GENERIC FRAMEWORK FOR FEDERATED SECURITY ENABLED SOFTWARE SYSTEMS

by

Ashok Shah

B.Sc (Chemistry) South Gujarat University, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE
in the School
of
Interactive Arts and Technology

SIMON FRASER UNIVERSITY
Fall 2006

APPROVAL

Name: Ashok Shah

Degree: Master of Science

Title of thesis: Generic Framework for Federated Security Enabled Software Systems

Examining Committee:

Dr. Chris Shaw

Dr. Marek Hatala, Senior Supervisor

Dr. Dragan Gasevic, Supervisor

Dr. Mohamed Hefeeda, External Examiner,
School of Computing Science,
Simon Fraser University

Date Approved: September 21, 2006

Abstract

There is a remarkable growth in the number of organizations providing online services, some of which are free and some are not. Federated security is a concept that allows different organizations to share access to services based on their mutual trust and user's security credentials. Different federated security solutions (FSS) come with their specifications and guidelines for interacting software components in the software system. The boundary between 'federation' and the software component is often murky, and some FSS guidelines lack stringent requirements for implementing the federation concept. In this thesis we analyze the concept of federation, and how different software components interact with each other with respect to the federation. Based on the analysis we develop a generic framework that can be used to connect any existing FSS. The generic framework provides the features that are superset of features provided by any existing FSS.

Keywords:

federated security, federation, framework, inter-operable federation, security, Shibboleth

This thesis is dedicated to my parents for all their love, encouragement, and support.

Acknowledgments

My first and foremost gratitude goes to my senior supervisor, Dr. Marek Hatala for his guidance through out the thesis. I am grateful for all the research opportunities and the endless discussion for my research.

I would like to thank my thesis committee for their guidance and feedbacks. I thank Dr. Dragan Gasevic for the numerous discussions in search of solutions to different problem. I would like to thank Dr. Mohamed Hefeeda for his critical feedback on my thesis.

My sincere appreciation goes to Dr. Griff Richards for all the new ideas and different research visions. I thank Ty Mey Eap for his insights in different aspects of my thesis.

I would like to thank all my colleagues working on eduSource, LORNET, and LionShare research projects. I would like to thank my friends Amit, Shital, Dhaval, Davis, Jurika, Shilpi, Jeff, Nima, Baljeet, Kandy, and Deval for their support and occasional adventures through out my thesis.

Contents

List of Tables

List of Figures

List of Programs

Chapter 1

Introduction

A federated security solution (FSS) provides a scalable security solution for organizations to share their resources with other organizations. Traditional one-on-one security solutions do not scale when large number of organization are involved. It requires each organization to create seperate account for each user in each organization. A FSS provides a scalable security solution by providing mechanisms to trust and exchange end-user security information between organizations. A FSS is similar to a passport system. A Canadian passport is recognized by every country in the world. If a Canadian citizen wants to go to the USA, the USA recognizes the Canadian passport and respects its value. Allowing the Canadian citizen to enter the USA border without any prior visa. In this example, the whole world could be considered as a 'federation'. A Canadian passport is the credential, and Canada is an identity provider that issues a Canadian passport to a Canadian citizen. The USA is a service provider where a Canadian citizen is seeking entry. A security policy could be considered as rules that govern the access of service. For example, USA has a security policy that allow citizens of Canada to enter the USA without acquiring a visitors visa. USA 'trusts' the government of Canada for issuing the passport. The passport holders decide themselves which country they want to federate their identity, that is to which country they want to travel.

In a software system anyone who offers a service is called a *service provider*. The *service provider* may want to allow access to specific end-users only, based on the end-user's credentials. An entity that issues end user's security credential or assertion is called an *identity provider* . The end user's security credentials are the end-user's properties inside

an organization: For example, John is a graduate student. This statement describes two attribute values. The attribute name 'status' has value 'graduate student', and the attribute name 'student name' has value 'John' in this example. Security credentials are security statements issued by some entity. A group of organizations that wants to participate in federated security have to have a common agreement for mutual trust.

The FSSs define sets of specifications and standards that are used to develop a complete FSS framework. This framework provides user management and access control solutions to organizations. The framework includes the message interaction sequence between different software entities and corresponding security requirements. For example, a FSS typically describes how *service provider* interacts with *identity provider*. A complete FSS consists of core federation components, and the supporting infrastructure components. A core federation component would provide the information about the members of the federation only. Whereas, the supporting infrastructure components use this information and allow users access, based on this information. Both core federation and infrastructural components work hand in hand to achieve federated security. Keeping a clear boundary between the core federation and other components allows easier implementation efforts and chances of interoperability with other federations. However, the message interaction sequences defined by FSSs consider the whole software system as one, and do not separate the interaction between the core federation components and the supporting infrastructural components.

FSSs depend on languages that allow communication of security credentials between different software entities. Assertions are a type of security credentials. There are two different types of assertions: authentication assertion and attribute assertion . Authentication assertions have a unique handle that could be used to obtain the attribute assertion. An authentication assertion could be considered as a proof that the user has authenticated with the *identity provider* and has other security information to obtain an actual attribute assertion if needed. An attribute assertion is a list of attribute names and values that particular end user holds within an organization. For example, if John Doe is a manager at BusinessXYZ, "employee status" could be thought of as the attribute name and "manager" is the attribute value.

eXtensible Access Control Markup Language (XACML) [45], and Security Assertion Markup Language (SAML) [17] are two languages that allow the transfer of security credentials between different software entities. XACML[45] is a flexible access control markup language for access control policies and a request/response language for FSSs. XACML

also dictates the data flow between different software entities in a system, which take up different roles in the data flow. Policy decision point (PDP) is an entity in the XACML data flow that makes the authorization decision of whether the access to requested resource should be granted. Policy enforcement point (PEP) is a first point of intercept for any incoming request in XACML data flow. Context handler is an entity in XACML data flow that converts decision requests from native request format to XACML canonical form, and vice versa.

SAML [17] is a markup language for expressing and exchanging security credentials only. It defines the message format for security credentials, queries, and responses. SAML assertions are the security credentials that are used for decision-making services. The entity that enforces access authorization, for example PEP, at the *service provider* uses the SAML assertions to take authorization decisions. SAML varies from XACML in that SAML allows for secure transfer of credentials between two entities over a network only. XACML provides a framework for making access control decisions. FSSs use SAML when exchanging security credentials between two software entities over a network.

Trust and Federation are the foundation of FSSs. They represent the type of relationships that exist between any two organizations in a FSS. Trust allows two organization to recognize security credentials issued by other organizations that are trusted. Federation allows taking an authorization decision. Trust is required between two organizations before users federate their credentials. FSSs consist of different software entities that are assigned some responsibilities in the software system. For example, *identity providers* are responsible for authenticating users and issuing credentials, and *service provider* provides some kind of service or resource to the user. *Federation member* is a term used for organizations that are part of a federation.

In this thesis we want to understand the requirement in FSS from two perspectives. First, we want to have a clear understanding of different message interaction sequences between the core federation components and the supporting infrastructural components. Second, we want to define a common set of functionalities that would enable an existing software application to be part of a federation with minimum efforts. We will use this understanding of a common set of functionalities required to develop a framework that allows an existing software application to connect and share resource with any FSS systems.

1.1 Motivation

Our initial experience with developing a federated security system [31] [32] left a few questions. The implementation was not smooth, compared to other modules of the project. We spent much time on understanding the different components in the whole FSS. It was difficult to track which components in the federation would interact with the end-user application. The time and efforts spent on understanding the FSSs raises several questions. Could this implementation be done in a better way? Could the implementation be made easier? Is it possible to reuse any of the code that we had already developed for this implementation?

One of the major efforts for the FSSs implementation is in handling the PKI credentials and dealing with the message interaction sequences between different software entities in FSSs. However, the implementation for a particular FSS remains the same. One of the main motivations for this thesis was the realization that if the implementation for particular FSS remains the same, a lot of design and code could be reused. To enable an end user application to connect and interact with FSSs, we could reuse code developed in our previous implementation. If we extract the technical functionalities from FSSs and make it generic enough to work with any existing federation, it would be of great help to the developers of the end user application to interact and connect to FSSs. These generic technical functionalities would reduce a lot of programming efforts for the developers who want to interact and connect to FSSs by reusing an existing implementation. The reusable nature of the code also increases the security of the software system because the security requirements are dealt with in the implementation once, and do not need to be taken care of in every implementation.

1.2 Objectives

The initial pilot study of existing FSSs indicated that each FSS supports a similar type of technical functionalities. But the message interaction sequences and syntax between different software entities within an organization in FSSs are complex, and different for different FSSs. We believe that it is possible to develop a middleware layer that would allow for easier implementation for software systems to connect to FSSs. We call the middleware layer a *generic framework for federated security enabled software systems.* Figure 1.1 shows the concept of the generic framework. This generic framework would interact with the FSSs and return the interaction results to the end user application. The end user application

developers do not have to deal with the message interaction sequences in different FSSs. Instead, the generic framework would provide a set of technical functionalities through which the user application could interact and connect to the FSSs. The proposed framework would act as a middleware layer between the software systems and the FSS components. As we see in figure 1.1, the generic framework allows the end user application to connect to two different kinds of FSS. The end-user application interaction with the generic framework is the same for both, even though the message interaction sequences in both FSSs are different, step number 1 and 8, and step number a and h. It should be noted that the thesis provides the research on FSSs only. The thesis doesnot provide any argument for comparing FSSs with traditional security solutions. The list of our objectives to develop a generic framework for software systems that want to interact and connect with FSSs follows:

- Understand existing FSSs
  We would first study existing FSSs to understand different components involved in the whole software system. This understanding should allow us to differentiate between the core federation components and the supporting infrastructure components in FSSs. This study would allow us to understand the dependencies for each software entities in FSSs. We would also study any attempts to achieve a similar goal to develop a generic framework.

- Define a specification with generalized technical functionalities for FSSs
  Once we understand existing FSSs, we would generalize the technical functionalities provided by each federation. End user applications have a common set of features available in each FSSs. The specification would be the union of the generalized technical functionalities needed by each FSSs.

- Propose a generic framework
  We then propose a generic framework for software systems to enable them to communicate in FSSs based on this specification. The framework is generic in that the end user application could connect to any kind of federation using the proposed framework. Ideally, only the implementation of particular components of the framework would differ for different federated security providers. The generic framework would make the implementation efforts easy for end user application developers. The implementation of the proposed framework for one type of FSS could be reused by other implementers, hence allowing code reusability.

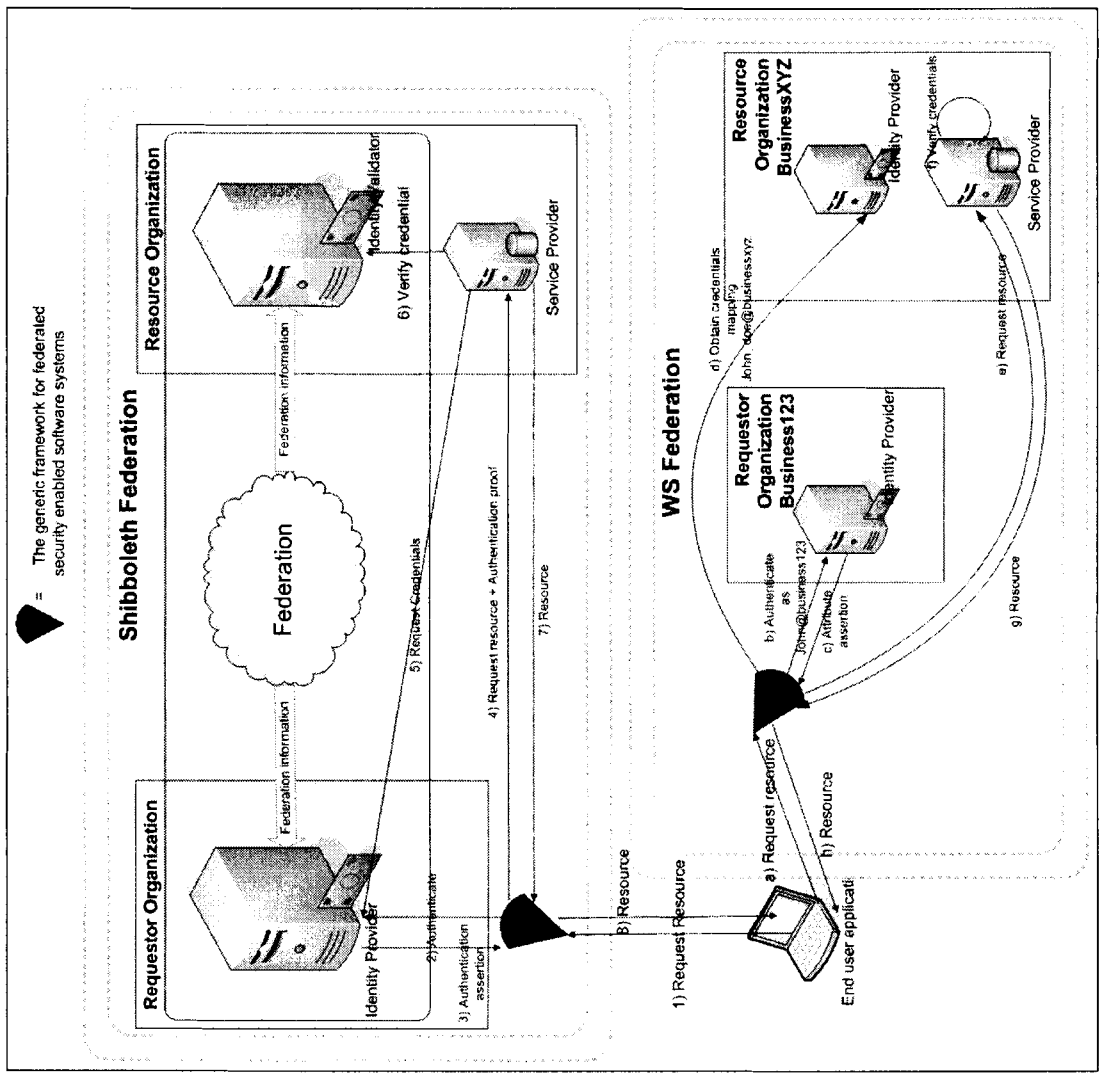

Figure 1.1: Communication flows with two federations using a generic framework component

Chapter 2

Background

This section provides a detailed review of currently existing FSSs. These solutions can be divided into two parts: well-known federated security solutions that are developed by and used in software industry, and academic federated security solutions. Typically, well-known expert entities in the industry are involved in developing standards and implementations for the security solutions provided by well-known federated security. The academic federated security solutions are developed by researchers in the academic world. Academic federated security solutions are typically based on a well-known federated security solution, solving some specific problem in existing well-known FSS. Studying each would allow us to identify different components and understand the message interaction sequences between the software entities in FSSs.

2.1 Well-known federated security solutions

This section reviews existing well-known federated security solutions, that is, set of specifications/implementations that are recognized and used by commercial as well as non-commercial organizations. The FSSs in this section are further divided into two categories: centralized federated security solution (CFSS) , and distributed federated security solutions (DFSS) . In a CFSS, end-user do not have any control over which organizations are part of the federation. A centralized entity is responsible for managing federation member information and serving it to other infrastructural components. In a DFSS users could federate their identity to different providers over the network, and there is no centralized entity. Each end-user application is responsible for managing its own federation members list.

2.1.1 Centralized federated security solution

This section describes major CFSSs in detail. The decision to include organizations in federation is made by some centralized entity. Users can not explicitly federate their identity to different *identity providers* and *service providers*. A centralized federation management manages the list of organizations that are part of the federation. This list is also referred to as a federation members list. Typically, when a new organization wants to become a part of a federation for CFSS type of federation, they have to communicate with the federation management outside the scope of the software system and conclude with an agreement. The federation management, after concluding with an agreement, includes the new organization in the federation. One well known FSS is: Shibboleth Federation.

Shibboleth Federation

Shibboleth [5] is a project of Internet2/MACE, developing architectures, policy structures and open source implementation to support inter-institutional sharing of web resources. Shibboleth uses a model called the Trust Federation model, which means that the members that are trusted are automatically part of the end-user's federation. End-users do not have the ability to directly federate or defederate their identity to other organizations.

Shibboleth has a complex message interaction sequence to achieve the federated security. Here we will mainly describe the parts of Shibboleth Federation that are relevant for the purpose of achieving FSS. There are three main components involved in Shibboleth Federation: origin site, target site or *service provider*, and *identity provider*. The portal from where the request originates is called the origin site. The portal that holds the resource that is being requested is called the target site. The entity that is responsible for issuing security credentials is called the *identity provider*. Home organization is defined as the organization that holds the credential for the end-user/requester.

Typically the origin site and the target site are from different organizations. To allow an end-user access to resources at the target site, using federated security, the origin site organization and the target site organization should have an agreement with federation management beforehand, so that both organizations can recognize and respect each other's identity and credentials.

Shibboleth provides a federation metadata file that holds the list of information about *identity providers* and *service providers* in the federation. How the metadata file is made

available to the federation members is not stringent in Shibboleth federation documents. The information about a *service provider* and a *identity provider* includes technical information, for example the digital certificate of the organization, as well as non-technical information, such as the name of the organization. The certificate included with the *service provider* or *identity provider* information allows other members to check the integrity of the security credentials, and confirm that the credentials were not compromised while transferring between different software entities over the network.

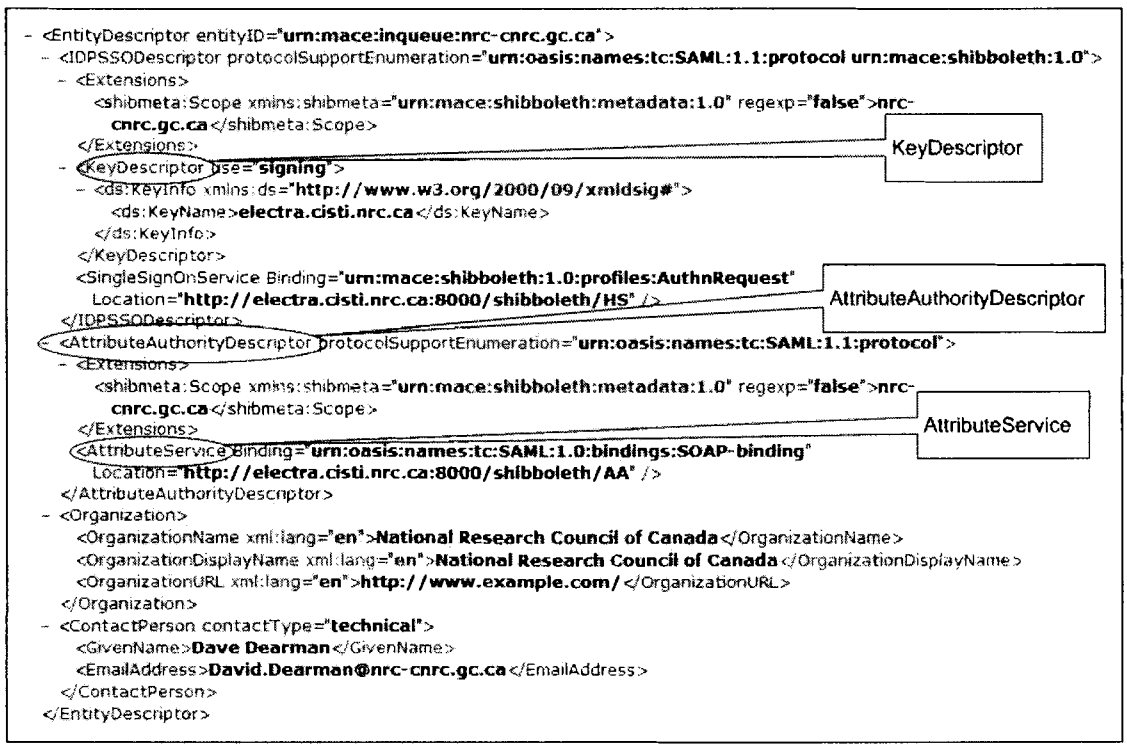

Figure 2.1: Shibboleth metadata example

Figure 2.1 shows a snippet of an entry for an *identity provider* information in a Shibboleth Federation metadata. The *KeyDescriptor* tag holds the certificate information of the *identity provider*. The *AttributeAuthorityDescriptor* holds the information about the entity that issues credentials. The *AttributeService* is a part of *AttributeAuthorityDescriptor* that holds the end point information where a *service provider* could query the *identity provider* to obtain end-user's security credentials. A Shibboleth Federation metadata also contains the information about the *service provider*. The format is similar to the code shown in figure 2.1.

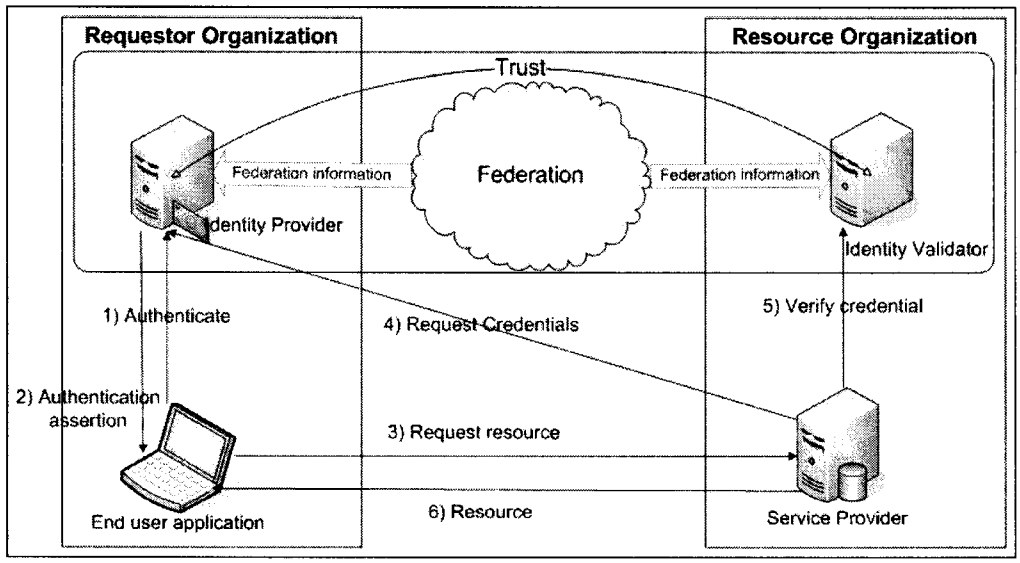

Figure 2.2: Shibboleth Federation

Figure 2.2 outlines the higher level message interaction sequence between different software components in Shibboleth. The figure separates the identity validator and the *service provider* for a clear understanding of the processes involved. In a real Shibboleth Federation, both *service provider* and identity validator reside on the same machine. As shown in figure 2.2 the end-user application authenticates with the *identity provider* and obtains authentication assertion, which is a unique identifier, as a proof of authentication. The end-user application includes this unique identifier with the request for the resource. The *service provider* retrieves the credentials from the end-user's *identity provider*. The *service provider* then validates the credentials before authorizing the request for resource.

When the end-user tries to access a resource from an organization other than the home organization, he/she is asked to authenticate to the home organization, if not already authenticated. Once authenticated with *identity provider*, Shibboleth Federation uses a complex message interaction sequence between different software entities to allow the target site to acquire the end-user's credentials from his/her *identity provider*. The details of the message interaction sequence are not relevant to this thesis. Software entities, that is *service providers* and *identity providers*, in the Shibboleth Federation should consult the federation metadata before issuing the credentials and verifying the credentials. When a *service provider* requests the end-user's credentials, the *identity provider* consults the federation metadata to check if the *service provider* is in the federation. The target site (*service*

provider) consults the federation metadata to check if the *identity provider* that issued the authentication assertion is in the federation. The acquired credentials allow the *service provider* to make an authorization decision about the requested resource. If the *identity provider* that issues the credentials is not a part of the federation, the request is denied without any further processing. Additionally, the *identity provider* signs the issued credentials so that the target site can verify that the credentials are not compromised while being transferred between different software entities over the network.

Studying different Shibboleth Federation implementations [6], [1], [3], [2] shows that the actual implementation of the federation metadata file distribution varies from federation to federation. Shibboleth does not provide stringent requirements for federation implementers for distributing the federation metadata file. From figure 2.2 we can clearly see that the software entities are divided into two categories, one which is core federation specific, that is, it provides the federation metadata file, and others that use the federation metadata file.

The user application that wants to interact and connect to the Shibboleth Federation should deal with different libraries for different technologies, for example, the login mechanism for the *identity provider*. The end-user application should also be able to handle an attribute assertion and an authentication assertion from the *identity provider* and be able to present it to the *service provider* at the correct time of the message interaction sequence, when trying to access a resource at the target site.

To summarize, Shibboleth has following characteristics.

- Shibboleth uses the notion of trust federation.
- Shibboleth is a centralized FSS, in a sense that each organization in the federation has the same set of trusted entities.
- Shibboleth Federations have a federation metadata file that serves as a core federation component.
- The federation implementation varies from one organization to another.
- Shibboleth does not have a trust brokerage system.
- An end-user application should be aware of different technologies involved in the whole federation, and should be able to communicate with other software entities using those technologies.

2.1.2 Distributed federated security solution

This section describes in detail existing DFSSs. End-users can explicitly federate and defederate their identity to different *identity providers* and *service providers*. Different *identity providers* and *service providers* can interact with each other to establish trust. There is no centralized entity to manage trust and federation for end-users, as CFSS. End-users in DFSS create and manage their own federation. DFSS introduces the concept of federating the security credentials to other organization. As a result, each organization involved in this interaction has the security credential mapping of the end-user's security credentials. The security credential mapping could also be considered as a mapping of the end-user's security credentials at each organization involved in this transaction. CFSS only allows the software entities to query the federation members for information from the federation metadata. The federation metadata is managed outside the scope of the software system. DFSS introduces technical functionalities that also allow the software entities in the software system to update the federation members' list. There are two DFSS: WS-Federation and Liberty Alliance Federation.

WS-Federation

WS-Federation [10] is a specification that defines mechanisms to allow different organizations to establish a FSS by allowing trust and federation between organizations. WS-Federation is based on other Web service standards, like WS-Security[46], WS-Policy[12], WS-PolicyAttachment[11], WS-Trust[9] etc. WS-Federation provides a distributed approach, where different *identity providers* in different organizations can interact with each other to establish trust. To achieve a federated security, WS-Federation provides a two layer approach. First is the trust between different *identity providers* and *service providers*. Second is the end-user's ability to federate their identity to other organizations. End-users cannot federate their identity to an organization unless it is trusted. Once the trust exists between a *identity provider* and a *service provider* of different organizations, end-users from one organization could federate their identity to the other trusted organization. If trust between the organizations does not exist already, WS-Federation provides different trust brokering methods that allow establishing trust between organizations. For example, the transitive trust model is transitive in the sense that if *identity provider* A trusts *identity provider* B, and *identity provider* B trusts *identity provider* C, then trust between *identity provider* A

and *identity provider* C can be brokered. One of the substantial features of WS-Federation is the user's ability to have different credentials at different organizations in federation. For example, let's say user John Doe is a MBA student at Simon Fraser University, and an Associate Professor at University of British Colombia. Both Simon Fraser University and the University of British Colombia are part of a federation, let's say the Canadian University Federation. If a resource at some organization in the Canadian University Federation has an access rights policy that allows access to only associate professors, WS-Federation would allow gathering multiple credentials from different organizations in the federation to satisfy the access rights policy for the requested resource, allowing John Doe access to the requested resource.

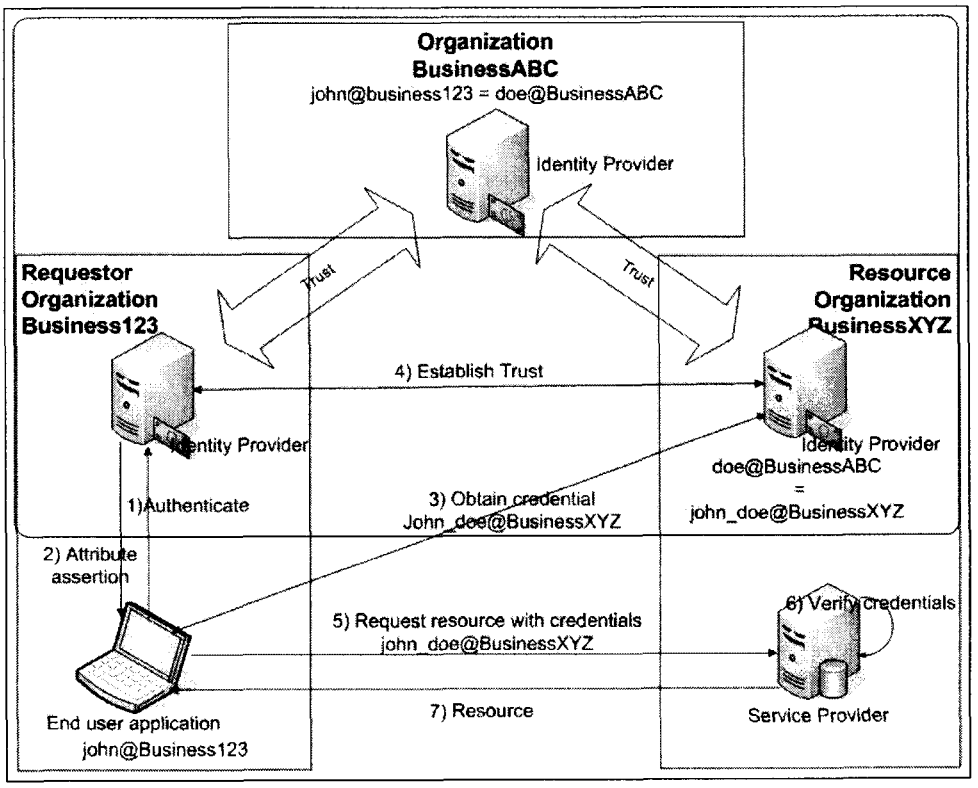

Figure 2.3: WS-Federation

WS-Federation allows different kinds of message interaction sequences between different software entities. An end-user application could either obtain attribute assertion directly from the *identity provider* and present them to the *service provider* or the end-user could present an authentication assertion to the *service provider*, when requesting a resource. The

service provider uses this authentication assertion to fetch the end-user's attribute assertion from the *identity provider*. Depending on what the end-user application provides, the *service provider* either fetches the attribute assertion from the *identity provider* and verifies them, or just verifies the attribute assertion.

Figure 2.3 shows a message interaction sequence between different software entities in WS-Federation where the end-user first authenticates with the *identity provider*, and then wants to request a resource from the *service provider*. There are different message interaction sequences possible in WS-Federation, but the federation functionality remains the same. In figure 2.3 there are three organizations involved: organization Business123, BusinessABC, and BusinessXYZ. A direct trust exists only between Business123 and BusinessABC, and BusinessABC and BusinessXYZ. Business123 and BusinessXYZ do not have a direct trust relation. The end-user first authenticates with its own *identity provider*, that is Business123, and obtains attribute assertion. For simplicity, lets say the attribute assertion is $john@Business123.com$. The real attribute assertion would be more complex. The end-user application then provides that information to the *identity provider* at BusinessXYZ requesting security credential mapping. Obtaining a security credential mapping is equivalent to federating the identity across another organization. Because BusinessXYZ does not have a direct trust relationship with Business123, BusinessXYZ initializes the process to establish trust with Business123, if possible. Because Business123.com and BusinessXYZ are connected through a transitive trust model, BusinessXYZ consults with BusinessABC, and brokers the trust for Business123 and derives the security credentials mapping, which is $john@Business123$, equivalent to $john\_doe@BusinessXYZ$. BusinessXYZ returns this security credentials mapping to the end-user application. The end-user application then includes this security credential, $john\_doe@BusinessXYZ$, along with the request to the resource to BusinessXYZ's *service provider*.

As we see in figure 2.3, there is no direct notion of trust between Business123 and BusinessXYZ. Instead it is a trust brokering architecture between a set of software entities that allow trust brokering if trust does not exist already. The WS-Federation does not have any equivalent to Shibboleth's federation metadata. Instead, each end-user application maintains its own list of trusted entities. It is worth noting that only *identity providers* are responsible for handling end-user's credentials and their corresponding mapping, as shown in figure 2.3 in WS-Federation. The *service provider* does not handle any security credentials information themselves. The *service provider* is only responsible for verifying

the security credentials. The *service provider* depends on the *identity provider* for being a part of federation and when security credentials mapping is needed.

The end-user application that wants to interact and connect to WS-Federation systems should deal with different libraries for different technologies involved in the WS-Federation. For example, the end-user application should be able to handle different kinds of assertion, and should also be aware of different message interaction sequences that allow end-users to federate the identity to other *identity providers.*

To summarize, WS-Federation has following characteristics:

- WS-Federation uses the notion of trust and federation to establish a FSS.
- WS-Federation is a distributed FSS, in that each end-user has their own federation.
- Only *identity providers* can be a part of federation in WS-Federation.
- WS-Federation has trust brokerage system.
- End-user application should be aware of different technologies involved in the whole federation, and should be able to communicate with other software entities using those technologies.

Liberty Alliance Federation

Liberty Alliance is a consortium of more than 150 companies working together towards developing an open, interoperable standard for federated security. The concept of federation in Liberty Alliance is very similar to the WS-Federation concept of federation. Liberty Alliance has a distributed trust federation system, like WS-Federation. However, the message interaction sequence and syntax between different software entities in Liberty Alliance is different than WS-Federation. In WS-Federation only *identity providers* can handle and interact with each other to broker trust and federate identity. In Liberty Alliance, both a *service provider* and a *identity provider* can interact with each other to broker trust and federate the end-user's identity. Both a *service provider* and a *identity provider* can be part of the federation.

Figure 2.4 shows a conceptual model of a FSS as in Liberty Alliance project. Liberty Alliance has a so-called 'circle of trusts'. End-users can have different profiles like 'work profile' and/or 'home profile', each having one or more *identity providers* as shown in figure

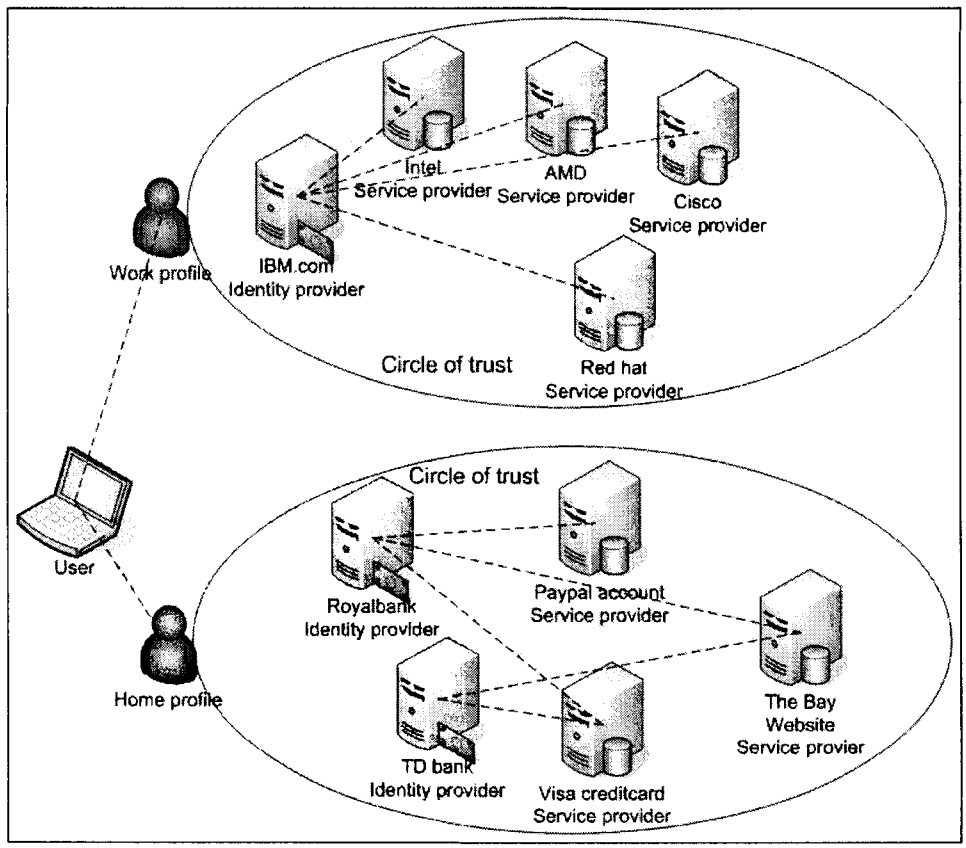

Figure 2.4: Liberty Alliance Architecture

2.4. If a *service provider* is out of the 'circle of trust', a trust brokering mechanism could be used to establish trust. If so, Liberty Metadata and Schema [23] dictate a message interaction sequence between different software entities in Liberty Alliance to add a *service provider* to the end-user's circle of trust. A circle of trust in Liberty Alliance is end-user oriented, in the sense that each end-user would have his/her own circle of trust.

To understand the Liberty Alliance federation in generic terms, that is the terms used with other FSSs discussed in this secion, the circle of trust could be called a federation. Liberty alliance also has two layers, similar to WS-Federation to achieve federated security. First, trust should exist between different *identity providers* and *service providers*. Second, users federate their identity to other *identity providers* and *service providers*, that is, create their circle of trust.

Figure 2.5 shows a message interacton sequence in the Liberty Alliance federation where the end-user authenticates with the Royalbank *identity provider* and wants to access a

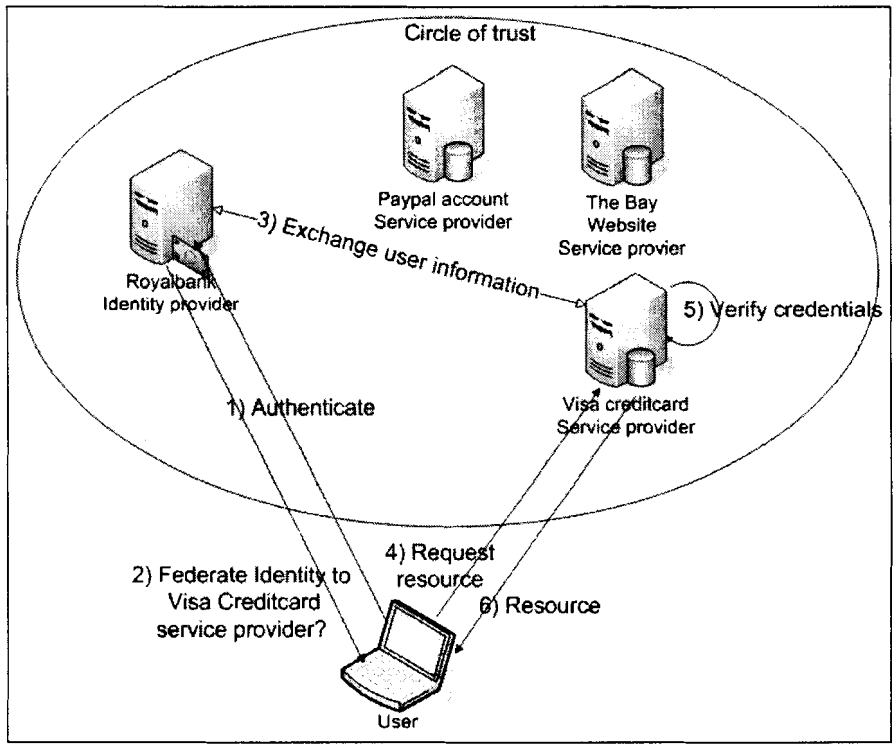

Figure 2.5: Liberty Alliance

resource at the Visa creditcard *service provider*. The Visa creditcard *service provider* and Royalbank *identity provider* have a trust agreement already in place. When the end-user first authenticates with the Royalbank *identity provider* for first time he is asked if he wants to federate his identity to the Visa creditcard *service provider*. If the end-user agrees to federate the identity, Royalbank *identity provider* and Visa creditcard *service provider* exchange messages to federate the identity. As a result, each entity involved in this message interaction would have a mapping of the end-user's security credentials at the other organization. This step happens for each entity the user selects to federate his/her identity. Once the identity of the end-user is mapped on both sides, the end-user can authenticate at either end, and provide the authentication assertion to other organization to access any resource as if they were on the same domain.

The end-user application that wants to interact and connect to Liberty Alliance systems should deal with different libraries for different technologies involved in the WS-Federation. For example, the end-user application should be able to handle different kinds of assertion, and should also be aware of different message interaction sequences that allow users to

federate the identity to other *identity providers.*

To summarize, Liberty Alliance federation has following characteristics:

- Liberty Alliance uses the notion of trust and federation in the software system.
- Liberty Alliance is a distributed FSS in the sense that each end-user has their own federation.
- Both *identity providers* and *service providers* can be a part of a federation in Liberty Alliance.
- Liberty Alliance has a trust brokerage system.
- End-user application should be aware of different technologies involved in the whole federation, and should be able to communicate with other software entities using those technologies.

2.2 Academic federated security solutions

In this section we will describe different FSSs that are developed in the academic community.

Different approaches towards FSSs have been proposed, by authors of [36], [59], [29], [24], [34], [49], and [53]. The approach in [36] is very similar to the Shibboleth Federation approach. Shibboleth uses different markup languages to represent attribute release policies at *identity provider* and the attribute assertions for end-users. Attribute release policies are partly equivalent to the user's ability to federate identity to other organizations. The attribute release policies in Shibboleth allow the end-user to control who could access their resources. The end-user cannot add organizations to the federation, but the end-user could deny releasing attributes to other organizations, which is equivalent to defederating identity in WS-Federation and Liberty Alliance federation systems. [36] proposes a common XACML [45] based message syntax for attribute release policies at *identity provider* and attributes assertions for end-users.

The authors of [59] proposes a federated security model where proxies are needed to act on behalf of the end-user. Present FSSs do not take into consideration the need for proxy servers acting on behalf of users. [59] addresses the user privacy issue in a software domain where proxy servers need to act on behalf of users. [34] presents a FSS which is very similar to the Liberty Alliance federation. [34] puts an additional layer of policies at the *identity*

provider to better preserve user privacy. [24] proposes an authorization model for a federated system that can take care of the end-user's security need in the federation. Typically FSSs are developed for client server architecture. The end-user application acts as a client, and a *service provider* acts as the server providing services. [24] provides a framework where an end-user can share their objects in a secure fashion in a federated security system, that is, an end-user applicatoin can also become a *service provider*.

[43], [8], [48], [52], and [37] presents the security and privacy analysis of different FSSs. [43] argues that the FSS, though it increases the scope of a software system, is just as secure as any other software system without FSS. The security authentication features are collected as an entity, which makes it easier for administrators to manage security, and detect a possible threat. The security measures can be increased, for example, strong authentication encryption, at the few authentication and authorization points in the federated systems. [8] presents a security and privacy analysis of different FSS in a matrix format. These papers help in understanding the concept of federation in a FSS, but do not provide an exact separation between federation and supporting software components.

The authors of [25] has goals very similar to this thesis, but a major difference exists in the approach. [25] describes author's understanding of federation in FSSs, and then provides a study of different existing federated security systems. [25] tries to map software entities from existing FSSs to the author's conceptual model. In our approach, we first try to understand different FSSs, and then propose a conceptual model for a generic framework for FSSs. We describe the resulting differences in the Analysis section.

2.3 Summary

To summarize, we realized that there are three major, well-established FSSs available. FSSs developed in an academic environment typically solve a specific problem in a well-known federated security system. Well-established FSSs gather the understanding of those problems, and consider them in newer versions of standards and implementations. Security and privacy analysis done by researchers in an academic environment are useful in understanding the different components involved in a federation. But neither of them analyze any FSSs to understand the functionality of federation, as in this thesis. The end-user application in existing FSSs should be aware of different technologies used by the software entities in the federation, and be able to communicate using those technologies. Each FSS has two

components: first, the core federation component that provides different software entities in the FSS with the federation information, second the supporting infrastructure components, that is, a *service provider*, an *identity provider* and an end-user application that uses the information provided by the core federation to achieve federated security. [25] has similar goal as this thesis to analyze the concept of federation in FSSs, but the approach is very different, producing different results. In section 5 we provide a comparison and proof of how our solution would outshine this solution.

Chapter 3

The Generic Framework

In this section we will first sketch the requirements for the generic framework for the FSSs discussed in section 2. We will then propose a specification defining a set of technical functionalities that the generic framework should support, followed by the generic framework. The technical functionalities are collected from two perspectives: first, the technical functionalities needed by the *identity provider* and *service provider* in a FSS to communicate with the core federation, and second, the technical functionalities needed by the end-user applications in FSSs. Separating the requirement collection into two perspective allows us to capture a bigger picture of the FSS, and develop the generic framework with the maximum possible generalization.

3.1 Requirements collection

We will now describe the message interaction sequence for FSSs discussed in section 2 in detail. Each FSS discussed in section 2 has a dedicated sub-section for requirement collection that describes the detail message interaction sequence. By describing the detail message interaction sequence, we want to understand the technical functionalities of different software entities in the FSS. In each sub section for requirement collection we cover different types of message interaction sequences to collect all the technical functionality in FSSs. Each requirement is identified by a unique *requirement id.* For a given *requirement id* the corresponding technical functionality remains the same, but the software entities that use the technical functionality may differ.

3.1.1 Shibboleth Federation

Shibboleth's approach toward federation is simple compared to the WS-Federation and Liberty Alliance Federation. The *service provider* and *identity provider* are the only infrastructural components that interact with the core federation component to obtain federation information. Because Shibboleth is CFSS, core federation is only required to serve the information about the federation members. The update of the federation members happens outside the software, with mutual agreement between different organizations.

Figure 3.1 shows the detail message interaction sequence for Shibboleth where the user authenticates with the *identity provider* and then accesses some resource from a *service provider*. When joining the federation the *service provider* specifies the set of attributes that it needs to make any kind of authorization decision. The end-user first authenticates with the *identity provider* and obtains an authentication assertion. The authentication assertion contains a unique identifier as a proof of authentication. A *service provider* when receives the request for a resource, including an authentication assertion, contacts the authenticating *identity provider* to obtain an attribute assertion. The *identity provider* contacts the federation to obtain information about the *service provider* before issuing the attribute assertion. First, it checks if the *service provider* requesting the assertion is in the federation. Second, the *identity provider* fetches the required attribute list for the *service provider* from the federation. Third, the *identity provider* obtains the certificate for the *service provider* to check the integrity of the assertion request from the *service provider*.

When the *service provider* obtains the attribute assertion, it passes the attribute assertion to the identity validator. Please note that to explain the flow of federation clearly, identity validator and *service provider* are shown as separate entities in the diagram. Typically they would reside on the same entity called *service provider*. The identity validator fetches information from the federation to verify the attribute assertion. First, the identity validator checks if the assertion is issued by an *identity provider* that is a member of the federation. Second, the identity validator obtains the *identity provider* certificate and checks the message integrity. If all the checks are completed successfully, the identity validator then runs the attribute assertion against the resource policy to grant access to the resource.

Table 3.1 summarizes the list of federation functionalities used by different software entities in the Shibboleth Federation:

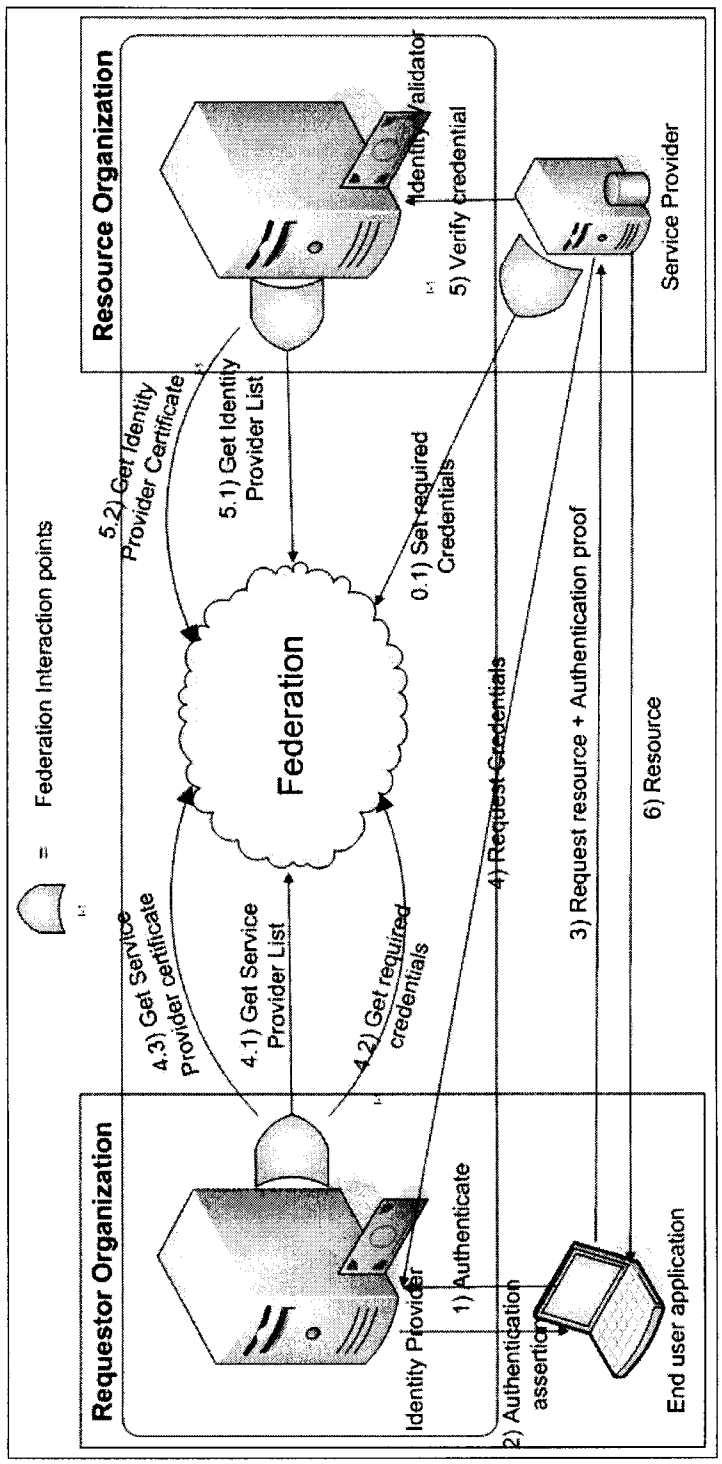

Figure 3.1: Detail Shibboleth Federation

Table 3.1: Shibboleth Federation technical functionalities

Requirement ID:	1
Step :	0.1
Feature :	Set required attributes
Description :	A *service provider* use this technical functionality to specify minimum credentials required to request any resource.
Requirement ID:	2
Step :	1
Feature :	Authenticate
Description :	An end-user application in FSSs uses this technical functionality to login to *identity provider* and obtain an authentication assertion.
Requirement ID:	3
Step :	3
Feature :	Request resource
Description :	End-user application uses this technical functionality to request a resource from *service provider*.
Requirement ID:	4
Step :	4.1
Feature :	Get federation *service provider* list
Description :	An *identity provider* uses this technical functionality to check if the *service provider* requesting the attribute assertion is part of the federation. If the *service provider* is not in the federation members list, then the *identity provider* would not issue attribute assertion to the *service provider*.
Requirement ID:	5
Step :	4.2
Feature :	Get required attributes

Continued on next page...

Table 3.1 – continued

Description :	An *identity provider* uses this technical functionality to get the list of attributes required by the *service provider*. *Identity provider* issues the attribute assertion based on the required attributes set by the *service provider*.
Requirement ID:	6
Step :	4.3
Feature :	Get *service provider* certificate
Description :	An *identity provider* uses this technical functionality to get the certificate of the *service provider* that requests the attribute assertion. The certificate allows the *identity provider* to check the integrity of the attribute assertion request sent by the *service provider*.
Requirement ID:	7
Step :	5.1
Feature :	Get federation *identity provider* list
Description :	An identity validator uses this technical functionality to retrieve the list of *identity providers* that are part of the federation. If the *identity provider* that issued the attribute assertion is not in the federation member list identity validator does not verify the attribute assertion.
Requirement ID:	8
Step :	5.2
Feature :	Get *identity provider* certificate
Description :	An identity validator uses this technical functionality to retrieve the *identity provider* certificate that issued the attribute assertion. *Identity provider* uses this certificate to check the integrity of the attribute assertion and the identity of the entity that issued the credentials.
Requirement ID:	9
Step :	5
Feature :	Verify credentials

Continued on next page...

Table 3.1 – continued

Description :	A *service provider* uses this technical functionality to verify end-user's security credentials.

3.1.2 WS-Federation

WS-Federation's approach toward federation is more complex compared to Shibboleth. WS-Federation provides a different message interaction sequence to achieve federated security. *Identity provider*, *service provider* and the end-user application can interact with the core federation to achieve federated security. The core federation component in WS-Federation should provide the federation member information, as well as technical functionalities to update the federation members list.

Figure 3.2 shows a message interaction sequence in WS-Federation where the end-user authenticates with the *identity provider* and wants to access resource from a *service provider* that is not part of the federation. The scenario described in figure 3.2 is the same as shown in figure 2.3. Figure 3.2 shows a detailed explanation of the exact message interaction sequence between different software entities in the federated system. The end-user first authenticates with Business123's *identity provider* and obtains an attribute assertion. Before sending a request to access the resource at BusinessXYZ, the end-user first federates his identity to BusinessXYZ. The federation manages internally how the identity is federated to BusinessXYZ's *identity provider*. When BusinessXYZ receives the federation request from Business123's user, BusinessXYZ first checks if Business123 is trusted. Because BusinessXYZ does not trust Business123, BusinessXYZ tries to establish trust by mechanisms defined in WS-Federation [10]. The details of the mechanisms used by WS-Federation to broker the trust between BusinessXYZ and Business123 is not relevant to this thesis. As discussed in section 2.1.2, federation uses a transitive brokerage method to establish trust between Business123 and BusinessXYZ. Once the trust is established, BusinessXYZ maps the users attribute assertion, that is, $john@Business123$, to the new attribute assertion that it recognizes, that is, $john\_doe@BusinessXYZ$.

Once the end-user application obtains the new mapped attribute assertion from - $john@Business123$ to $john\_doe@BusinessXYZ$, the end-user could request object from the *service provider* at BusinessXYZ as $john\_doe@BusinessXYZ$. The *service provider* would recognize the provided credentials without any input from the federation because the credentials are provided by *service providers* in its own organization.

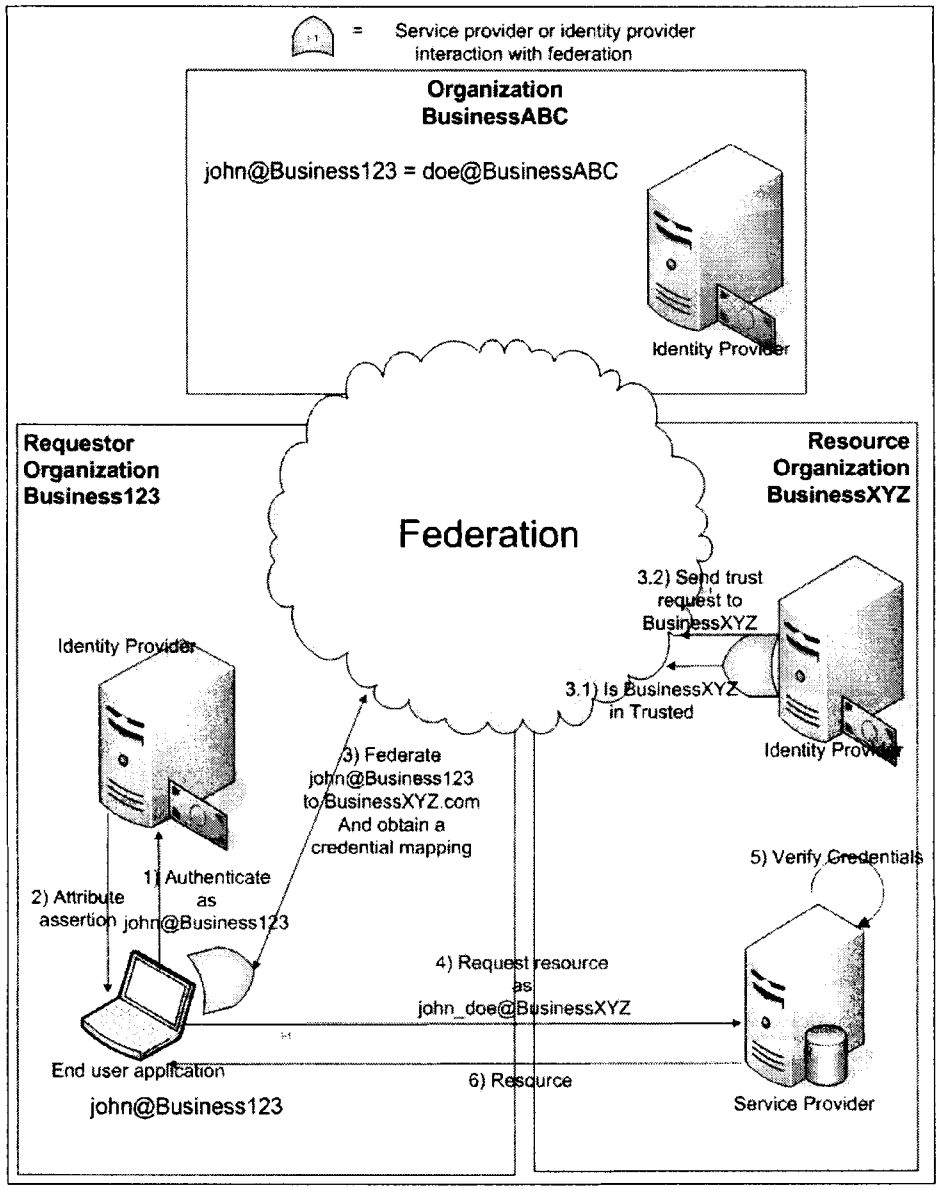

Figure 3.2: Detail WS-Federation

Table 3.2: WS-Federation technical functionalities

Requirement ID:	2
Step :	1
Feature :	Authenticate
Description :	An end-user application in FSSs uses this technical functionality to login to *identity provider* and obtain an authentication assertion.
Requirement ID:	10
Step :	3
Feature :	Federate identity to other organization
Description :	End-user applications use this technical functionality to federate their identity to other *identity providers* .
Requirement ID:	11
Step :	3.1
Feature :	Is particular organization trusted
Description :	An *identity provider* uses this technical functionality to check if a particular organization is trusted. *Identity provider* is required to check the trust with user organization before accepting the identity federation requests.
Requirement ID:	12
Step :	3.2
Feature :	Request trust brokerage
Description :	An *identity provider* uses this technical functionality to establish trust with other organization. The WS-Federation system handles the actual details of how the trust is established. *Identity provider* only needs to submit the request for trust brokerage.
Requirement ID:	9
Step :	5
Feature :	Verify credentials

Continued on next page...

Table 3.2 – continued

Description :	A *service provider* uses this technical functionality to verify end-user's security credentials.
Requirement ID:	3
Step :	4
Feature :	Request a resource
Description :	An end-user application uses this technical functionality to request a secured resource from the *service provider*. Depending on the end-user's preference, the end-user application either includes an authentication assertion or an attribute assertion with the request for a resource.

3.1.3 Liberty Alliance Federation

Liberty Alliance's approach is very similar to WS-Federation's. The message flow sequence in Liberty Alliance and WS-Federation are similar at the conceptual level. The core federation component in Liberty Alliance should also provide information about the federation members, as well as technical functionalities, to update the federation members list.

Figure 3.3 shows a message interaction sequence in Liberty Alliance Federation where the end-user authenticates with the *identity provider*, and wants to access a resource from a *service provider*. The *identity provider* asks the end-user to set preferences for which entities to federate his/her identity when the account is accessed the first time, or each time the *identity provider* adds a new member to its trust list. For example when the end-user authenticates with the Royalbank *identity provider*, Royalbank *identity provider* asks the user if he wants to federate his identity to Visa creditcard *service provider*. Royalbank *identity provider* remembers this preference and federates (or does not) the end-user's identity to the Visa creditcard *service provider* every time the end-user authenticates. If the trust between the entities that are involved in federating identity does not exist, a trust brokerage mechanism is used to establish trust, if possible. The federation handles the details of how the trust is established, if it is possible. The end-user when requests a resource from the Visa creditcard *service provider* using Royalbank *identity provider* security credentials. The Visa creditcard *service provider* can give access to the resource. Because Royalbank *identity provider* federated the end-user's identity to Visa creditcard service, the Visa creditcard service does not need to perform any additional check when it receives a request from that

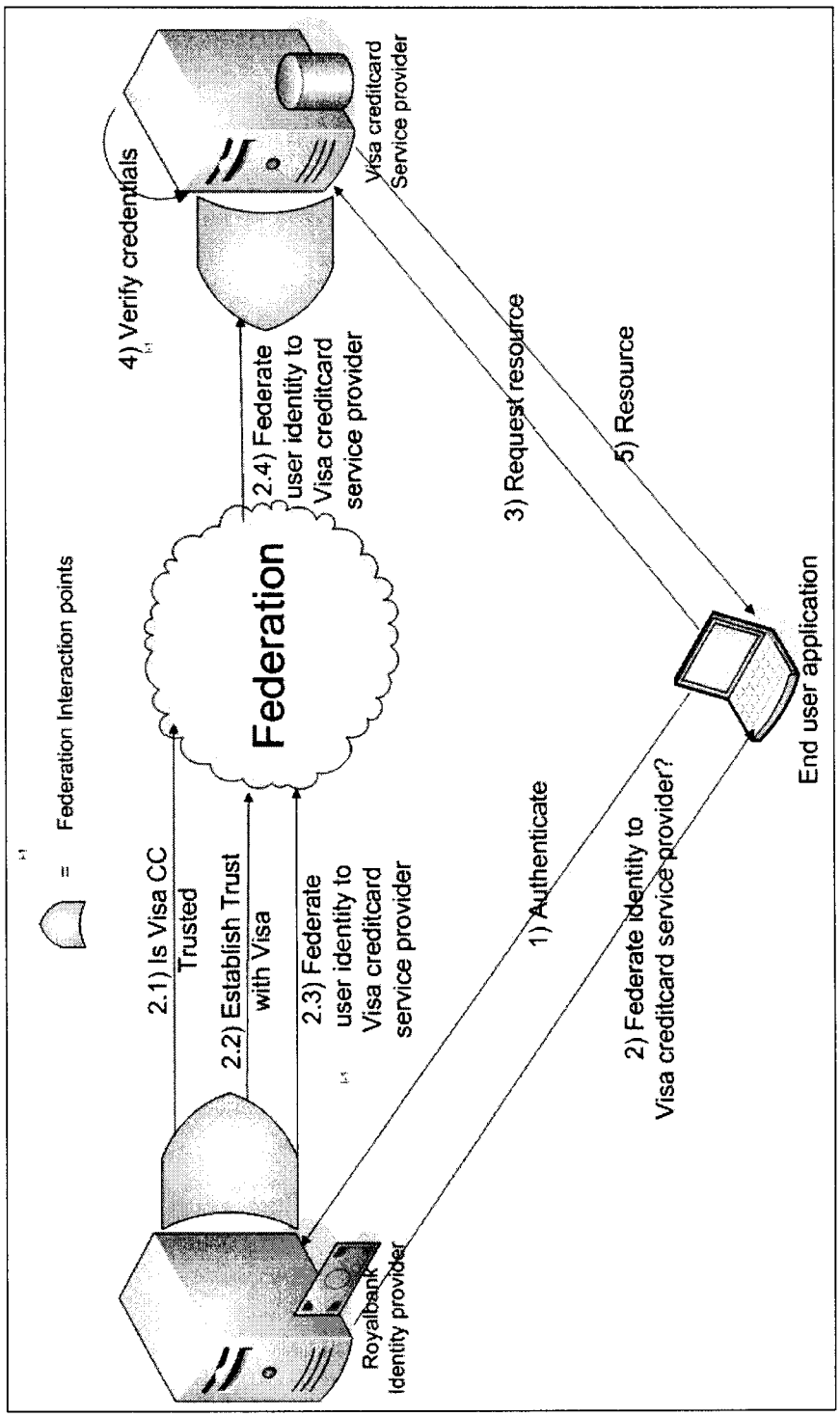

Figure 3.3: Detail Liberty Alliance Federation

user.

To summarize, table 3.3 is the requirement from core federation in Liberty Alliance:

Table 3.3: Liberty Alliance technical functionalities

Requirement ID: Step : Feature : Description :	2 1 Authenticate End-user applications in FSSs use this technical functionality to login to *identity provider* and obtain an authentication assertion.
Requirement ID: Step : Feature : Description :	11 2.1 Is particular organization trusted A *identity provider* uses this technical functionality to check if a particular organization is trusted. *Identity provider* is required to check the trust with the end-user organization before accepting the identity federation requests.
Requirement ID: Step : Feature : Description :	12 2.2 Request trust brokerage A *identity provider* uses this technical functionality to establish trust with other organization. The WS-Federation system would handle the actual details of how the trust is established. *Identity provider* would only need to submit the request for trust brokerage.
Requirement ID: Step : Feature : Description :	10 2.3 Federate identity to other organization An *identity provider* uses this technical functionality to federate user's identity to other organization.
Requirement ID: Step :	9 4

Continued on next page...

Table 3.3 – continued

Feature :	Verify credentials
Description :	A *service provider* uses this technical functionality to verify end-user's security credentials.
Requirement ID:	3
Step :	3
Feature :	Request a resource
Description :	An end-user application uses this technical functionality to request a secured resource from the *service provider*. Depending on the user preference, the end-user application either includes an authentication assertion or an attribute assertion with the request for resource.

3.2 Specification for the generic framework

In section 3.1 we studied different federated systems with the goal of understanding the detail message interaction sequences between different software entities in the federated system. The study allowed us to analyze technical functionalities needed by different software entities in FSS. This section gathers these technical functionalities, and presents a generic specification that could be used to develop a generic framework to enable software systems with federated security. The analysis of the existing FSSs in section 3.1 shows that there are three main software entities in the FSSs that need to interact with the core federation: the *identity provider*, the *service provider*, and the end-user application. The generic specification technical functionalities should be the union of the technical functionalities analyzed for each FSSs in section 3.1. Figure 3.4 shows the UML usecase diagram for the union of the technical functionalities analyzed for each FSSs in section 3.1. The technical functionalities are divided into four categories: technical functionalities needed only by *identity providers*, technical functionalities needed only by *service providers*, technical functionalities needed by both *service providers* and the *identity providers*, and the technical functionalities needed by the client application.

Below is the description of the technical functionalities for the generic specification for the FSS:

- Get *identity provider* information

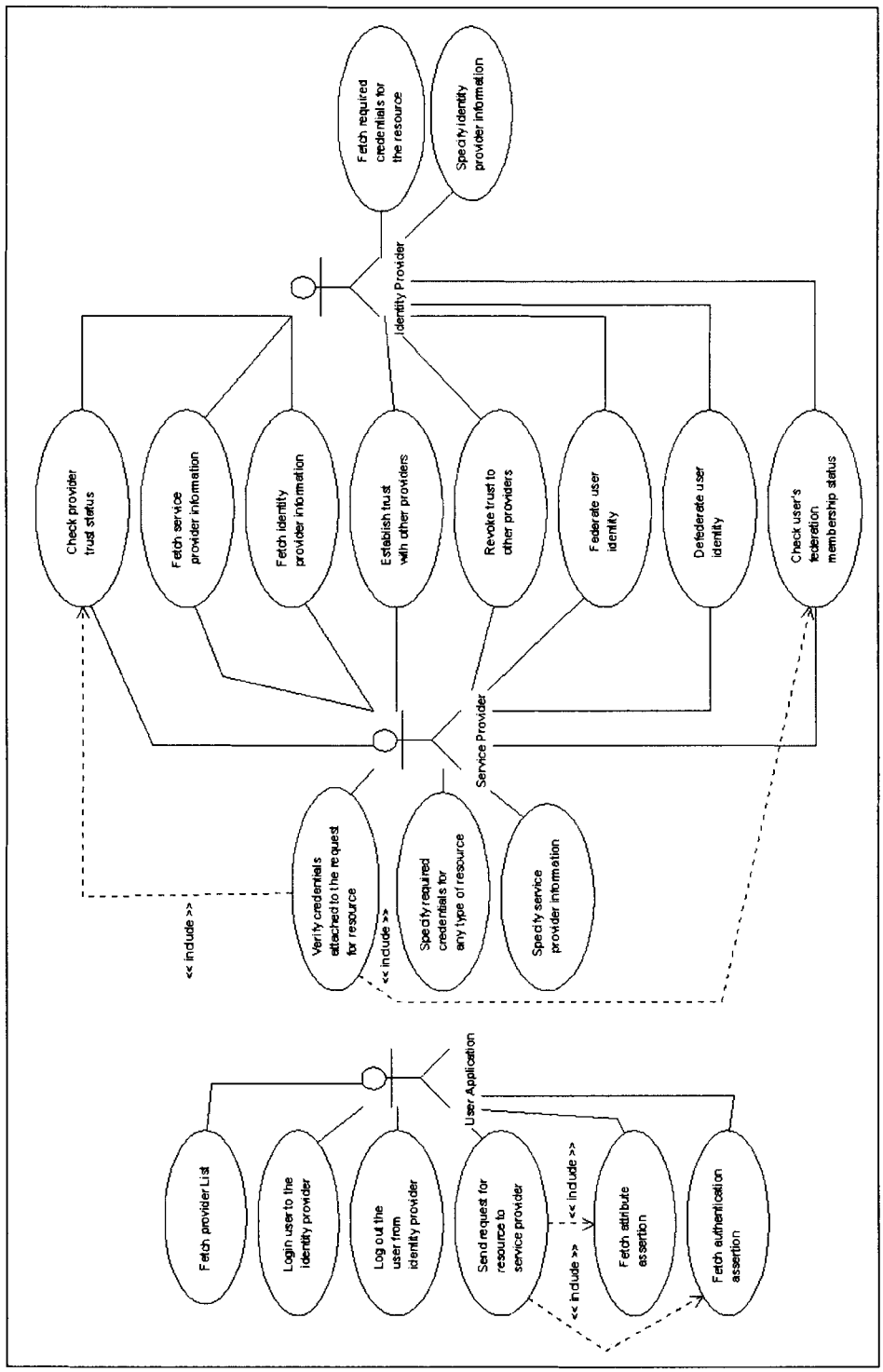

Figure 3.4: Use case diagram for generic federation framework

The infrastructural components use this technical functionality to get the information about an *identity provider* in trust list. This technical functionality generalizes the *requirement id* 8.

- Set *identity provider* information
  Identity provider use this technical functionality to upload its technical information to the federation, for example X509Certificate.

- Get *service provider* information
  The infrastructural components use this technical functionality to fetch the information about other *service providers* in the trust list. This technical functionality generalizes the *requirement id* 6.

- Set *service provider* information
  Service providers use this technical functionality to upload its technical information to the federation, for example X509Certificate.

- Set required credentials
  Service providers use this technical functionality to upload the information about the minimum credentials that they require to process any kind of request for resource. This technical functionality generalizes the *requirement id* 1.

- Get required credentials
  An *identity provider* uses this technical functionality to fetch the information about the minimum credentials required by the *service provider* to process any kind of request for resource. This technical functionality generalizes the *requirement id* 5.

- Is organization a trusted member
  The infrastructural components use this technical functionality to check whether a particular organization is trusted. This technical functionality generalizes the *requirement id* 11.

- Is user a federation member
  The infrastructural components use this technical functionality to check if the user federated his/her identity.

- Federate identity
  The infrastructural components use this technical functionality to federate end-user's

identity to other organizations. This technical functionality generalizes the *requirement id* 10.

- Defederate identity
  The infrastructural components use this technical functionality to defederate the end-user's identity from other organizations. This technical functionality is used by the *sign out* technical functionality described in WS-Federation [10] and Liberty Alliance [4] specifications.

- Establish trust
  The infrastructural components use this technical functionality to establish trust with other organizations. For example, this technical functionality acts as a wrapper for establishing trust using the community trust model or brokerage trust model in Liberty Alliance [4]. This technical functionality generalizes the *requirement id* 12.

- Revoke trust
  The infrastructural components use this technical functionality to revoke trust with any organization in the trust list.

- Login
  An end-user application uses this technical functionality to allow end-users to login to the *identity provider*. The end-user application do not have to deal with the complications of handling the login procedure. For example, if the *identity provider* uses Kerberos ticket mechanism in JAAS to allow the end-user to login, the implementation of this technical functionality in the `ClientFederationHandler` API would take care of the implementation of the same. The end-user application just needs to provide the user name and password to obtain the Kerberos ticket. This technical functionality generalizes the *requirement id* 2.

- Logout
  An end-user application uses this technical functionality to logout the end-user from the *identity provider*, after which the *identity provider* does not issue the end-user's credentials to anyone.

- Send request
  An end-user application uses this technical functionality to allow the end-user to send

a request for a resource to any federation member's *service provider*. This technical functionality helps in simplifying the implementation efforts for end-user application developers. This technical functionality would interact with multiple entities in the federation if required to obtain the resource. For example, if an SFU student wants to access an online book from the UBC library website, this technical functionality interacts with the *service provider* entity at UBC and the *identity provider* entity, if needed to obtain the access for the resource. This technical functionality generalizes the *requirement id* 3.

- Verify credentials
  A *service provider* and an end-user application use this technical functionality to verify the credentials coming from other federation members. This technical functionality simplifies the implementation efforts at the end-user application. This functionality would interact with multiple entities in the federation if required to verify the credentials. For example, when the end-user application gets request for a resource which has only authentication assertion. The generic framework would communicate with the *identity provider* that issued the authentication assertion and obtain a user attribute assertion and verify them. This technical functionality generalizes the *requirement id* 9.

3.3 The generic framework

This section sketches the generic framework based on the generic specification described in section 3.2. The generic framework can be used by the end-user application developers to allow the end-user application to connect and communicate with the FSS. Figure 3.4 highlights that the usecases/technical functionalities for the generic framework can be divided into four categories: technical functionalities needed by *identity providers* only, technical functionalities needed by *service providers* only, technical functionalities needed by both *service providers* and the *identity providers*, and the technical functionalities needed by the end-user application.

3.3.1 Overview

Section 3.2 highlights the three software entities involved in FSSs: *identity providers*, *service providers*, and end-user applications. Section 2 highlights the well-known federated security systems are divided into two categories: CFSS and DFSS. The generic framework is based on these two types, with the insights from the academic FSSs. The generic framework provides abstract classes for each software entity that needs to interact with the core federation. These abstract classes are divided based on the software entity they represent, and the type of federation they belong to. For example, the generic framework provides two abstract classes, `CentralizedFederationIdentityProvider` for *identity provider* in a CFSS, and `DecentralizedFederationIdentityProvider` for *identity provider* in a DFSS. The abstract classes in the generic framework provide implementation of technical functionalities that remain common throughout the type of FSS the class represents.

Description

Figure 3.5 shows the overview of the framework, which is explained in detail in the following figures. Figure 3.5 shows four categories for the generic framework: `ServiceProviderFederationHandler`, `IdentityProviderFederationHandler`, and `ClientFederationHandler`. Each represents an interface for a software entity they represent. For example `ServiceProviderFederationHandler` represents the *service provider* in a FSS. `FederationHandler` represents an interface that holds the technical functionalities that are common to all three entities in the FSS, *service providers*, *identity providers*, and end-user applications.

Figure 3.6 shows the details of the common technical functionalities needed by all three entities in FSS. Figure 3.7 shows the details of the technical functionalities needed by the *service provider*. Figure 3.8 shows the details of the technical functionalities needed by the *identity provider*. Figure 3.9 shows the details of the technical functionalities needed by the end-user application.

The `FederationHandler`, in figure 3.6 provides an interface for two kinds of technical functionalities to communicate with the core federation in a FSSss: query and update. Technical functionalities in the query category allow the software system that implements this abstract class to query the federation and obtain the information about the *service provider* and the *identity provider* that are part of the federation. Technical functionalities

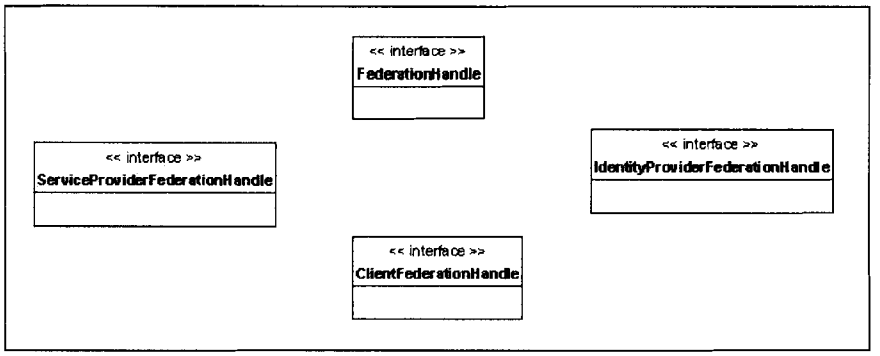

Figure 3.5: The Generic Framework Overview

in the update category allows the implementing software system to modify the federation members list. `CentralizedFederationProviderHandler` and `DecentralizedFederationProviderHandler` are abstract classes that provide implementation for the technical functionality that allows updating the federation information in `FederationHandler` that is generic for a CFSS and a DFSS respectively. `Provider` is a wrapper class that holds the information about the *service provider* or the *identity provider.*

The `ServiceProviderFederationHandler`, in figure 3.7, provides technical functionalities specific to the *service provider.* It allows the implementing software system to set information required to obtain access to its secured resource, and verify the end-user's credentials when the end-user tries to access a secured resource. The abstract class `CentralizedFederationServiceProviderHandler` provides implementation for some technical functionalities that are common to the CFSS. The abstract class `DecentralizedFederationServiceProviderHandler` does not provide any implementation but is present to keep the framework consistent.

The `IdentityProviderFederationHandler`, in figure 3.8, provides technical functionalities specific to the *identity provider.* It allows the implementing software system to get information required to obtain access to its secured resource at the *service provider* in a FSS. `CentralizedFederationIdentityProviderHandler` provides implementation for some technical functionalities that are common to the CFSS. `DecentralizedFederationIdentityProviderHandler` does not provide any implementation but is present to keep the framework consistent.

The `ClientFederationHandler` is an interface, shown in figure 3.9, that provides technical functionalities that are required for the end user application.

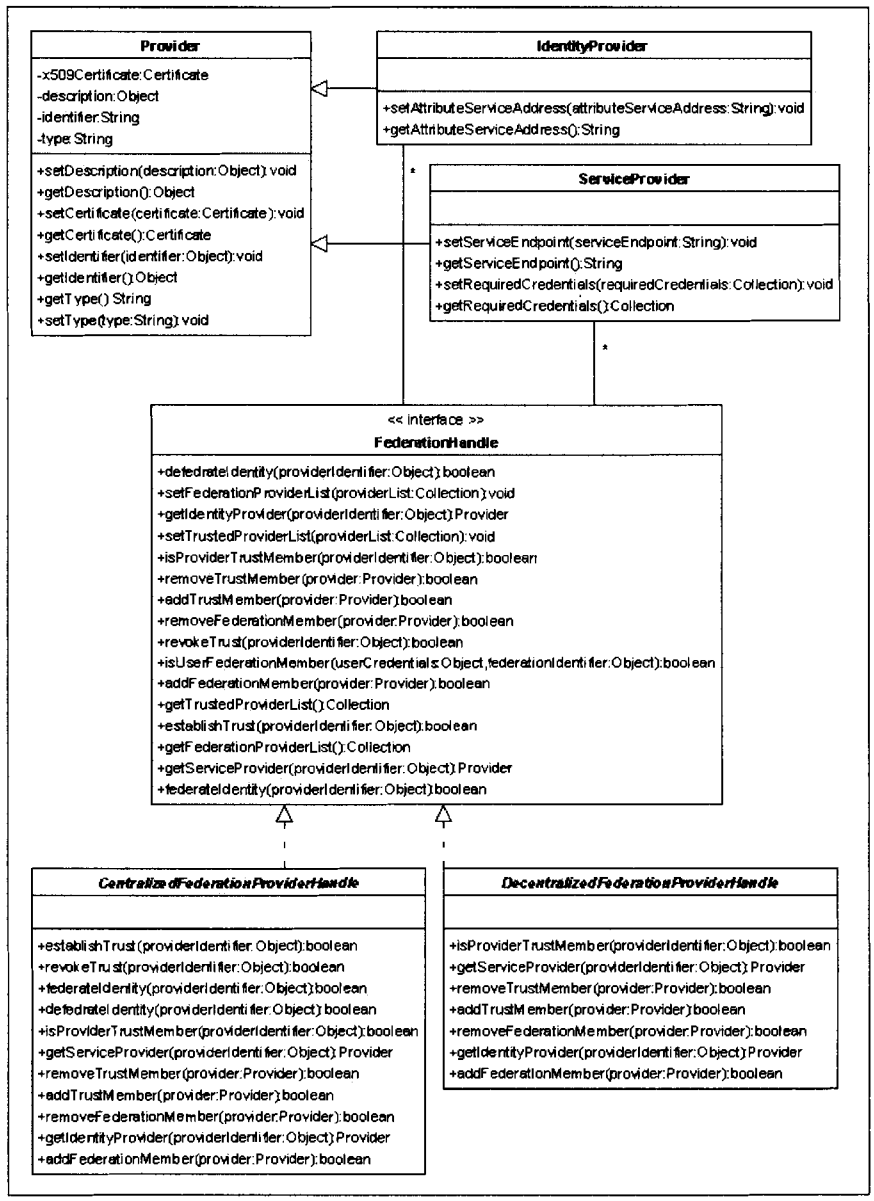

Figure 3.6: The Generic Framework FederationHandler

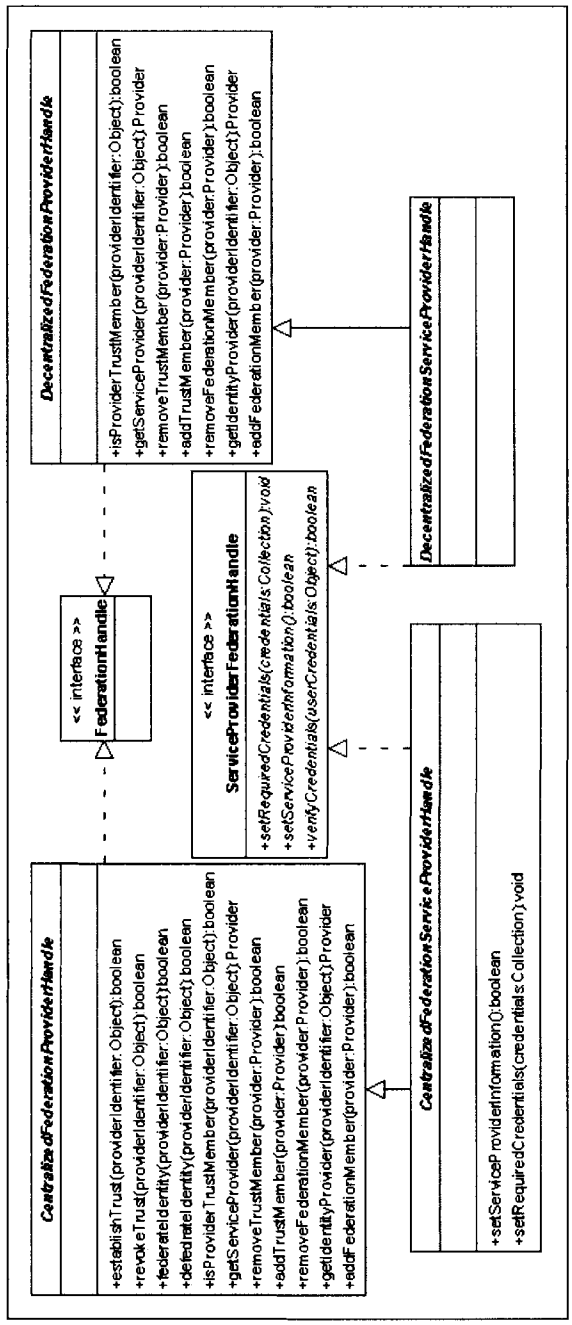

Figure 3.7: The Generic Framework ServiceProviderHandler

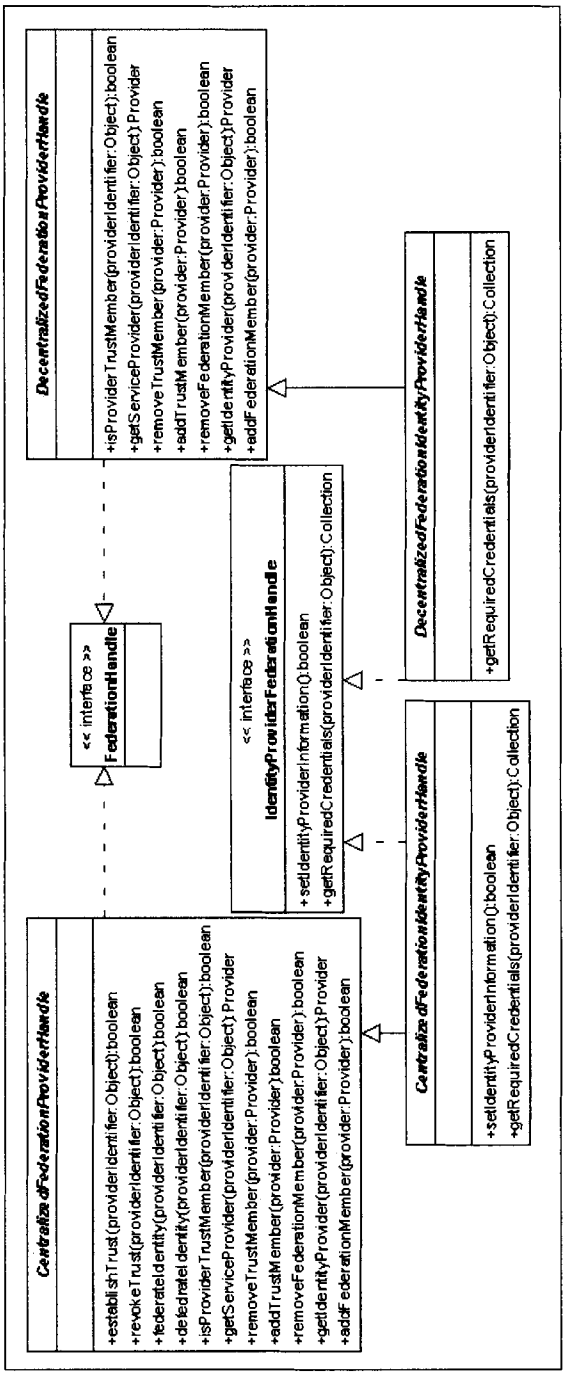

Figure 3.8: The Generic Framework IdentityProviderHandler

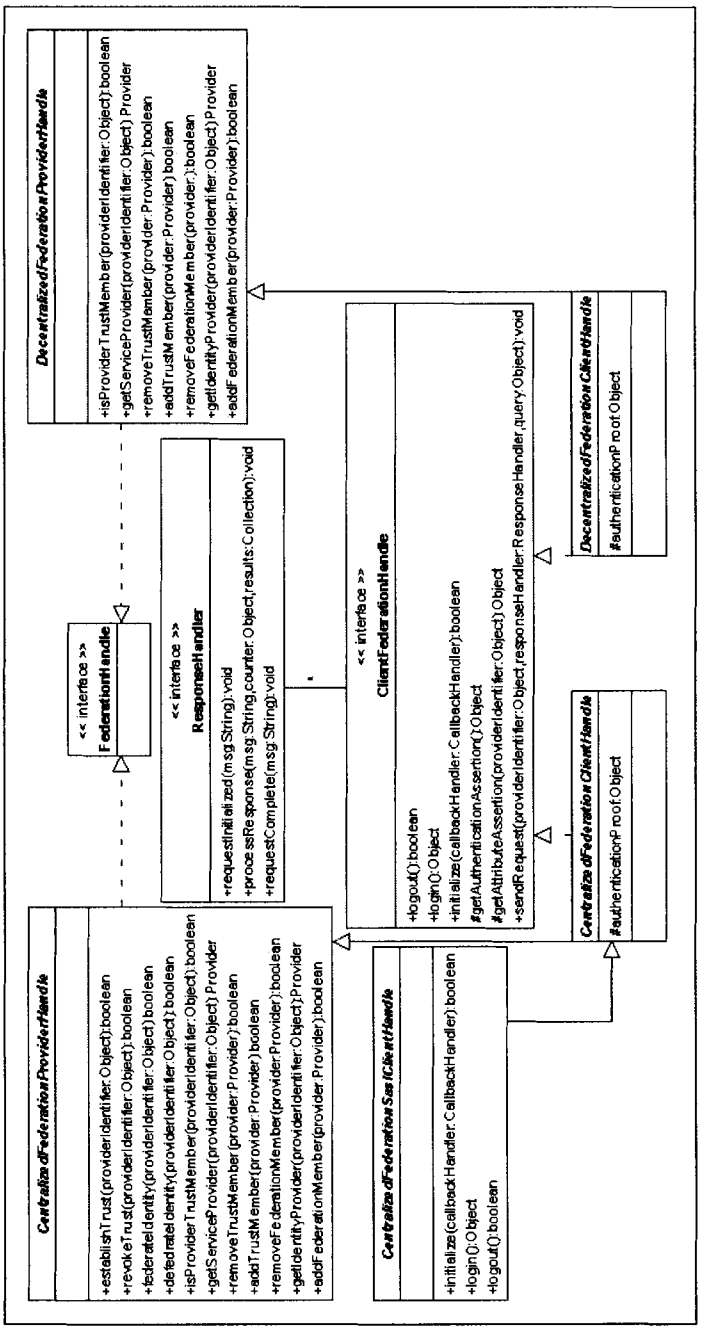

Figure 3.9: The Generic Framework Client's Federation Handler

`CentralizedFederationClientHandler` and `DecentralizedFederationClientHandler` are abstract classes that provide implementation of the federation specific technical functionalities needed by the end-user application. These abstract classes allow the end-user application to worry only about the implementation of the search results. The complex handling of the security credentials and authentication is handled by the implementation in these abstract classes. The abstract class `CentralizedFederationSaslClientHandler` shows that the design of the generic framework is modular. It provides implementation for the authentication mechanism to the SASL type of authentication system at *identity provider*. SASL [44] is a framework for authentication that decouples authentication mechanisms from application protocols. It allows the end-user applications to select the authentication mechanisms that are supported by the server on the run. If an end-user application wants to use any customized type of authentication mechanism, it could extend abstract class `CentralizedFederationClientHandler` and provide the implementation for the technical functionalities that are specific to the authentication mechanism. `ClientFederationHandler` uses a listener class `ResponseHandler` to allow the end-user application to extract the response sent by the *service provider* to the generic framework. The end-user application that needs to connect to FSS should extend appropriate abstract class based on the type of federation they want to connect.

Figure 3.10 shows the overall flow of messages in the framework. The end-user application developers should first initialize the class implementing `ClientAppHandler`. The technical functionality `initialize` allows the `ClientAppHandler` class to fetch the authentication information from the client, for example username and password. The end-user application then fetches the list of *service providers* where the resources are available. The end-user application sends a request for a resource to one of the *service provider* on this list. When the `ClientAppHandler` gets the request for a resource from the user, it first checks if the user is authenticated at the *identity provider*; if not, it authenticates with the *identity provider*. `ClientAppHandler` then obtains the attribute assertion from the *identity provider* and sends the request to the *service provider*. The attribute assertion is not stored at the `ClientAppHandler`. Instead a new attribute assertion is obtained for each request for a resource, enforcing the privacy that is recommended by the FSSs. The *service provider* when receives the request for a resource, first checks if the *identity provider* that issued the attribute assertion is trusted, then it checks if the user has federated his/her identity before verifying the attribute assertion. If all the checks are finished without error, the *service*

provider returns the response to the `ClientAppHandler`. `ClientAppHandler` returns the results to the end-user application through the listener class that is provided during the initial request for the resource.

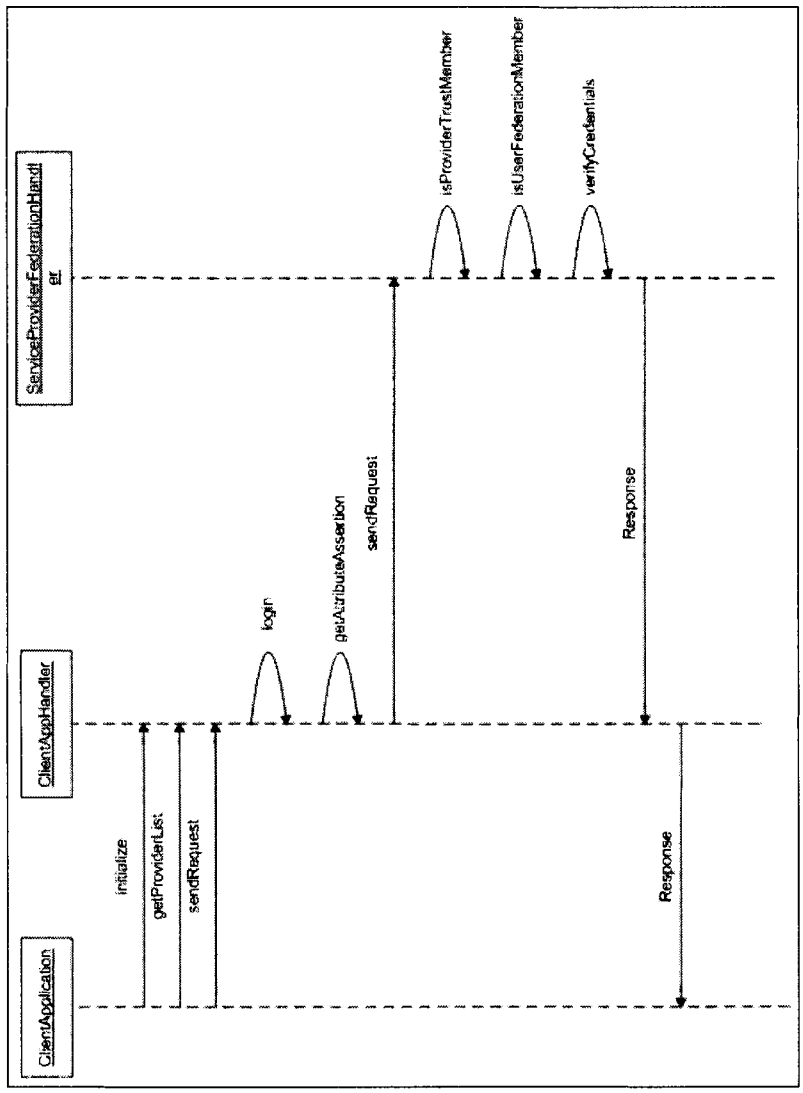

Figure 3.10: The generic framework flow diagram

Chapter 4

Implementation

This section presents the detailed description of the implementation of the generic framework proposed in section 3. The implementation of the generic framework is divided into two categories, first, implementation of the generic framework, which in includes providing a skeleton that would provide the features that are discussed in section 3.1. We than use this generic framework implementation to implement the second part, which is implementing the generic framework for a specific federation. We choose Shibboleth federation for the federation sepcific implementation. Towards the end of this section we present a modularity metric for the generic framework.

4.1 The generic framework implementation

The generic framework acts as a local federation information manager for the end-user applications. This framework allows the end-user application to abstract the functionality required by the FSSs, and acts as a layer between the end-user application and the actual federation. All the communication happens through the generic framework. The implementation consists of interfaces, abstract classes, and wrapper classes. There are three wrapper classes defined in the generic framework: `ServiceProvider`, `IdentityProvider`, and `ResponseHandler`. `ServiceProvider` and `IdentityProvider` hold technical information about *service provider* and *identity provider* respectively. `ResponseHandler` is a wrapper class to handle the response coming from other service providers in the FSS.

The abstract classes provide implementation for the common feature for the type of federation they represent, that is, either CFSS or DFSS. The implementation in the abstract

classes provides ways to use and/or modify the federation members information. The information about the two type of relationships between organizations as described in section 1 are represented as protected variables in `CentralizedFederationProviderHandler` and `DecentralizedFederationProviderHandler`, shown in program 4.1.

```
protected static Collection trustProviderList;
protected static Collection federationProviderList;
```

Program 4.1: Representation of types of relationship between organizations

These variables are protected to allow access to any class extending either of `CentralizedFederationProviderHandler` or `DecentralizedFederationProviderHandler`. These variables are populated during the initialization of the end-user application. For a given end-user application, `trustProviderList` holds the technical information about each organization that is trusted by the end-user application, and `federationProviderList` holds the technical information about each organization that is part of the end-user's federation. Program 4.2 is the code snippet for the implementation to populate `trustProviderList` for Shibboleth type of federation.

The generic framework uses these two variables to retrieve information about the trusted organizations or organizations that are in end-user's federation. For example, the implementation of `getServiceProvider` method in the `CentralizedFederationProviderHandler` abstract class gives the technical information about a *service provider* with given identifier. If the *service provider* with such an identifier is not in the trusted list then it returns `null`.

4.2 Federation specific implementation

This section describes the details of the federation-specific implementations needed for the generic framework. The federation-specific implementation is required for the *requirement ids* 10, 11, 12 as introduced in section 3.1.

For example, the code shown in program 4.6 shows the Shibboleth Federation specific way to check if the user is a federation member.

Because the Shibboleth Federation is a CFSS, Shibboleth does not allow any update features (discussed in section 3.1.1). If implementing a DFSS: WS-Federation or Liberty Alliance Federation, the end-user application developers should provide a federation-specific

Program 4.2: Implementation to populate trusted organizations

```
public Collection getTrustedProviderList() {
    ConfigManager.setHome("conf");
    String home = ConfigManager.getHome();
    String metadataFile = home + "IQ-metadata.xml";
    try {
        if (federationMetadata == null) {
            System.out.println("FederationMetadata
              is null");
            URL metadataUrl = new File(metadataFile).toURL();
            Document metadataDocument = Parser.loadDom(metadataUrl, false);
            if (metadataDocument == null) {
                throw new NullPointerException("Parse error while"
                        + "parsing metadata file:"+metadataFile);
            }
            Element metadata = metadataDocument.getDocumentElement();
            XMLMetadataProvider mdp =
                new XMLMetadataProvider(metadata);
            if (mdp == null) {
                throw new NullPointerException("error creating "
                + "XMLMetadataProvider from parsed XML metadata");
            }
            System.out.println("FederationMetadata all set");
            federationMetadata = mdp;
            EntitiesDescriptor ed =
                 federationMetadata.getRootEntities();
            Iterator it = ed.getEntityDescriptors();
            while (it.hasNext()) {
                EntityDescriptor obj=(EntityDescriptor)it.next();
                String id = obj.getId();
                AttributeAuthorityDescriptor
                    attributeAuthorityDescriptor = obj
                        .getAttributeAuthorityDescriptor(
                        "urn:oasis:names:tc:SAML:1.1:protocol");
                if (attributeAuthorityDescriptor != null) {
                    IdentityProvider pi = new IdentityProvider();
                    pi.setIdentifier(id);
                    pi.setType(FrameworkConstants.IDENTITY_PROVIDER);
                    pi.setDescription(attributeAuthorityDescriptor);
                    providerInformationHashMap.put(id, pi);
                    System.out.println("Found AttributeAuthorityDescriptor");
                }
                SPSSODescriptor spssoDescriptor = obj.getSPSSODescriptor(
                      "urn:oasis:names:tc:SAML:1.1:protocol");
                if (spssoDescriptor != null) {
                    ServiceProvider pi = new ServiceProvider();
                    pi.setIdentifier(id);
                    pi.setType(FrameworkConstants.SERVICE_PROVIDER);
                    pi.setDescription(spssoDescriptor);
                    providerInformationHashMap.put(id, pi);
                    System.out.println("Found SPDescriptor");
                }
            }
        }
    } catch (Exception e) {throw new RuntimeException(e);}
    return providerInformationHashMap.values();
}
```

```
public ServiceProvider getServiceProvider(Object providerIdentifier) {
    if (!isInitialized()) {
        initialize();
    }
    Iterator it = providerList.iterator();
    while (it.hasNext()) {
        ServiceProvider pi = (ServiceProvider) it.next();
        if (pi.getType().equals(FrameworkConstants.SERVICE_PROVIDER)) {
            if (pi.getIdentifier().equals(providerIdentifier)) {
                return pi;
            }
        }
    }
    return null;
}
```

Program 4.3: Implementation to fetch technical information about a ServiceProvider

```
public abstract class CentralizedFederationSaslClientHandle extends
CentralizedFederationClientHandle
```

Program 4.4: Signature of abstract class CentralizedFederationSaslClientHandle

```
public abstract class CentralizedFederationCASClientHandle extends
CentralizedFederationClientHandle
```

Program 4.5: Signature of abstract class CentralizedFederationCASClientHandle

```
public boolean isUserFederationMember(Object userCredentials,
Object federationIdentifier) {
if(userCredentials != null){
if(userCredentials instanceof SAMLAssertion){
SAMLAssertion samlAssertion = (SAMLAssertion) userCredentials;
String issuer = samlAssertion.getIssuer();
IdentityProvider identityProvider = getIdentityProvider(issuer);
if(identityProvider != null){
return true;
        }
      }
    }
return false;
}
```

Program 4.6: Shibboleth implementation to check if the user is a federation member

implementation for methods that update the federation information: `federateIdentity`, `defederateIdentity`, `establishTrust`, and `revokeTrust`. The developers should also provide implementation for other federation specific implementation that are discussed above.

4.3 The generic framework modularity

This section presents the modular nature of the generic framework. Different implemented classes are divided into different levels each of which provides implementation for a features that are common to the level they fall into. Different classes implemented at different level could be implemented further to get a federation sepcific implementation of the generic framework. This ability to extend any class at any level provides a highly modular environment for end user application developers. For example, if the end user application developer wants to connect the end user application to a CFSS that uses SASL-CA as an authentication mechanism, they could reuse a technology specific implementation of the generic framework, that is `CentralizedFederationSaslClientHandle` as shown in table 4.1. Similarly if a group of experts provided implementation of Shibboleth specific federation that would fall into level three and the end user application developers could reuse that implementation, hence allowing high code reusability. It should be noted that not all the classes fall into one or the other level, only the classes that provide abstraction for the features of a FSS are shown in the table 4.1. The classes that are wrapper classes or listener classes that do not represent any federation and/or technology specific implementation do not fall into any level shown in table 4.1.

Table 4.1: The generic framework modularity table

Level	Generic framework levels

1	`ClientFederationHandle`
	`IdentityProviderFederationHandle`
	`ServiceProviderFederationHandle`
	`FederationHandle`

2	`CentralizedFederationClientHandle`
	`DecentralizedFederationClientHandle`
	`CentralizedFederationFederationIdentityProviderHandle`

Continued on next page...

Table 4.1 – continued

Level	Generic framework levels
	`DecentralizedFederationFederationIdentityProviderHandle`
	`CentralizedFederationFederationServiceProviderHandle`
	`DecentralizedFederationFederationServiceProviderHandle`

3	`CentralizedFederationSaslClientHandle`
4	¡end-user application specific classes¿

Level 1 represents a very abstract level of the generic framework. It includes only the interfaces that provide a raw skeleton of the features for end user application developers. Level 2 represents federation type specific implementation. That is, the implementation of the features in the generic framework based on either it is for CFSS or DFSS. Level 3 represents technology specific implementation of the generic framework. This implementation is typically provided by expert implementers. The end user application is expected to extend one or more classes at this level to connect the FSS.

4.4 Summary

Dividing the implementation into different interfaces and abstract class gives more flexibility to the framework. The end-user application developer has flexibility to either extend the implementation available in the generic framework, or can implement the whole generic framework by implementing four main interface classes: `FederationHandle`, `IdentityProviderFederationHandle`, `ServiceProviderFederationHandle` , and `ClientFederationHandle`.

However, if the end-user application developer chooses to use only a certain part of the implementation of the generic framework he could extend the abstract classes at a different level shown in table 4.1. `CentralizedFederationSaslClientHandle` abstract class provides implementation for an end-user application that uses a SASL type of authentication mechanism to login to a CFSS. If the end-user application developer wants to develop implementation for some other authentication mechanism, he could use the same signature of the abstract class `CentralizedFederationSaslClientHandle` with a different name. For example, the end-user application developer could implement another abstract class called `CentralizedFederationCASClientHandler` with the signature as shown in program 4.5,

similar to the signature of abstract class `CentralizedFederationSaslClientHandle` as shown in program 4.4. The actual implementation of the abstract class shown in program 4.5 could be for a CAS (Central Authentication Service) type of authentication mechanism.

The table 4.2 shows the numerical facts about the number of methods implemented for the different type of federations.

Table 4.2: The generic framework features implementation summary

Description of federation type	Number of methods
Total number of methods in the generic framework	27
Total number of methods implemented for CFSS	15
Total number of methods implemented for DFSS	8
Total number of methods implemented for Shibboleth specific federation	5

Chapter 5

Analysis

This section presents the analysis of the generic framework proposed in section 3. The analysis comprises of two parts: complexity, and security analysis. The complexity analysis compares the efforts needed to implement the generic framework with other FSSs. We use the software architecture analysis presented in [26] as a base for comparing the software architecture of the generic framework. The security analysis presents the assessment of security parameters to verify the security measures in the generic framework. We use the security analysis methods and taxonomy presented [21] and [57] to form a base matrix for security analysis of the generic framework. The actual values for each column in this matrix are derived by studying and comparing the standards for different FSSs. For each type of security threat we check if the attack is possible in theory for the given type of federation. In conclusion we present a comparison discussion of the generic framework architecture with another similar approach.

5.1 Complexity analysis

The complexity analysis presents an assessment of the integration efforts needed for the generic framework and other FSSs. The main goal of this analysis is to compare the complexity involved in implementing the generic framework with implementation efforts for other FSSs.

5.1.1 Integration complexity analysis

For a given set of technologies and software architectures, the software architecture that require the implementers to know and use smaller number of technologies is considered less complex over other software architectures. The count of number of technologies used in implementing a given software architecture could also prove to be an excellent comparison of the complexity.

The existing FSSs use four main technologies to keep the system secure: PKI infrastructure, SAML [17], federation specific libraries, and authentication mechanism. *PKI infrastructure* is an arrangement that provides for trusted third party vetting of, and vouching for, end-user credentials. The PKI infrastructure is used in combination with other technologies like, X509Certificate, encryption and digital signatures. *SAML* [17] is an assertion markup language that allows for exchanging authentication assertion and attribute assertions. The *federation specific libraries* are used to handle federation specific features. For example Shibboleth libraries provides a wrapper class that holds the technical as well as non technical information about the *identity provider* and *service provider*. The *identity provider* authenticates the end-users before they issue authentication assertions and/or attribute assertions. The authentication mechanism require at least one set of libraries to deal with the authentication mechanism chosen by the *identity provider*, for example SASL [44].

Table 5.1 shows the comparison of different technologies needed for implementation of each federation. The values of the fields are defined as 'N' or 'Y'. 'N' is equal to Not Required, and 'Y' is equal to Required. We analyzed the specification documents and implementation guidelines for each of the studied federations to understand the technologies that would be required for the implementation. If a federation asks for the end-user application developer to directly interact with certain technology and use a third party API than we consider that the knowledge of that particular technology is required.

Table 5.1: Requirement of understanding of implementation technologies by end-user application developers

	PKI	SAML	Federation library	Authentication - mechanism
The generic framework	N	N	Y	N
Shibboleth	Y	Y	Y	Y
WS-Federation	Y	Y	Y	Y
Liberty Alliance	Y	Y	Y	Y

Table 5.1 shows that the generic framework requires the developers to understand and use the least number of technologies. However, when comparing the maintaining and updating in FSSs and the generic framework, other parameters affect the difficulty level as well, for example, the complexity of the implemented modules themselves. If the code is designed modular, that is keeping different functionality in separate modules, the easier it is to maintain and update the code. Because the generic framework collects the federation-specific features in the framework, and presents a simple interface to the end-user application developers, the maintaining of the code is divided into modules, and hence easy. For example the authentication mechanism that the *identity provider* uses to authenticate the end-user. If the *identity provider* changes the authentication mechanism, only the implementation of the login functionality is changed. The end-user application is not affected by that change. In the implementation for other FSSs, because the clients use the federation libraries directly, any change in the federation, or message interaction sequence would require a change in the end-user application as well. Table 5.2 in section 5.1.2 shows the comparison of the code reusability and evolution for different FSSs and the generic framework.

5.1.2 Code reusability and evolution analysis

This section presents the complexity analysis of the generic framework from the perspective of reusability and code evolution. The purpose of the analysis is to compare the amount of code that can be reused for a given FSS. [26] presents a comprehensive study of different software architecture analysis methods. [26] shows an example comparison matrix for different software architecture comparison methods. Two main software architecture methods

that are described in [26] and used in the analysis are Scenario-Based Architecture Analysis Method (SAAM) and Software Architecture Analysis Method for Evolution and Reusability (SAAMER). SAAM presents a scenario based software architecture evaluation matrix. The main goal of SAAM is to verify basic architectural assumptions and principles against the desired properties of an application. SAAMER extends SAAM to present a verification of the quality attributes like evolution and reusability in software architecture.

SAAMER presents a framework to gather and analyze the software architecture information. Gathering the information about the software architecture for analysis is divided into four perspectives, Stakeholder Objectives, Architectural Objectives, Scenarios, and Quality Assurance. *Stakeholder objectives* cover only the interaction with the software system from the stakeholder perspective. Other architectural objectives are not part of stakeholder objectives. *Architectural objectives*, on the other hand, are the objectives for the software systems. *Scenarios* are different use cases that the software system should support. If all scenarios for the software system are known, architectural goals could be considered as a subset of the scenarios. Because the use cases described in previous sections are a complete set of use cases that the FSS should support, architectural objectives are not considered in the comparison to remove redundancy. In this section we present the software architecture comparison using the stakeholder objectives and scenarios. *Quality assurance* compares the quality of the software architecture in its development phase. The next section presents the comparison of the quality assurance in the form of security analysis.

For any FSS the stakeholder could be considered as the end-user application developer. The main goal of the end-user application developers is to be able to connect to the FSS and to be able to communicate with the FSS. Connecting to the FSS includes being able to authenticate to some entity in the FSS, and communicate with the FSS means to be able to share/retrieve metadata or objects from other software systems using the same FSS. Section 3.2 shows the complete list of different use cases that the FSS should support. These use cases can be divided into either of the two stakeholder objectives. Table 5.2 shows the complete list of stakeholder objectives and the use case scenarios. The comparison is based on whether implementation for a specific use case is required in the given software architecture. Two possible values for the comparison are 'Required' and 'Not Required'. Given two software architectures, if a use case requires implementation by the end-user application developers and the other does not require implementation efforts, than the software architecture that does not require implementation is simpler for implementers. We

use the requirements presented in figure 3.4 as the basis for comparison. For a given use case if a federation requires the end-user application developers to implement that use case than we consider that the implementation is "Required".

Table 5.2 shows that the generic framework provides implementation for a larger number of use cases than any other software architecture, and hence provides a lot of code reusability, and is much simpler to implement than other software architectures. The implementers of the generic framework could take extra care while implementing the technical functionalities critical to the security of the whole system. The reusable security related technical functionalities of the FSS increases the security of the system because the end-user application developers do not need to implement all the security related technical functionalities. They benefit from the expert implementers of the generic framework for FSS.

Table 5.2: Complexity analysis

Stakeholder Objective	UseCases	Code changes analysis			
		Generic framework	WS-Federation	Libery Alliance Federation	Shibboleth Federation
Connect to federated security enabled software systems	Login to identity provider	Not required	Required	Required	Required
	Logout at identity provider	Not required	Required	Required	Required
	Federate identity to other providers	Not required	Required	Required	Not required
	Defederate identity to other providers	Not required	Required	Required	Not required
	Fetch attribute assertion	Not required	Required	Required	Required
	Fetch authentication assertion	Not required	Required	Required	Required
	Establish trust	Not required	Required	Required	Not required
	Revoke Trust	Not required	Required	Required	Not required
Interact with federated security enabled software systems	Send request for resource and interpret the response	Required	Required	Required	Required
	Specify required credential for resources	Required	Required	Required	Required
	Verify credentials	Required	Required	Required	Required

Continued on next page...

Table 5.2 – continued

Stakeholder Objective	UseCases	Code changes analysis			
		Generic framework	WS-Federation	Libery Alliance Federation	Shibboleth Federation
	Set service provider information	Required	Required	Required	Required
	Get identity provider information	Not required	Required	Required	Required
	Check provider trust	Not required	Required	Required	Required
	Check user federation status	Not required	Required	Required	Required

5.2 Security analysis

Security in FSSs is understood as securing the resource at the *service provider*, and at the same time preserving the end-user's privacy. FSSs are built on the least revealing architecture, that is the access to the resource should be granted based on the minimum required knowledge about the user. For example, if the resource in a library is accessible to any student at the university, a library connected via FSSs should be able to authorize access based on the knowledge that the end-user is a student at the university, and should not need to know the actual identity of the end-user.

[21] presents a discussion of the security analysis of the software architecture in a top down approach. A comprehensive taxonomy of the software security and analysis is presented in [57] togather with a matrix for comparing the software security with different type of possible security threats. [57] divides the security evaluation of the software architecture at three different layers, application layer, platform layer, and network layer. The generic framework is an application that is independent of the platform and the network layers.

In this section we present the security analysis at the application layer only. [57] analyzes four major types of application layer security threats, credential theft, functional manipulation, data theft, and application denial of service. *Credential theft* describes the types of threat in which an attacker gains unauthorized access to the security credentials in the end-user application without consent by the entity responsible. *Functional manipulation* is the type of attack in which the attacker is able to manipulate some or all of the functionalities provided by the software architecture. *Data theft* is the type of attack in which the attacker gets unauthorized access to the data used by the application internally or externally. *Application denial of service* is similar to the Denial of Service at the network layer. In *application denial of service* the attacker is able to affect the availability of the system.

The security analysis is divided into five properties of the software application[57]: correctness, reliability, efficiency, integrity, and usability. *Correctness* is the extent to which a program satisfies its specification and fulfills the customer's functional objectives. *Reliability* refers to the correctness of the system at any given time. *Efficiency* is the least possible use of computational resources to achieve a given task. *Integrity* is the method to verify that the data is not compromised over the network. *Usability* is the time and resource required to learn and operate the program. Some software application properties like *reliability* and *usability* are not appropriate for a software security comparison of the generic framework. However, [21] asks for more security analysis properties like *Confidentiality* and *Privacy* in addition to the properties metioned above.

Table 5.3 shows the comparison of each security application properties against the type of security threat. In table 5.3, G stands for the generic framework, W stands for WS-Federation, L stands for Liberty Alliance Federation, and S stands for Shibboleth Federation. There are three possible values for each possible combination of analysis: either empty, * or **. * means that particular combination of security threat has a negative effect on the given system, but does not pose any irreparable damage. ** shows that the particular combination of security threats have a strong negative effect, and causes irreparable harm. The empty cell shows that the given security threat does not have any effect on the given system. The analysis is a theoretical analysis, we study the specification documents and guidelines for each federation to understand the security requirements and guidelines. The security parameter that are only recommended by certain FSSs are considered vulnerable because of the possibility of miss handling by the end-user application developers during the

implementation. For a given type of security threat in a given FSS if the threat presents a strong negative effect that is irreparable, we derive a **. Similarly if it presents a negative effect which is reparable, we derive a *.

We will show how to read the above table 5.3 for the first security property *Correctness.* We will show how to read and interpret each row in this particular security property, and explain how the analysis values are derived.

To interpret the first row for *Correctness* we ask the following questions:

Given that the credential theft has happened how much of the Correctness in the generic framework?

Given that the Credential theft has happened how much of the Correctness in WS-Federation is affected?

Given that the Credential theft has happened how much of the Correctness in Liberty Alliance is affected?

Given that the Credential theft has happened how much of the Correctness in Shibboleth is affected?

EXPLAINATION:In the three major FSSs, that is Shibboleth, WS-Federation, and Liberty Alliance, end-user application developers need to handle the security credentials, and follow the standards and recommendation. This leaves a possibility for the end-user application developers to accidentally mishandle the security credentials. IF the security credential theft happens it does not affect the correctness of the other FSSs because the application does not use the stolen security credentials, hence that the security threat has negative implementation on the application but does not pose any irreparable changes in the application. Whereas in the generic framework, the end-user application developers do not have to handle any kind of security credentials. The generic framework handles security credentials for the user enforcing the recommendations in the standards provided by the FSSs. This removes any possibility of the credential theft happening in the generic framework.

Table 5.3: Security analysis

		G	W	L	S
Correctness	Credential theft		*	*	*
	Function manipulation	**	**	**	**
	Data theft		*	*	*
	App. denial of service	**	**	**	**
Efficiency	Credential theft				
	Function manipulation				
	Data theft				
	App. denial of service	**	**	**	**
Integrity	Credential theft				
	Function manipulation	**	**	**	**
	Data theft				
	App. denial of service				
Confidentiality	Credential theft				
	Function manipulation	**	**	**	**
	Data theft				
	App. denial of service				
Privacy	Credential theft		**	**	**
	Function manipulation	**	**	**	**
	Data theft		**	**	**

Continued on next page...

Table 5.3 – continued

		G	W	L	S
	App. denial of service				

To interpret the second row for *Correctness* we ask the following questions:

Given that the Function manipulation has happened how much of the Correctness in the generic framework?

Given that the Function manipulation has happened how much of the Correctness in WS-Federation is affected?

Given that the Function manipulation has happened how much of the Correctness in Liberty Alliance is affected?

Given that the Function manipulation has happened how much of the Correctness in Shibboleth is affected?

EXPLAINATION:Function manipulation happens when an attacker is able to replicate the library and the actual function call sequence, replacing it with their own function. In this situation the application loose control of the application and is completely vulnerable, and hence the security threat for each FSSs including the generic framework for this type of threat have a strong negative effect.

To interpret the third row for *Correctness* we ask the following questions:

Given that the Date theft has happened how much of the Correctness in the generic framework?

Given that the Date theft has happened how much of the Correctness in WS-Federation is affected?

Given that the Date theft has happened how much of the Correctness in Liberty Alliance is affected?

Given that the Date theft has happened how much of the Correctness in Shibboleth is affected?

EXPLAINATION:In the three major FSSs, that is Shibboleth, WS-Federation, and Liberty Alliance, end-user application developers need to handle the application date, and follow the standards and recommendation. This leaves a possibility for the end-user application developers to accidentally mishandle the application data. If the data theft has happened it does not affect the correctness of the other FSSs because the application data that is stolen

is not reused in the application and, hence the data threat has negative implementation on the application but does not pose any irreparable changes in the application. Whereas in the generic framework, the end-user application developers do not have to handle any application data. The generic framework handles application data for the user enforcing the recommendations in the standards provided by the FSSs. This removes any possibility of the data theft happening in the generic framework.

To interpret the fourth row for *Correctness* we ask the following questions:

Given that the Application denial of service has happened how much of the Correctness in the generic framework?

Given that the Application denial of service has happened how much of the Correctness in WS-Federation is affected?

Given that the Application denial of service has happened how much of the Correctness in Liberty Alliance is affected?

Given that the Application denial of service has happened how much of the Correctness in Shibboleth is affected?

EXPLAINATION:In application denial of service attack, the attacker is able to keep the application busy with the fake requests making it unavailable for actual requests. Typically the application should be protected at the network layer rather than in the implementation of any standards in an application. If application denial of service has happened in any kind of FSSs it has a strong negative effect.

Table 5.3 shows that the generic framework provide the same level of security as other FSSs, and higher level of security other security threats. It shows that the generic framework has better security than other FSS when it comes to the privacy for *data theft*, privacy for *credential theft*, and correctness for the *credential theft* compared to other FSSs. The major security threats that the generic framework deals with are *credential theft* and *data theft*. The generic framework provides a local middleware layer between the end-user application and the FSS. The generic framework implements and enforces certain security requirements for handling the credentials and data in the FSSs. This prevents problems like *credential theft* and *data theft* from.

5.3 Discussion

This section presents an architectural comparison between the generic framework and [25]. The main goal of the paper [25] is to identify the fundamental concepts, structure and operation underlying the trust realm in FSSs. Trust realm dictates how different software components from different organizations recognize, trust, and interact with each other. Trust realm is similar to the concept of federating the identity in this thesis. A part of this thesis presents the analysis of the FSSs, how different software entities interact with each other in the FSS to achieve the federated security. This analysis is used to develop the generic framework. The generic framework hides the technical details and simplifies the programming efforts for end-user applications by categorizing the FSSs and providing the implementation of certain components that remain common for certain types of federations. The design of generic framework allows the implementers to integrate the FSSs with the minimal knowledge required to operate with FSSs. However, the basic goal of this thesis and the paper [25] are the same; that is, to understand the requirements for a FSSs.

[25] first presents the understanding of the FSSs from the author's perspective and then draws a sketch of different components that would be required in a FSS. [25] introduces four main components for a FSSs: Policy Enforcement Point (PEP), Policy Decision Point (PDP), Security Token Service (STS), and Credential Processing Service (CPS). A PEP is the first point of contact for an incoming request, or a last point of contact for an outgoing request. PEP acts a coordinator for the internal message flow on how the request, incoming or outgoing, is handled. PDP is the authorization point where the security credentials in the request are checked against the security policy for the resource. STS is a service that 'issue' and 'verify' security credentials for the FSS infrastructural components. STS could be considered a superset of *identity provider* as described in this thesis. CPS is a service that transforms the security credentials that are domain independent so that they can be understood by other software entities on local network.

The four major components described in [25] work hand-in-hand to achieve federated security. [25] describes different message interaction sequences between different software components, similar to Liberty Alliance and WS-Federation, to achieve a federated security. Figure 5.1 shows a generic message interaction sequence for the architecture proposed in [25] when a request is sent to the service provider .

One of the major differences in this thesis and [25] is the approach. [25] first describes

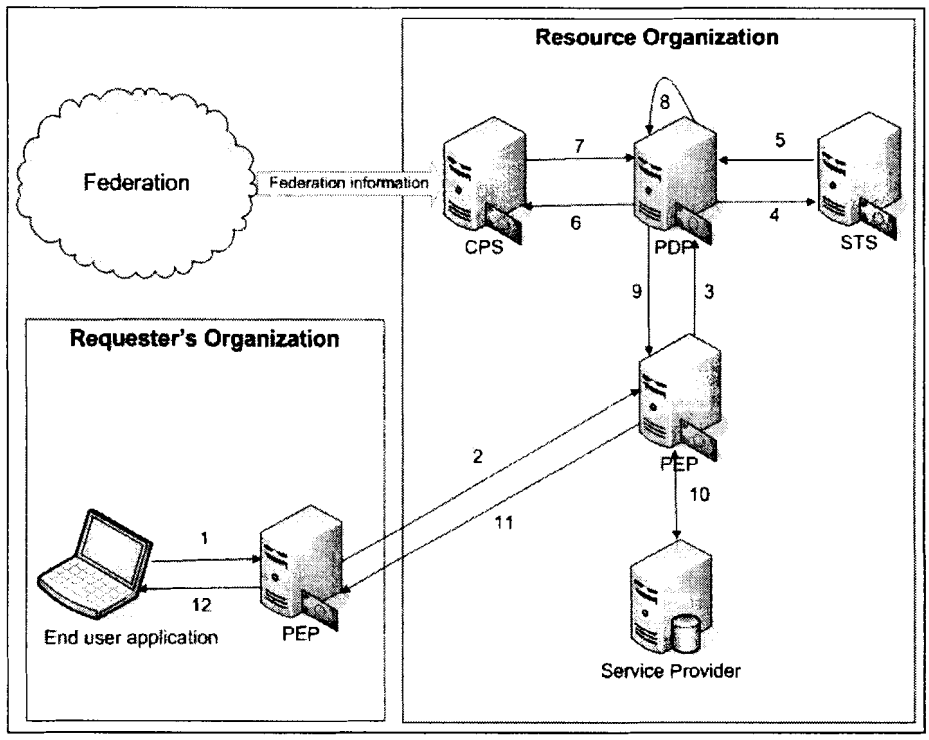

Figure 5.1: Federation Anatomy Diagram

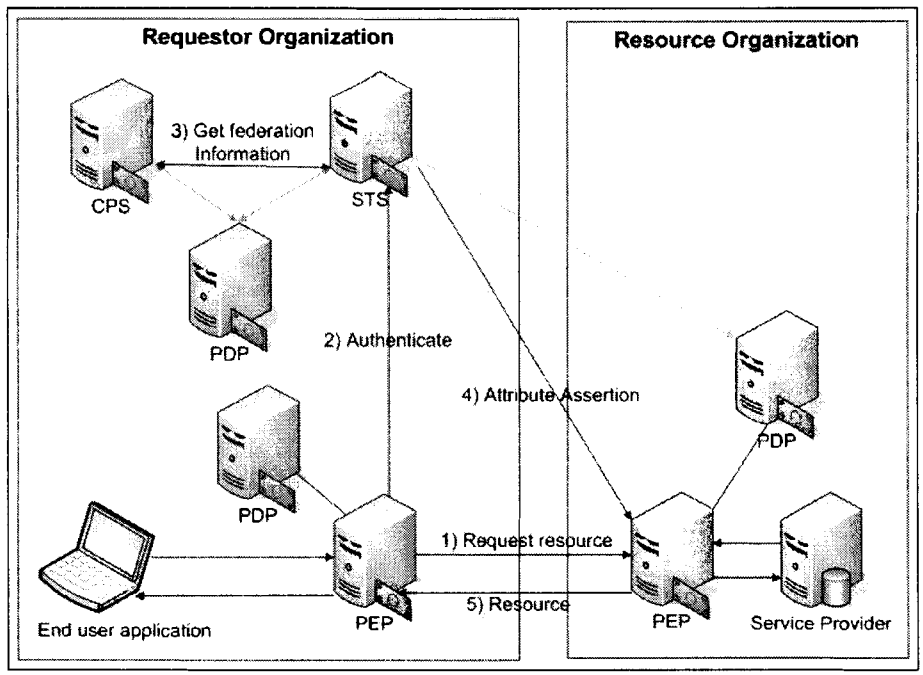

Figure 5.2: Mapping of Federation Anatomy and Shibboleth Federation

the author's understanding of the FSSs, and then tries to map other existing FSSs to their understanding. This thesis tries to understand the existing FSSs, and then generalizes the FSSs based on that understanding. The difference is clearly visible in the end results. [25] presents a mapping of their architecture with other FSSs like Shibboleth, WS-Federation, and Liberty Alliance. Figure 5.2 shows a mapping of Shibboleth architecture with the architecture described in [25]. In figure 5.2 it is hard to visualize which entities would interact with the core federation. The paper [25] introduces additional software entities that are designated very specific roles. Comparing this with the generic framework, the architecture for the generic framework is much simpler. The generic framework hides details of other software entities and technologies. If we compare figure 5.2 with figure 1.1 it is clear that the architecture presented in the generic framework is much simpler than the architecture described in [25].

5.4 Summary

The complexity analysis shows that for end-user application developers to connect the application to the federation is much less complex using the generic framework compared to implementing the FSS directly. The generic framework is modular, that is it separates the features needed by the core federation and the software entities, which allows easy updating and maintaining of the generic framework. The generic framework categorizes the FSSs and provides the implementation for certain features that remain constant throughout the type of federation. The security analysis shows that the generic framework can handle the privacy of the user in a better way than other implementations of FSSs. The generic framework enforces the security parameters recommended by the FSSs wherever applicable, hence making the system more secure. The discussion shows that the architecture in the generic framework is simpler than the architecture of the system designed with a similar goal of understanding the underlying structure of the FSSs.

Chapter 6

Future Work and Conclusion

This chapter presents the summary, scope and limitation, and future work of this research.

6.1 Summary

This research analyzed the requirements for connecting software applications to the network that uses the concept of federation as a method to make access decision about user requests. Based on the understanding of the existing FSSs, we have proposed a generic framework with interfaces that are necessary for connecting end-user applications to the federation. We have implemented a test infrastructure to demonstrate the feasibility of the proposed solution. We tested our implementation by connecting it to the InQueue [2] which is a Shibboleth-based federation. We showed that the implementation for our generic framework could be reused by other software clients to connect to similar kind of federation systems. This implementation could be used to connect to other Shibboleth-based federations like InCommon [2], Haka [1] etc. The end-user application developer would only need to change the configuration parameters so that the implementation could communicate with the desired federation entities over the Internet.

We see the following impacts made by the work in this thesis. First, the generic framework clearly defines required software entities and their interactions as well as interactions with the connected system. Secondly, by following the proposed generic framework, it is possible to connect the end-user applications to any existing FSSs. The implementation for this component can be reused for similar kind of federations. The generic framework provides a modular approach, allowing end-user application developers to reuse different

modules of federation implementation code or develop their own code implementation of each interfaces defined by the generic framework. Finally, we have implemented the generic framework and provided implementation one popular federation that is in use.

6.2 Scope and limitations

Shibboleth, Liberty Alliance, and WS-Federation are three federated security solutions used in the academic as well as professional environment. This research generalizes the federated security requirements for these providers. We only tested our proposed solution by implementing the system connected through a Shibboleth-type federated security system. To our best knowledge, based on the research done in this field, the generic framework would be able to connect software clients to any existing FSSs. However, it is possible that we might have not foreseen some basic requirements that might be introduced in the upcoming FSSs. To accommodate those requirements, some changes might be needed in the generic framework to operate with the new FSSs.

6.3 Future work

In this thesis we proposed and implemented a generic framework to connect software clients to FSSs. We implemented the generic framework for InQueue[3], which is a Shibboleth-based federation. We would like to connect our generic framework with more federations, for example, Haka[1], and Switch[6]. We would also like to implement our generic framework for DFSSs like Liberty Alliance and WS-Federation. Currently WS-Federation does not have an implementation available, and the implementations available for Liberty Alliance are commercial. We want to use the experience gained from implementation of different FSSs to conceptualize an inter-operable plug-in in the generic framework that allows one type of FSS to communicate with others. We believe that if the FSSs are generalized with proper care, it is possible to develop interoperability between different kinds of federations. Interoperability in this sense means that a software client using the implementation for a Shibboleth-based federation would be able to communicate to the WS-Federation. We understand that it is possible that the interoperability plug-in cannot support all the functionalities in each federation, but some basic functionality such as sharing and requesting objects could be fulfilled.

Appendix A

Abbreviations

- FSS - Federated Security Solution
- CFSS - Centralized Federated Security Solution
- DFSS - Decentralized Federated Security Solution
- SASL - Simple Authentication and Security Layer
- CAS - Central Authentication Service
- PKI - Public Key Infrastructure
- JAAS - Java Authentication and Authorization Service
- SAML - Security Assertion Markup Language
- XACML - eXtensible Access Control Markup Language
- PDP - Policy Decision Point
- PEP - Policy Enforcement Point
- STS - Security Token Service
- CPS - Credential Processing Service
- SAAM - Scenario-Based Architecture Analysis Method
- SAAMER - Software Architecture Analysis Methods for Evolution and Reusability

- API - Application Programming Interface
- UML - Unified Modeling Language.

Appendix B

Java Interface for the Generic Framework

The following interfaces are defined in the Generic Framework.

B.1 FederationHandle

```
package ca.sfu.federation.framework;

import ca.sfu.federation.framework.wrappers.IdentityProvider;
import ca.sfu.federation.framework.wrappers.Provider;
import ca.sfu.federation.framework.wrappers.ServiceProvider;

/**
 * This interface defines the common technical functionalities required by both
 * ServiceProviders and IdentityProviders.
 *
 */
public interface FederationHandle {
    /**
     * This method is used to defederate end user's identity from other service
     * and identity providers.
     * @param providerIdentifier the unique identifier for a provider where
     * the user wants to defederate his/her identity.
     * @return true if the identity has been defederated successfully, false
     * otherwise.
     */
    public boolean defedrateIdentity(Object providerIdentifier);
```

```
/**
 * Used to set the organizations that are part of end user's federation.
 * @param providerList the <code>Collection</code> of providers that are
 * part of end user's federation
 */
public void setFederationProviderList(java.util.Collection providerList);

/**
 * Returns the technical information about the identity provider with the
 * given identifier
 * @param providerIdentifier identifier of required IdentityProvider
 * @return IdentityProvider with the given unique identifier
 */
public IdentityProvider getIdentityProvider(Object providerIdentifier);

/**
 * Used to set the organizations that are part of end user's trust network.
 * @param providerList the <code>Collection</code> of providers that are part of
 * end user's trust network.
 */
public void setTrustedProviderList(java.util.Collection providerList);

/**
 * Checks the trust status of the Provider with given identifier.
 * @param providerIdentifier identifier of the Provider to check the trust
 * member ship
 * @return true if the Provider with given identifier is in trust network, false otherwise.
 */
public boolean isProviderTrustMember(Object providerIdentifier);

/**
 * Removes the trust relationship with Provider of given identifier
 * @param provider Provider to remove the trust
 * @return true if the trust is removed successfully with the given
 * provider, false otherwise.
 */
public boolean removeTrustMember(Provider provider);

/**
 * Adds the trust relationship with Provider of given identifier
 * @param provider Provider to add the trust
 * @return true if the trust is added successfully with the given provider,
 * false otherwise.
 */
public boolean addTrustMember(Provider provider);
```

```
/**
 * Removes the federation with Provider of given identifier
 * @param provider Provider to has to be removed from the federation.
 * @return true if the federation is removed successfully with the given
 * provider, false otherwise.
 */
public boolean removeFederationMember(Provider provider);

/**
 * Initiates the process to revoke the trust with the Provider that has
 * given identifier.
 * @param providerIdentifier the identifier of the provider to revoke
 * trust with.
 * @return true if the trust is revoked successfully, false otherwise.
 */
public boolean revokeTrust(Object providerIdentifier);

/**
 * Checks if the organization that issued the given user's credential
 * is a part of federation member list.
 * @param userCredentials usre's credentials
 * @param federationIdentifier The identifier of the federation to check
 * the membership with.
 * @return true if the given user's credentials are part of given
 * federation membership list.
 */
public boolean isUserFederationMember(Object userCredentials,
        Object federationIdentifier);

/**
 * Adds the Provider to federation member list.
 * @param provider Provider to add to the federation member list.
 * @return true if the federation membership is added successfully with
 * the given provider, false otherwise.
 */
public boolean addFederationMember(Provider provider);

/**
 * Returns the list of the Providers that are part of trust network.
 * @return Collection of the ServiceProvider and IdentityProvider that
 * are part of the trust network.
 */
public java.util.Collection getTrustedProviderList();

/**
 * Initiates the process to establish the trust with the Provider that
 * has given identifier.
```

```
     * @param providerIdentifier the identifier of the provider to establish
     * trust with.
     * @return true if the trust is established successfully, false otherwise.
     */
    public boolean establishTrust(Object providerIdentifier);

    /**
     * Returns the list of the Providers that are part of the end user's federation.
     * @return Collection of the ServiceProvider and IdentityProvider
     * that are part of the end user's federation.
     */
    public java.util.Collection getFederationProviderList();

    /**
     * Returns the technical information about the service provider with
     * the given identifier
     * @param providerIdentifier identifier of required ServiceProvider
     * @return ServiceProvider with the given unique identifier
     */
    public ServiceProvider getServiceProvider(Object providerIdentifier);

    /**
     * This method is used to federate end user's identity to other service
     * and identity providers.
     * @param providerIdentifier the unique identifier for a provider where
     * the user wants to federate his/her identity.
     * @return true if the identity is fedarated successfully, false otherwise.
     */
    public boolean federateIdentity(Object providerIdentifier);
}
```

B.2 IdentityProviderFederationHandle

```
package ca.sfu.federation.framework.identityprovider;

import java.util.List;

import ca.sfu.federation.framework.FederationHandle;
import ca.sfu.federation.framework.wrappers.Provider;

/**
 * This interface defines the technical functionalities required by the
 * IdentityProvider.
 */
interface IdentityProviderFederationHandle {
    /**
```

```
     * This method should be used to submit the technical information about
     * the IdentityProvider to the federation management.
     * @return true if the IdentityProvider information is submitted to the
     * federation management successfully, false otherwise.
     */
    public boolean setIdentityProviderInformation();
    /**
     * Returns the minimum required credentials by the ServiceProvider that
     * has the given identifier.
     * @param providerIdentifier the identifier of the ServiceProvider to
     * fetch the required credentials for.
     * @return Collection of the credentials required by the ServiceProvider.
     */
    public java.util.Collection getRequiredCredentials(Object providerIdentifier);
}
```

B.3 ServiceProviderFederationHandle

```
package ca.sfu.federation.framework.serviceprovider;

import java.util.List;

import ca.sfu.federation.framework.FederationHandle;

/**
 * This interface defines the technical functionalities required by the
 * ServiceProvider.
 */
public interface ServiceProviderFederationHandle {

    /**
     * This method should be used to submit the minimum required set of
     * credentials to the federation management. This information shows the
     * minium required credentials by this ServiceProvider to take any
     * authorization decision.
     * @param credentials Collection of the minumum required credentials
     * needed to take any authorization decision.
     */
    public abstract void setRequiredCredentials(java.util.Collection credentials);

    /**
     * This method should be used to submit the technical information about
     * the ServiceProvider to the federation management.
     * @return true if the ServiceProvider information is submitted to the
     * federation management successfully, false otherwise.
     */
```

```
    public abstract boolean setServiceProviderInformation();

    /**
     * This method is used by the ServiceProvider to take authorization
     * decision for the incoming request with end user's credentials.
     * @param userCredentials end user credentials to verify
     * @return true if the user's organization is trusted and the user is in
     * the fedration of this ServiceProvider.
     */
    public abstract boolean verifyCredentials(Object userCredentials);
}
```

B.4 ClientFederationhandle

```
package ca.sfu.federation.framework.client;

import javax.security.auth.callback.CallbackHandler;

import ca.sfu.federation.framework.wrappers.Provider;
import ca.sfu.federation.framework.wrappers.ResponseHandler;

/**
 * This interface defines the technical functionalities required by the
 * end user application to communicate and connect to a federation.
 */
public interface ClientFederationHandle {
    /**
     * This method should be used to intialize this class. The callback handler
     * chould provide all the information required to authenticate to the
     * IdentityProvider and fetch the attribute assertion.
     * @param callbackHandler callback handler.
     * @return true if the initialization process is completed successfully,
     * and the callbackhandler is able to handle all required type of
     * callbacks, false otherwise.
     */
    public boolean initialize(CallbackHandler callbackHandler);

    /**
     * This method should be used to send a request for resource to a
     * ServiceProvider.
     * @param query the query to send to the ServiceProvider
     * @param toProvider The provider to send the request for resource to.
     * @param counter counter to hold result counters if any.
     * @param responseHandler ResponseHandler to handle the response from
     * the ServiceProvider.
     */
```

```
    public void sendRequest(Object query, Provider toProvider,
            Object counter, ResponseHandler responseHandler);

    /**
     * This method is used by the end user application to authenticate to the
     * IdentitProvider.
     * @return true if the end user application is able to authenticate with
     * the IdentityProvider successfully, false otherwise.
     */
    public Object login();

    /**
     * This method should be used to logout from the IdentityProvider.
     * @return true if the user is logged out from the IdentityProvider,
     * false otherwise.
     */
    public boolean logout();
}
```

Bibliography

[1] Haka Federation. http://www.csc.fi/suomi/funet/middleware/english/index.phtml.

[2] InCommon - shibboleth implementation. http://www.incommonfederation.org/.

[3] InQueue - shibboleth implementation. http://inqueue.internet2.edu/.

[4] Liberty alliance project. https://www.projectliberty.org/index.php.

[5] Shibboleth Project. http://shibboleth.internet2.edu/.

[6] SWITCH - NetServices. http://www.switch.ch/aai/.

[7] *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES 2006, The International Dependability Conference - Bridging Theory and Practice, April 20-22 2006, Vienna University of Technology, Austria.* IEEE Computer Society, 2006.

[8] Gail-Joon Ahn, Dongwan Shin, and Seng-Phil Hong. *Information Assurance in Federated Identity Management: Experimentations and Issues*, volume 3306. January 2004.

[9] Steve Anderson et al. Web Services Trust Language (WS-Trust). http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf, 2005.

[10] Siddharth Bajaj et al. Web Services Federation Language (WS-Federation). http://specs.xmlsoap.org/ws/2003/07/secext/WS-Federation.pdf, 2003.

[11] Siddharth Bajaj et al. Web Services Policy Attachment (WS-PolicyAttachment). http://specs.xmlsoap.org/ws/2004/09/policy/ws-policyattachment.pdf, 2006.

[12] Siddharth Bajaj et al. Web Services Policy Framework (WS-Policy). http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf, 2006.

[13] Bhagyavati and Glenn Hicks. A basic security plan for a generic organization. *J. Comput. Small Coll.*, 19(1):248–256, 2003.

[14] K. Bhoopalam, K. Maly, F. McCown, R. Mukkamala, and M. Zubair. *A Flexible Framework for Content-Based Access Management for Federated Digital Libraries*, volume 3652. September 2005. doi:10.1007/11551362_49.

[15] L. C. Briand, J. W. Daly, and J. K. Wust. A unified framework for coupling measurement in object-oriented systems. *Software Engineering, IEEE Transactions on*, 25:91–121, 1999.

[16] Wentong Cai, Stephen J. Turner, and Boon Ping Gan. Hierarchical federations: an architecture for information hiding. In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, pages 67–74, Washington, DC, USA, 2001. IEEE Computer Society.

[17] Scott Cantor et al. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, 2005.

[18] Seung-Lyeol Cha, Thomas W. Green, Chong-Ho Lee, and Cheong Youn. *The Hierarchical Federation Architecture for the Interoperability of ROK and US Simulations*, volume 3397. January 2005.

[19] Brent N. Chun and Andy Bavier. Decentralized trust management and accountability in federated systems. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9*, page 90279.1, Washington, DC, USA, 2004. IEEE Computer Society.

[20] D. Cooper et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

[21] F. Copigneaux and S. Martin. Software security evaluation based on a top-down mccall-like approach. *Aerospace Computer Security Applications Conference, 1988., Fourth.*

[22] C. Cowan. Software security for open-source systems, 2003.

[23] Peter Davis et al. Liberty Metadata Description and Discovery Specification. http://www.projectliberty.org/specs/liberty-metadata-v1.1.pdf.

[24] Sabrina De Capitani di Vimercati and Pierangela Samarati. *An authorization model for federated systems*, volume 1146. June 1996. doi:10.1007/3-540-61770-1_30.

[25] I. Djordjevic and T. Dimitrakos. A note on the anatomy of federation. *BT Technology Journal*, 23(4):89–106, 2005.

[26] Liliana Dobrica and Eila Niemela. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[27] Asa Elkins, Jeffery W. Wilson, and Denis Gracanin. Security issues in high level architecture based distributed simulation. In *WSC '01: Proceedings of the 33nd conference on Winter simulation*, pages 818–826, Washington, DC, USA, 2001. IEEE Computer Society.

[28] Jacky Estublier, Herve Verjus, and Pierre-Yves Cunin. Designing and building software federations. *euromicro*, 00:0121, 2001.

[29] K. Geihs, R. KalcklÃ?sch, and A. Grode. *Single Sign-On in Service-Oriented Computing*, volume 2910. January 2003.

[30] W.G. Griswold, H. Rajan, M. Shonle, K. Sullivan, N. Tewari, Yuanfang Cai, and Yuanyuan Song. Modular software design with crosscutting interfaces. *Software, IEEE*, 23.

[31] Marek Hatala, Ty Mey (Timmy) Eap, and Ashok Shah. Federated security: Lightweight security infrastructure for object repositories and web services. *nwesp*, 0:287–292, 2005.

[32] Marek Hatala, Ty Mey (Timmy) Eap, and Ashok Shah. Unlocking repositories: Federated security solution for attribute and policy based access to repositories via web services. *ares*, 0:895–903, 2006.

[33] Marek Hatala, Griff Richards, Timmy Eap, and Jordan Willms. The interoperability of learning object repositories and services: standards, implementations and lessons learned. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 19–27, New York, NY, USA, 2004. ACM Press.

[34] Jingsha He and Ran Zhang. *Towards a Formal Framework for Distributed Identity Management*, volume 3399. January 2005.

[35] Wolfgang Hommel. *Using XACML for Privacy Control in SAML-Based Identity Federations*, volume 3677. September 2005. doi:10.1007/11552055_16.

[36] Wolfgang Hommel. *Policy-Based Integration of User and Provider-Sided Identity Management*, volume 3995. January 2006. doi:10.1007/11766155_12.

[37] Audun Jøsang, John Fabre, Brian Hay, James Dalziel, and Simon Pope. Trust requirements in identity management. In *CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 99–108, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[38] Rick Kazman, Len Bass, Mike Webb, and Gregory Abowd. Saam: a method for analyzing the properties of software architectures. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 81–90, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[39] Susan Landau et al. Liberty ID-WSF Scurity Privacy Overview. http://www.projectliberty.org/specs/liberty-idwsf-security-privacy-overview-v1.0.pdf.

[40] Seok Won Lee, Robin A. Gandhi, and Gail-Joon Ahn. Establishing trustworthiness in services of the critical infrastructure through certification and accreditation. In

SESS '05: Proceedings of the 2005 workshop on Software engineering for secure systems building trustworthy applications, pages 1–7, New York, NY, USA, 2005. ACM Press.

[41] G. Della Libera et al. Web Services Security Policy Language (WS-SecurityPolicy). http://www.oasis-open.org/committees/download.php/16569/.

[42] Chung-Horng Lung, Sonia Bot, Kalai Kalaichelvan, and Rick Kazman. An approach to software architecture analysis for evolution and reusability. In *CASCON '97: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 15. IBM Press, 1997.

[43] Paul Madsen, Yuzo Koga, and Kenji Takahashi. Federated identity management for protecting users from id theft. In *DIM '05: Proceedings of the 2005 workshop on Digital identity management*, pages 77–83, New York, NY, USA, 2005. ACM Press.

[44] John Meyers, Alexy Melnikov, and Kurt Zeilenga. Simple authentication and security layer (sasl). http://tools.ietf.org/html/4422, 2006.

[45] Tim Moses et al. eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2005.

[46] Anthony Nadalin. Web Services Security : SOAP Message Security 1.0 (WS-Security). http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, 2004.

[47] B. Pfitzmann and M. Waidner. Analysis of liberty single-sign-on with enabled clients. *Internet Computing, IEEE*, 7, 2003.

[48] Birgit Pfitzmann. *Privacy in Enterprise Identity Federation*, volume 2760. December 2003.

[49] Birgit Pfitzmann and Michael Waidner. *Federated Identity-Management Protocols*, volume 3364. September 2005. doi:10.1007/11542322_20.

[50] Rubén Prieto-Díaz. Domain analysis: an introduction. *SIGSOFT Softw. Eng. Notes*, 15(2):47–54, 1990.

[51] Susanne Röhrig and Konstantin Knorr. Security analysis of electronic business processes. *Electronic Commerce Research*, 4(1-2):59–81, 2004.

[52] Christian Schlaeger and Guenther Pernul. *Authentication and Authorisation Infrastructures in b2c e-Commerce*, volume 3590. January 2005. doi:10.1007/11545163_31.

[53] Christian Schläger, Thomas Nowey, and José A. Montenegro. A reference model for authentication and authorisation infrastructures respecting privacy and flexibility in b2c ecommerce. In *ARES* [7], pages 709–716.

[54] P. Seltsikas. Can europe's governments manage identity? *BT Technology Journal*, 23(4):80–88, 2005.

[55] Akinori Shiraga, Tsuyoshi Abe, and Masahisa Kawashima. An authentication method for interaction between personal servers based on the exchange of addresses. In *DIM '05: Proceedings of the 2005 workshop on Digital identity management*, pages 63–69, New York, NY, USA, 2005. ACM Press.

[56] Victor L. Voydock and Stephen T. Kent. Security mechanisms in high-level network protocols. *ACM Comput. Surv.*, 15(2):135–171, 1983.

[57] Huaiqing Wang and Chen Wang. Taxonomy of security considerations and software quality. *Communications of the ACM*, 46.

[58] Thomas Wason et al. Liberty ID-FF Architecture Overview. http://www.projectliberty.org/specs/draft-liberty-idff-arch-overview-1.2-errata-v1.0.pdf.

[59] Sven Wohlgemuth and GÃŒnter MÃŒller. *Privacy with Delegation of Rights by Identity Management*, volume 3995. January 2006. doi:10.1007/11766155_13.

[60] Uwe Zdun. *Loosely Coupled Web Services in Remote Object Federations*, volume 3140. January 2004.

[61] N. Zhang, L. Yao, J. Chin, A. Nenadic, A. McNab, A. Rector, C. Goble, and Q. Shi. Plugging a scalable authentication framework into shibboleth. In *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 271–276, Washington, DC, USA, 2005. IEEE Computer Society.

Index