following sections summarize the tasks defined in the gannt chart.

9.2.3.1. Shibboleth 2.0 in depth

This phase is a Shibboleth 2.0 study more extended than the one done in the federation implementations study task. It studies the components that compose Shibboleth 2.0, its relationships between them and their possible configurations.

9.2.3.2. Shibboleth 2.0 deployment

This task is the development of the pilot test to study the viability of federating e-Catalunya.

9.2.3.2.1. Environment set up

Installation of all the software required to develop the pilot test, such as the Vmware Workstation and the OpenSuse operative system.

9.2.3.2.2. First configuration

Installation and initial configuration of the Shibboleth 2.0 Identity Provider and Service Provider. This first configuration is easier than the one needed to federate correctly e-Catalunya, but it is done to test the correct communication between the SP and IdP before attempting the desired one.

9.2.3.2.3. Second configuration

Second and final configuration of both the IdP and the SP. At the end of this phase the e-Catalunya must be ready to be federated.

9.2.3.3. Documentation

Documentation includes internal reports, personal notes as well as the final master thesis report.

9.2.4. Second Stage planning revision

This second stage ended two weeks latter than what was initially planned. Except for the environment set up task, all the other tasks estimated time were exceeded. In the end the study of Shibboleth 2.0 was done while its deployment so it would be easier to understand.

The delay in the deployment was mainly due to misconfigurations hard to detect. The configuration of Shibboleth 2.0 resulted to be more difficult that what was expected.

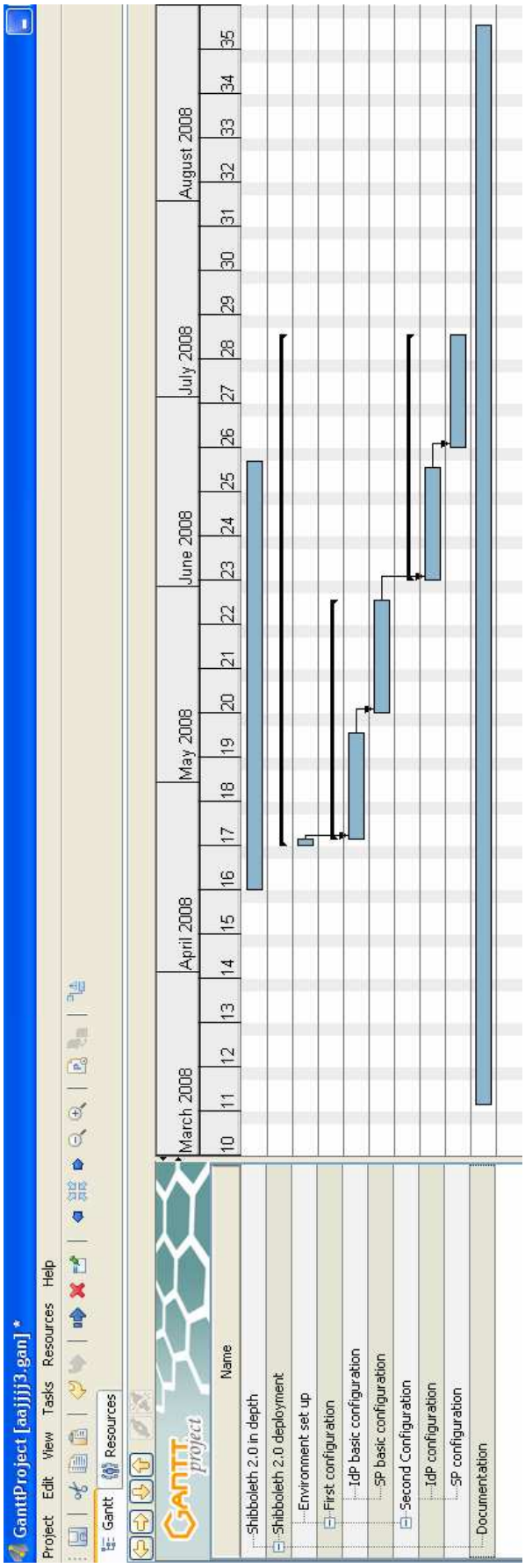

Figure 71: Gantt chart with the real time dedication per task of the second phase of the project.

9.3. Achieved Objectives

At the end of this project both main objectives have been accomplished.

This thesis provides to the LCFIB a general study about the federation identity model. The most important federation specifications are reviewed and compared remarking some of the main features of a federated identity management system. Also, some implementations of the federated model are overviewed emphasizing who their developers are, what kind of technical support they offer, how many federations are built with them and how much useful documentation they have.

Likewise the pilot test was finished and tested. With the deployment of Shibboleth 2.0 a simulation of a federated e-Catalunya was achieved with minimal changes inside the e-Catalunya platform.

9.4. Future Lines

This project offers a great deal of possible future approaches.

First of all, the study of the federation model along with its specifications and some of its implementations can be iteratively reviewed. Federated identity management is in one of its highest moments and has a continuous incoming of new information. Therefore the study of federation specifications and new implementations is probably none ever ending, at least for a few years. This of course is not an isolated phenomenon in the technological world which is notable for its constant evolution.

Focusing in the pilot test some of the following aspects could be done:

- Testing other SAML 2.0 bindings and profiles that the one used in the pilot test like the SAML 2.0 Single Sign-On Browser/Artifact Profile or in the Single Sign-On Browser/Post Profile use the HTTP POST Binding instead of the HTTP Redirect Binding.
- Build specific trust credentials for both providers (Identity Providers and Service Provider) instead of using the ones that are generated during the installation process.
- Installing and configuring the Shibboleth Discovery Service.
- Study the possibility of implementing a Single Logout mechanism that could be integrated in the Shibboleth 2.0 system.

Future versions of Shibboleth should be checked since in theory Single Logout is going to be implemented and could be interesting to exploit it.

Another interesting future prospect could be the study of OpenID for some of the ECAT Platform distributions. OpenID was turned down due to its simplicity and some of its security issues that were not appropriate for the e-Catalunya platform. But OpenID is a good solution for some contexts where nothing complex is needed and privacy and security are not the principal goals. Moreover some popular web communities, such as LiveJournal and Blogger, enable OpenID authentication which would allow the ECAT Platform to be federated with them.

Finally the last, but not the least of the future prospects, is the actual deployment of a federation with the e-Catalunya platform and other organizations to offer more and functionalities both to the e-Catalunya users and to the other organization users.

9.5. Personal Conclusions

Now, at the end of these six months of work and research, it might be the perfect time to ask oneself: "And what about it?" Maybe it sounds like a strange question; let me rephrase it: "What has this meant to you and what is this important for?" Or maybe the question is quite easier; "What have you learnt?"

I have learnt lots of things; both in the technical context and in the personal one and to try to write them down would probably take more than what is intended here. So I will try to summarize this six month experience and to remark only those points that are interesting for the reader.

I would like to start speaking about the e-Catalunya platform. Although I haven't had to manipulate the platform directly, unlike someone that is developing a new tool for example, I learned quite a lot with it. e-Catalunya is an interesting and complex project that integrates many different technologies together. You can spend months just getting familiarized with it, and when you think that you know it all then "puff!!" you discover something new. Discovering all its components, how they interact, how they are integrated within the platform, and why they are integrated in that specific way instead of another, along with other architectonical and design aspects, was quite a learning experience.

For understanding the design of e-Catalunya the knowledge I had acquired in Enginyeria del Software 2 (ES2) and in Disseny de Sistemes Basats en Web (DSBW) was much helpful and I was able to extend it, specifically learning new

design patterns and new agile development methodologies as well as studying ones that I already knew, in a real case.

Moreover e-Catalunya introduced me directly into the world of the Web 2.0 and the new social phenomenon it is creating. Personally I think that e-Catalunya is one of the best examples when talking about the new trends of the Web. Having read the "Weaving the Web: Origins and Future of the World Wide Web" of the Tim Berners-Lee I dare to say that e-Catalunya accomplishes much of the objectives and expectations that Mr. Berners-Lee had in mind.

Regarding the federation model there is much to say. To claim that studying it has given me plenty of knowledge is obvious, since six month ago I had never heard of federations or Single Sign-On. The world of federations is a large and interesting one. It is interesting for the technology behind it as well as for its social impact in the net. To study and understand it was actually a complex task. But as I was advancing though my study the concepts became clearer and I began to have a more structured and organized knowledge about the subject. The more I understood, the more it interested me and the more I realized that the work that had been done to define and implement it was remarkable and outstanding.

In addition, the prospect the federation model offers is an amazing one. To imagine the services in the web interconnected between them, enriching the user experience and at the same time, easing the work for the system administrators and developers, is quite interesting.

With the Shibboleth 2.0 deployment I was able to put in practice some of the theoretical knowledge I had learnt in subjects like Projecte de Xarxes i Computadors (PXC) and Seguretat en Sistemes Informàtics (SSI). Shibboleth 2.0 is a complex system that requires advanced knowledge in system administration, Apache server and Tomcat server. Consequently not only did I put in practice what I had previously learnt in classes, but I enhanced it.

Needless to say, this research did not only enrich me with technical knowledge. It taught me (and I mean really, really taught me) the importance of good planning, good communication, good organization and most important, good management of the information.

Last, but not the least, I would like to comment the great experience of working in the LCFIB: It probably has been the greatest of all. In the e-Catalunya project I had the chance to experience team work in a real environment for the first time. To work with such exceptional people, most of them with lots of more experience behind their backs than me, who where always ready to give a helping hand, enriched me in a technical aspect as well as in personal.

10. Bibliography

P. Hallam-Baker et al. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)" <http://www.oasis-open.org/committees/security/ >, OASIS Security Services Technical Committee, November 2002.

P. Mishra et al. "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)" <http://www.oasis-open.org/committees/security/>, OASIS Security Services Technical Committee, November 2002.

J. Hodges et al. "Glossary for the OASIS Security Assertion Markup Language (SAML)" <http://www.oasis-open.org/committees/security/>, OASIS Security Services Technical Committee, November 2002.

S. Cantor et al. "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0" <http://www.oasis-open.org/committees/security/>, OASIS Security Services Technical Committee, March 2005.

S. Cantor et al. "Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0." <http://www.oasis-open.org/committees/security/>, OASIS Security Services Technical Committee, March 2005.

S. Cantor et al. "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0." <http://www.oasis-open.org/committees/security/>, OASIS Security Services Technical Committee, March 2005.

J. Hodges et al. "Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0." <http://www.oasis-open.org/committees/security/>, OASIS Security Services Technical Committee, March 2005.

S. Cantor et al. "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0." <http://www.oasis-open.org/committees/security/>, OASIS Security Services Technical Committee, March 2005.

J. Kemp et al. "Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0" <http://www.oasis-open.org/committees/security/>, OASIS Security Services Technical Committee, March 2005.

Cantor, Scott, Kemp, John, Champagne, Darryl, eds. "Liberty ID-FF Bindings and Profile specification," Version 1.2-errata-v2.0, <http://www.projectliberty.org/specs>, Liberty Alliance Project, September 2004.

Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema specification," Version 1.2-errata-v3.0, <http://www.projectliberty.org/specs>, Liberty Alliance Project, December 2004.

Madsen, Paul, eds. "Liberty ID-FF Authentication Context Specification," Version 1.3, <http://www.projectliberty.org/specs>, Liberty Alliance Project, December 2004.

Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.1, <http://www.projectliberty.org/specs>, Liberty Alliance Project, December 2004.

C. Cahill. "Identity Management Standards", <http://www.projectliberty.org>, Liberty Alliance Project, August 2007.

Liberty Alliance. "Liberty Alliance & WS-Federation: A Comparative Overview", <http://www.projectliberty.org/specs>, Liberty Alliance Project, October 2003.

Hodges, Jeff, eds. "Liberty Technical Glossary," Version 1.4, <http://www.projectliberty.org/specs>, Liberty Alliance Project , December 2004.

BEA, IBM, Microsoft, RSA Security, VeriSign. "Web Services Federation Language: Active Requestor Profile", <http://www.ibm.com/developerworks/library/specification/ws-fed/>, July 2003

BEA, IBM, Microsoft, RSA Security, VeriSign "Web Services Federation Language: Passive Requestor Profile", <http://www.ibm.com/developerworks/library/specification/ws-fed/>, July 2003

BEA, IBM, Microsoft, RSA Security, VeriSign. "Web Services Federation Language", <http://www.ibm.com/developerworks/library/specification/ws-fed/>, July 2003

specs@openid.net. "OpenID Authentication 2.0 - Final" <http://openid.net/specs/>, OpenID, August 2007.

specs@openid.net. "OpenID Attribute Exchange 1.0 – Final", <http://openid.net/specs/>, OpenID, December 5, 2007.

S. Cantor et al. "Shibboleth Architecture: Protocols and Profiles." <http://shibboleth.internet2.edu/>, Internet2-MACE, February 2005

Internet2. "Shibboleth Protocol Specification", <https://spaces.internet2.edu/display/SHIB2/TechnicalSpecs>, Shibboleth, September 2005.

Differences between SAML 2.0 and 1.1 (2008). SAML XML.org [Website]. Avaliable from: <http://saml.xml.org/differences-between-saml-2-0-and-1-1>

S. Brands. "The problem(s) with OpenID (2007)", Identity corner [Website]. Available from: <http://idcorner.org/2007/08/22/the-problems-with-openid/>

B. Forrest. "Pros and Cons of OpenID (2007)" O'Reilly Radar [Website]. Available from: <http://radar.oreilly.com/2007/02/pros-and-cons-of-openid.html>

B. Laurie. "OpenID: Phishing Heaven (2007)" Links [Website]. Available from: <http://www.links.org/?p=187>

A. Tulshibagwale. "Steps for choosing the right federated identity standard for your company (2005)" Computer World [Website]. Available from: <http://www.computerworld.com/securitytopics/security/story/0,10801,105019,00.html>

E Norlin, D Platt. "Federated Identity Standards. Confused? You bet you are (2004)", SOA World Magazine [Website]. Avaliable from: <http://soa.sys-con.com/node/46566>

A. Le Van Gong. "Deep-dive on SAML 2.0 vs WS-Federation 2007", blogs.sun.com [Website]. Avaliable from: <http://blogs.sun.com/hubertsblog/entry/deep_dive_on_saml_2>

J.Hodges. "Technical Comparison: OpenID and SAML - Draft 06 (2008)", IdentityMeme.org [Website]. Avaliable from: <http://identitymeme.org/doc/draft-hodges-saml-openid-compare-06.html>

A. Buecker, W. Filip, H. Hinton, H.P. Hippenstiel, M. Hollin, R Neucom, S. Weeden, J. Westman. "Federated Identity Management and Web Services. Security with IBM Tivoli Security Solutions", October 2005.

F. Yin. "Universal Federation Architecture Enabling Unified Identity Federation & Web Services Security", December 2006.

J.H. Clippinger. "Higgins: Towards a foundation layer for the social Web", June 2006.

S. Shah. "The JBoss Federated SSO Framework JBoss", JBOSS, http://jbossworld.com>, February 2008.

J. Reede. "On A-Select and Federated Identity Management Systems", Master's Thesis, University of Twente. Supervisors: Dr. M.U. Reichert, A. Wombacher. August 2007.

G. Huntington. "Creating a Federated Authentication Trust (2006)", Authentication World [Website]. Available from: <http://www.authenticationworld.com/Authentication-Federation/CreatingAFederatedAuthenticationTrust.pdf>

PingIdentity. "Federated Identity Deployment Architecture White Paper", PingIdentity [Website]. Available from: <www.pingidentity.com/information-library/white-papers>, September 2006.

I. Stenberg. "Server Based Computing and Web Single Sign-On integration", Master's Thesis, Helsinki University of Technology, Supervisors: Professor Heikki Saikkonen

PAPI development team. "A Detailed Description of the PAPI protocol" RedIris [Website]. Available from: **<**http://papi.rediris.es/rep/PAPI_Protocol_Detailed.pdf>

Wikipedia: http://es.wikipedia.org/wiki/Wiki

Entr'ouvert: http://www.entrouvert.com/en/

Higgins: http://www.eclipse.org/higgins/

JBoss Federation: http://wiki.jboss.org/wiki/FederationServer

OpenSSO: https://opensso.dev.java.net/

ZXID: http://www.zxid.org/

PAPI: http://papi.rediris.es/

Shibboleth 2.0: https://spaces.internet2.edu/display/SHIB2/Home

11. Appendix

11.1. Glossary

11.1.1. Abbreviations

DCE: Distributed Computing Environment

ECP: Enhanced Client or Proxy

GUID: Globally Unique Identifiers

ID-FF: Identity Federation Framework

IdP: Identity Provider

ID-SIS: Identity Service Interface Specification

ID-WS: Identity Web Services Framework

LDAP: Lightweight Directory Access Protocol

LECP: Liberty-enabled client and proxy profile

NCS: Network Computing System

OASIS: Organization for the Advancement of Structured Information Standards

OP: OpenID Provider

OSF: Open Software Foundation's

PAOS: Reverse SOAP

PKI: Public Key Infrastructure

RP: Relaying Party

SAML: Security Assertion Markup Language

SLO: Single Logout

SOAP: Simple Object Access Protocol

SP: Service Provider

SSL: Secure Socket Layer

SSO: Single Sign-on

STS: Security Token Service

TLS: Transport Layer Security

UUID: Universally Unique Identifier

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

XACML: eXtensible Access Control Markup Language

XML: Extensible Markup Language

XRDS: eXtensible Resource Descriptor Sequence

XRI: eXtensible Resource Identifier

WAP: Wireless Application Protocol

WML: Wireless Markup Language

WSDL: *Web Services Description Language*

11.1.2. Terminology

Account linking: A method of relating accounts at two different providers that represent the same principal so that the providers can communicate about the principal.

Anonymity: The quality or state of being anonymous, which is the condition of having a name or identity that is unknown or concealed.

Artifact: opaque reference to an assertion that is sent from the Identity Provider to the Service provider. The Service Provider dereferences it to acquire the original assertion at the Identity Provider through direct communication.

Assertion: A statement that is taken as being correct or true.

Attribute: An attribute is an atom of information which is defined by the intersection of attribute name and attribute value. Attributes must be considered in terms of the subject about whom they are asserted and the authority who is asserting the atom of information is true.

Attribute Authority: A system entity that produces attribute assertions.

Attribute Assertion: An assertion that conveys information about attributes of a subject.

Authentication: Authentication refers to the confirmation that a user who is requesting services is a valid user of the network services requested. Authentication is accomplished via the presentation of an identity and credentials.

Authentication Assertion: An assertion that conveys information about a successful act of authentication that took place for a subject.

Authentication Authority: A system entity that produces authentication assertions.

Authorization: Authorization refers to the granting of specific types of service to a user, based on their authentication, what services they are requesting, and the current system state.

Authorization Decision Assertion: An assertion that conveys information about an authorization decision.

Back Channel, Direct Communication: Back channel refers to direct communications between two system entities without "redirecting" messages through another system entity such as an HTTP client

Credentials: Data that is transferred to establish a claimed principal identity.

End User: A natural person who makes use of resources for application purposes

Federated Identity: A principal's identity is said to be federated between a set of Providers when there is an agreement between the providers on a set of identifiers and/or attributes to use to refer to the Principal

Federate: To link or bind two or more entities together

Federation: A federation is a collection of organizations that agree to interoperate under a certain rule set to share user identity information. Federations will usually define trusted roots, authorities, and attributes, along with distribution of metadata representing this information.

Front Channel, Indirect Communication: Front Channel refers to the "communications channel" that can be established between two HTTP-speaking servers by employing "HTTP redirect" messages and thus passing messages to each other via a user agent, e.g. a web browser, or any other HTTP client.

Identifier: A data object mapped to a system entity that uniquely refers to the system entity. A system entity may have multiple distinct identifiers referring to it. An identifier is essentially a "distinguished attribute" of an entity.

Identity Provider: A kind of service provider that creates, maintains, and manages identity information for principals and provides principal authentication to other service providers within a federation.

Login, Logon, Sign-On: The process whereby a user presents credentials to an authentication authority, establishes a simple session, and optionally establishes a rich session.

Logout, Logoff, Sign-Off: The process whereby a *user* signifies desire to terminate a simple session or rich session.

OpenID Provider: An OpenID Authentication server on which a Relying Party relies for an assertion that the end user controls an Identifier.

Persistent Pseudonym: A privacy-preserving name identifier assigned by a provider to identify a principal to a given relying party for an extended period of time that spans multiple sessions.

Principal: General term for a party that benefits from being able to authenticate itself. This can be a tangible individual, also known as a user, or an abstract process such as a web service.

Private Key: A private key is a value - known only to one party – that can be used to decrypt encrypted messages, issue digital signatures and compute the corresponding public key. The private key must be kept private and must not be made publicly available. This term is most often used in the context of public key cryptography and not in the context of traditional cryptography.

Provider: A generic way to refer to both identity providers and service providers.

Public Key A public key is a value that can be used to effectively encrypt messages and verify digital signatures. The public key can be made publicly available; it does not contain secret information.

Public Key Certificate: Electronic document which incorporates a digital signature to bind together a public key with an identity

Public Key Cryptography Public key cryptography is the science of information security that uses private key and public key pairs for encryption, decryption and signature creation and verification. The problem of the key distribution is solved because the public key can be made publicly available, just the private key is kept as a secret.

Relying Party The relying party is defined per-flow and is always the provider receiving and utilizing information from another entity in a given flow. Generally, this will be a particular SP.

Role: The actions and activities assigned to or required or expected of a person or an entity.

Resource: A resource in our sense can be a piece of data or a service provided by a system

Single Logout: Reverse action of Single Sign-On. Once a user is logged in a system using Single Sign-On, Single Logout deletes all the sessions the user had with all systems inside the Single Sign-On domain.

Service Provider: A role donned by a system entity where the system entity provides services to principals or other system entities.

Single Sign-On: Method of access control that enables a user to log in once and gain access to the resources of multiple software systems without being prompted to log in again.

Subject A principal in the context of a security domain. SAML assertions make declarations about subjects.

System Entity: An active element of a computer/network system. For example, an automated process or set of processes, a subsystem, a person or group of persons that incorporates a distinct set of functionality.

User-Agent: The end user's Web browser which implements HTTP/1.1

Wireless Markup Language: is a markup language intended for devices that implement the Wireless Application Protocol (WAP) specification, such as mobile phones, and preceded the use of other markup languages now used with WAP, such as XHTML and even standard HTML.

WSDL: XML-based language that provides a model for describing Web services.

XRDS: XML format for discovery of metadata about a resource, in particular discovery of services associated with the resource, a process known as service discovery

XRI: Scheme and resolution protocol for abstract identifiers compatible with Uniform Resource Identifiers and Internationalized Resource Identifiers.