TAMPERE UNIVERSITY OF TECHNOLOGY

PASI MUSTONEN

ACCESS CONTROL FOR LINKED DATA APPLICATIONS

Master's Thesis

Examiners: adjunct professor Ossi Nykänen, researcher Anne-Maritta Tervakari
Examiners and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 8 May 2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
MUSTONEN, PASI: Access control for Linked Data applications
Master of Science Thesis, 41 pages
May 2013
Major: Hypermedia
Examiners: Adjunct professor Ossi Nykänen, researcher Anne-Maritta Tervakari
Keywords: access control, OAuth 2.0, HTTP Basic Authentication, keyring

This thesis investigates how access control can be implemented in applications which utilize Linked Data. From the viewpoint of access control the relevant subject is, instead of Linked Open Data, specifically Linked Closed Data.

The research question was scrutinised by implementing with Java programming language a module, called AccessController, with which applications can retrieve data from servers regardless if the data is open or closed. The module also manages credentials of the end-user by storing the credentials as encrypted in a file.

For closed data protected with OAuth 2.0, the module offers an option to grant access tokens for an untrusted third-party application. With access tokens the untrusted can retrieve data owned by the granter.

The module was also used as a library when implementing a Apache Ant task. In an Ant pipeline the task has the same features as the module has in a Java application.

The AccessController module meets its requirements well. However, the most significant weakness is that the expiring time of access tokens cannot be controlled.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
MUSTONEN, PASI: Pääsynhallinta linkitetyn datan sovelluksissa
Diplomityö, 41 sivua
Toukokuu 2013
Pääaine: Hypermedia
Tarkastajat: dosentti Ossi Nykänen, tutkija Anne-Maritta Tervakari
Avainsanat: pääsynhallinta, OAuth 2.0, HTTP Basic Authentication, avainrengas

Tämä diplomityö tutkii, kuinka voidaan toteuttaa pääsynhallinta sovelluksissa, jotka hyödyntävät linkitettyä dataa (Linked Data). Pääsynhallinnan kannalta oleellista on nimenomaan linkitetty suljettu data (Linked Closed Data) linkitetyn avoimen datan (Linked Open Data) sijasta.

Tutkimuskysymystä selvitettiin toteuttamalla Java-ohjelmointikielellä moduuli, AccessController, jonka avulla sovellukset voivat hakea dataa palvelimilta riippumatta siitä onko data avointa vai suljettua. Moduuli myös hallinnoi loppukäyttäjän kirjautumistietoja säilyttämällä niitä kryptattuna tiedostossa.

OAuth 2.0:lla suojatulle suljetulle datalle moduuli tarjoaa mahdollisuuden myöntää ei-luotetulle kolmannelle osapuolella ns. access token'eita, joilla ei-luotettu applikaatio voi hakea myöntäjän omistamaa dataa.

Moduulia käytettiin myös kirjastona toteutettaessa Apache Ant task'ia. Toteutetulla task'lla voi Ant putkilinjassa tehdä samat toiminnot kuin käytettäessä moduulia Java-applikaatiossa.

AccessController-moduuli täyttää sille osoitetut vaatimukset hyvin. Suurimpana puutteena mainittakoon kuitenkin se, ettei access token'eiden vanhentumisaikaa voi hallita.

FOREWORD

This Master's thesis is written for the hypermedia studies of Tampere University of Technology. I started writing the thesis in September 2012. I would like to express my gratitude to adjunct professor Ossi Nykänen and researcher Anne-Maritta Tervakari for examining my thesis. I thank Ossi Nykänen also for suggesting the topic of the thesis and advising me through the writing process.

Tampere, May 2013

Pasi Mustonen
Tekniikankatu 14 F 301
33720 TAMPERE

CONTENTS

LIST OF FIGURES

LIST OF LISTINGS

LIST OF ABBREVIATIONS

BRM	browser relayed message
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	integrated development environment
LCD	Linked Closed Data
LOD	Linked Open Data
PIN	personal identification number
RDF	Resource Description Framework
SSO	Single sign-on
TSP	trusted service provider
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	eXtended Markup Language

1. INTRODUCTION

There are already available XML pipeline tools, e.g. Apache Ant and XProc, which can retrieve Linked Open Data. Also many programming languages, such as Java or Python, have class with which it is possible to retrieve open online data. However, these tools and languages lack of feature of retrieving Linked Closed Data. Therefore, this thesis tries to find a solution for Linked Closed Data retrieval in applications.

The writing of the thesis started in September 2012. The topic was suggested by the examiner of thesis, adjunct professor Ossi Nykänen.

1.1 Research question

As its main research question, this thesis investigates ''How to implement with Java programming language an application that retrieves linked data from resource servers protected with different authorization techniques and stores credentials for the resource servers in a keyring file". Actually, the thesis aims to implement a package, or a module, which can handle the access granting for different resource servers. The module is called AccessController.

The thesis also intends to find the answers to the following issues:

1. Can the access controlling package be used as an independent external module?
2. How do the authentications with OAuth 2.0 and HTTP Basic Authentication work together?
3. Can accesses to resource servers be revoked?
4. Can AccessController module manage any kind of credentials?
5. Is AccessController module usable?

For testing the functionality of the module, an application called Soil Sample Mapper was designed. Furthermore, AccessController's feature of granting access tokens was tested by creating two simple applications: an application which acts as the owner of the resources and grants the tokens, and an untrusted application which uses the granted tokens. For proving AccessController's suitability as an external module, an Apache Ant task was also implemented.

1.2 Structure of the thesis

This thesis is structured as follows. The second chapter depicts technologies associated to the AccessController module. Chapter 3 is a selection of studies related to the subject of this thesis. Chapter 4 presents the evaluation framework for AccessController and lists the evaluation criteria. In chapter 5 the methods, class structure and exceptions of AccessController are presented. Chapter 6 describes the case studies developed for testing the functionality of AccessController: the chapter covers Soil Sample Mapper, the untrusted application and the Apache Ant task. Chapter 7 states the methods and the results of the thesis. The last chapter, chapter 8, draws the conclusions.

2. TECHNOLOGIES

This chapter deals with technologies related to the AccessController module. The AccessController module is designed for retrieving both Linked Open Data and Linked Closed Data and it provides features for obtaining closed data from resource servers protected with OAuth 2.0 or HTTP Basic Authentication.

2.1 Linked Data

Linked Data is machine-readable and explicitly defined data published on the Web. Linked Data is linked to other external data sets and can be linked to from external data sets. [3]

A set of rules, the 'Linked Data principles', describes how data published on the Web becomes a part of a single global data space [3]:

- Use URIs as names for things
- Use HTTP URIs so that people can look up those names
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
- Include links to other URIs, so that they can discover more things

These rules are actually expectations of behaviour. Hence, breaking them does destroy data, but misses an opportunity to interconnect the data. The lack of interconnection of data reduces possibilities to reuse data in unexpected ways. And it is expressly the unexpected re-use of information which is the value added by the Web. [1]

Linked Data is divided into two types by whether the data has access or legal restrictions or not. These two types are Linked Open Data which does not has any restrictions (LOD), and Linked Closed Data (LCD) which has access or legal restrictions. Two following to sections detail LOD and LCD.

2.1.1 Linked Open Data

Berners-Lee states that Linked Open Data is Linked Data which is released under an open licence, which does not impede its reuse for free [1]. The following star scheme defines the quality of LOD [1]:

- 1 star: Available on the web but with an open licence, to be Open Data
- 2 stars: Available as machine-readable structured data
- 3 stars: In addition to the previous qualifications, available in non-proprietary format
- 4 stars: In addition to the previous qualifications, uses open standars from W3C (RDF and SPARQL) to identify things for making the data linkable
- 5 stars: In addition to the previous qualifications, is linked to data of publishers to provide context

Linked Data does not have to generally be open and it possible to have 5-star Linked Data without it being open. However, if the data claims to be Linked Open Data then it has to be open to get any star at all. [1]

2.1.2 Linked Closed Data

Linked Closed Data is Linked Data for which access to or use to is restricted technically or legally. The restrictions of Linked Data vary with the access restrictions (who is permitted to 'see' data) and the license (what is permitted to do with the data). The access can be restricted only by the resources of the dataset host; restricted, open to users who meet specific access criteria; or private, open only to its owner.

Dataset licenses may waive all intellectual property rights; impose, for example, attribution or copyleft; or specify permitted uses of a dataset. Usually, a restrictive license goes along with an access restriction because otherwise license breaches would be difficult to detect and punish. [5]

2.2 OAuth 2.0

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. [10]

In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource owner. [10]

Instead of using the resource owner's credentials to access protected resources, the client obtains an access token - a string denoting a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization

server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server. [10]

2.2.1 Obtaining an access token

Figure 2.1 shows the abstract OAuth 2.0 flow, which describes the interaction in authorization. In step A the client requests authorization directly from the resource owner or indirectly via the authorization server as an intermediary. In step B the client receives an authorization grant which represents the resource owner's authorizaton. The client requests an access token by presenting the authorization grant in step C. In step D the authorization server authenticates the client and validates the authorization grant. If the authorization grant is valid the server issues an access token. In step E the client requests the protected resource from the resource server and authenticates by presenting the access token. If the access token is valid the the resource server serves the requested resource in step F. [10]

The methods used by the resource server to validate the access token are beyond the scope of OAuth 2.0 specification. However, the methods involve an interaction or coordination between the resource server and the authorization server. [10]

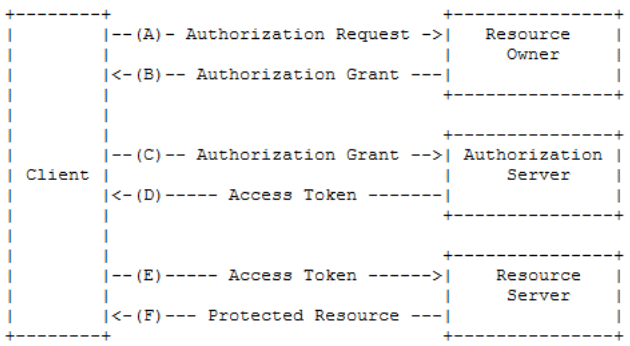

Figure 2.1: Obtaining an access token

Listing 2.1 shows a successful HTTP response of an access token request. The response is a JSON document including required, recommended and optional parameters. The required paremeters are *access_token* and *token_type*. The parameter *expires_in*, which states the lifetime in seconds of the access token, is the recommended parameter. *Refresh_token* is an optional parameter. *Scope* is an optional parameter if its value is identical to the scope requested by the client, otherwise it is a required parameter. [10]

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
```

```
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

Listing 2.1: Succesful response to an access token request

Typical lifetime of an access token is an hour. However, the lifetime is not guaranteed and the server may postpone the expiration time. [7]

2.2.2 Refreshing an expired access token

The flow of obtaining a new access token by using a refresh token is described in figure 2.2. In step A the client request an access token from an authorization server and in step B the server issues an access token and refresh token if the authorization grant was valid. In step C the client requests a protected resource from the resource server and if the access is valid the resource server serves the request. Steps C and D repeat until the access token expires and the resource server returns an invalid token error. If the client knows the access token expired it skips to step G where the client requests a new access token by authenticating with the authorization server and presenting the refresh token. In step H the authorization server authenticates the client the client and validates the refresh token. If the refresh token was valid the server issues a new access token and optionally a new refresh token. [10]

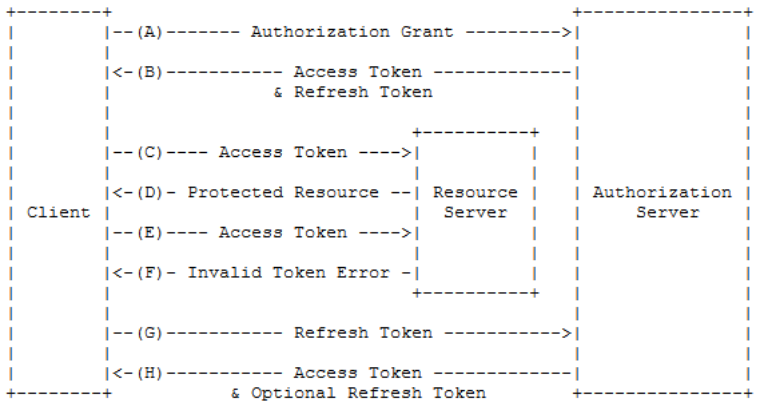

Figure 2.2: Refreshing an expired access token.

A refresh token is a credential used to obtain an extra access token when the current access token becomes invalid or expires. A refresh token is typically a long-lasting credential and therefore bound to the client it was issued. If the client type is confidential or the client was issued client credentials, the client must authenticate with the authorization server. [10]

2.2.3 Authorization grant

An authorization grant is a credential representing the resource owner's authorization used by the client application to obtain an access token. The specification of OAuth 2.0 defines four types of an authorization grant; authorization code, implicit grant, client credential grant and resource owner password credentials. [10]

The authorization code is obtained by using an authorization server as an intermediary between the client and resource owner. The implicit grant type does not issue intermediate credentials (e.g. an authorization code). In the implicit flow the client is issued an access token directly. The resource owner password credentials (i.e. username and password) can be used directly as an authorization grant to obtain an access token. This grant type should be used only if there is a high degree of trust between the resource owner and the client. The client credentials grant can be used when the authorization scope is limited to the protected resources under the control of the client. [10]

The grant type used by AccessController is an authorization code. Figure 2.3 illustrates the flow of obtaining an access token by the authorization code grant type. In step A the client directs the resource owner's user-agent to the authorization endpoint. In step B the authorization server authenticates the resource owner. If the owner grants access the authorization server redirects the user-agent back to the client using the redirection URI provided earlier (step C). The redirection URI includes an authorization code. In steps D and E the client exchanges the authorization code for an access token (described in section 2.2.1). [10]

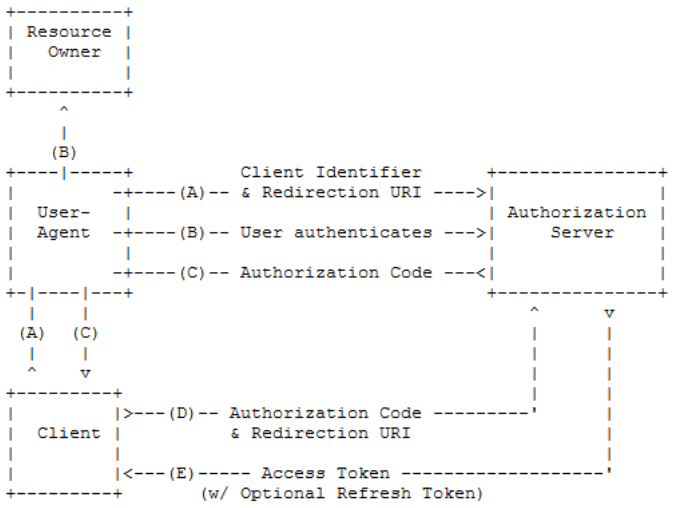

Figure 2.3: Obtaining an access token with authorization code grant.

The client constructs the authorization request URI by adding the following parameters to the query component of the authorization endpoint URI. The required

parameters of the URI are *response_type* and *client_id* which is the client identifier. When using authorization code grant the value of *response_type* must be set to "code". Parameters *redirect_uri* and *scope* are optional. Parameter *state*, which is an opaque value used by the client to maintain state between the request and callback, is a recommended parameter. [10]

2.2.4 Token revocation

An authorization server can support the revocation of tokens by providing a token revocation endpoint. A compliant endpoint must be able to revoke refresh tokens but access token revocation may be also supported. [11]

Since token revocation requests are plain text credentials in a HTTP request, the authorization server must support TLS 1.0 or its future replacements. The *token* parameter is the only requested parameter of the client's HTTP request which also includes the authentication credentials of the client. [11]

The authorization server first validates the client credentials (if present) and verifies whether the client is authorized to revoke the particular token. Next the authorization server invalidates the token. If the token is a refresh token and the server can revoke access tokens, then the the server should also invalidate all access tokens issued for that refresh token. The authorization server indicates a succesful revocation by a HTTP status code 200, a failed client authentication by a code 401, unauthorized client for revoking the token by a code 403, and all the other error conditions by a code 400. [11]

The authorization server or a related entity may also offer its end-users a self-care portal for deleting access given to third-party client applications. Such a portal offers the possibility to an end-user to look at and revoke all access grants the user once authorized. [11]

2.3 HTTP Basic Authentication

The basic authentication scheme of HTTP protocol is based on the model that the user agent must authenticate itself with a user-ID and a password for each realm. The scheme assumes that the connection between the client and server is trusted and therefore it is a non-secure method. [2]

To receive an authorization the client base64 encodes the user-ID and password separated by a single colon (":") character and sends the result string of the encoding in credentials. [2]

3. RELATED APPLICATIONS AND STUDIES

This chapter presents applications relevant to the theme of this theses. The selected surveys scrutinise on applications in which the end-users login into a SSO system.

3.1 Security of SSO mechanisms

The security of the deployed single sign-on mechanisms was investigated by Wang et al. The methodology in the research was the following. The researches were not able to observe the messages between the identity provider (e.g. Facebook) and relying parties. Hence, an automated black-box test was made on the HTTP messages which the web browser passes between the relying party and the identity provider. These messages are called browser relayed messages (BRM). [16]

The most important result of the research was that there are security-critical logic flaws in SSO systems. For example, a malicious application could act as an identity provider, relying party or browser and in this way steal the user's credentials. However, these flaws can be discovered from BRMs without even access to source code or other insider knowledge of the systems. [16]

3.2 Challenges and concerns which web users face when using OpenID for authentication

Sun et al. researched the challenges and concerns web users face when using a SSO solution called OpenID for authentication. They intended to find answers to the following research questions [15]:

1. What are a web user's perceptions, challenges, concerns and perceived benefits when using OpenID?
2. How do the the OpenId login user interface and workflow impact users' mental models?
3. What factors influence users' adoption intentions?
4. What changes in the login process could improve users' experience and adoption incentives?

The researchers made an in-lab exploratory study for understanding these requirements. In the study a phishing resistant, privacy-preserving browser add-on was designed. [15]

The researchers found, for example, the following concerns and misconceptions: no perceived urgent need for Web SSO, single-point of failure and and security misconceptions of which one example is that the mojority of participants thought that they were giving their usename and password to the relying parties directly. Additionally, the participants were concerned about phishing attacks to identity providers, privacy and trustworthy of relying parties, and they had also account linking misconceptions. [15]

As a results the researchers perceived that users value the concept of SSO but require a usable, secure and privacy-preserving experience. Therefore, many users would use a Web SSO solution only on relying-party websites that are familiar and trustworthy. [15]

3.3 Resistance of browser-based OAuth authorization

Richardson designed *launchpadlib* library which presents *Launchpad* website as a densely interconnected network of Python objects. The launchpadlib application uses OAuth for authorizing users to Launchpad website. The site hosts colloborative deveplopment for the Ubuntu Linux distribution, many of Ubuntu's component packages, and many unrelated open source software projects. It protects its resources with OAuth authentication. [14]

The launcpadlib application opens a web page on the address www.launcpad.net in the end-user's web browser. The web page explains that the application wants to access to the end-user's Launchpad account. The end-user either denies the access or grants a limited or full access. If the end-user grants an access the access token is authorized and launchpadlib can use Launchpad on the end-user's behalf. [14]

According to the paper the best way to distinguish between legimate delegation of authority and a fishing attempt is to handle the authentication from the end-user's web browser because it is a trusted client: the user trusts it with passwords and the user trust its address bar. [14]

However, some developers did not like the authentication system of launcpadlib. Hence, they wrote hacks which simulated the browser-based authorization protocol by asking an username and password. From an architectural standpoint these hacks were spyware. [14]

A compromise in the authentication system was made: it is possible to authorize an OAuth access token without opening the end-user's web browser. In this solution the authorization is made by *pinentry* which is a suite of small programs for reading passphrases and PIN numbers in a secure manner. [14]

3.4 Gaining access to organisational resources in a secure way with login accounts of social networks

Chadwick et al. designed a federated identity management service that allows users to access organisational resources using their existing login accounts in social networking (SN) sites, without compromising the security of the organisation's resources. [4]

In the service users link together their various accounts (organisational and external). The strongest registration procedure of of one linked account is leveraged by the other site's login processes. [4]

According to the paper the major problem in using social networks for SSO is that there is no authentication of their users' identities at the time of registration. This means that the physical identity and virtual identity are not binded together. [4]

A trusted service provider (TSP) manages the account linking procedure. The TSP remembers the set of account that the user has linked together. The TSP has to be trusted by the service providers and users. The trust is achieved when the TSP is part of the service provider's domain and managed by the service provider itself. [4]

To define the security of authentication the researches determined the lowest level of assurance. The components of the lowest level of assurance are the following; the protocol that is used to communicate the authentication assertion between the identity provider and service provider; the authentication mechanism and tokens that are used to authenticate the user by the identity provider; and the authentication procedure used by the identity provider during user enrollment and registration. [4]

The service achieved its initial objectives and allows external users access university resources using their existing external login accounts, without having to first register a new account at the university. Furthermore, students are able to link their external and university accounts together. [4]

4. EVALUATION FRAMEWORK

This chapter presents the evalution framework for evaluating the AccessController module. The last section of the chapter lists the evaluation criteria.

4.1 SSO framework features

There are two different scenarios which can be supported by a SSO framework like AccessController; login first and application first. In the login first scenario the user first performs login to a SSO infrastructure and then chooses a service to access. In the application first scenario the user first tries to access a service but because the user has not been authenticated yet, the service redirects the user to the login service and after a successful logon the user is redirected back to the service. [12]

SSO framework should provide a logout service, rather a single sign-off service in which a single action of signing out revokes access to multiple servers. [12]

An important security requirement for SSO frameworks is a secured interaction between the client and the authorization server or resource server. The secured interaction can be achieved, for example, with TSL or SSL protocol. [12]

4.2 Password manager features

Some features of password managers can be compared to features of AccessController. Password managers can be web browser's built-in password stores, or web-based or local (desktop) applications.

In order to access credentials a secure password manager requires a masterpassword which only the user knows. The encrypted masterpassword is stored in local file system in desktop applications and online (e.g. in a cloud) in web-based applications. Some password managers provide an option to use a key file instead of a master password or combine the key file and password methods.

For extra security multi-factor authentication, for example a master password and one time passwords, can be used. Because web-based applications can be accessed from any device they have better accessibility than desktop applications. On the other hand, in desktop applications the user has control over where the data is stored. Users do not have that control in web-based password managers. [13]

4.3 Other features

Some SSO frameworks has centralized management which includes, for example, user provisioning, profile management, trust configuration and usage monitoring. [9]

Considering tokens of OAuth 2.0 there are some desirable features for AccessController. The scopes and expiring times of tokens should be possible to change after obtaining the tokens from the authorization server. For security reasons also the scope of a token should be granted for reaching only access to necessary resources.

4.4 List of evaluation criteria

The list of evaluation criteria for the AccessController module is the following:

1. Does the module support both the login first and application first scenarios?
2. Does the module provide a single sign off service or at least a logout service?
3. Is the interaction between the client and the authorization server or resource server secured?
4. Does the module require a masterpassword for accessing credentials?
5. Does the module provide an option to use a key file instead of a master password or combine the key file and password methods?
6. Can a multi-factor authentication, for example a combination of a master password and one time passwords, be used?
7. Can the keyring file be accessed from any device?
8. Does the end-user have control over where the credentials are stored?
9. Does the module has a centralized management which includes, for example, user provisioning and usage monitoring?
10. Can the scope and expiring time of an access token be changed after the token is obtained from the authorization server?
11. Does the scope include only the minimum access for resource servers?

AccessController will be evaluated according this list in section 7.2.

5. ACCESSCONTROLLER

AccessController package can be used as a module for helping to retrieve data, especially Linked Data, from open or closed online resource server, or from a local file system. Regardless of the type of the resource's location, data is retrieved with a single method; 'getData'.

5.1 Interface

The methods of the AccessController interface are presented in the listing 5.1. The 'getData' method is for retrieving data from all kind of resource servers and locations; servers protected with HTTP Basic Authentication or OAuth 2.0, servers providing open data and local file systems. This method takes a username and the URI of a resource as its parameters.

The 'openKeyRing' method grants access to a keyring file which includes usernames and passwords for resource servers. By providing a username and the URI of a resource server as parameters for the 'revokeAccess' method a user can revoke the access of the username for the server. The method removes a password or refresh token from a keyring file and revokes the refresh token if the method was called for revoking a token.

```
package accessController;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URISyntaxException;
import java.util.List;

import com.google.api.client.auth.oauth2.Credential;


public interface AccessController {
        void openKeyRing(String masterPassword,
        String fileName)
        throws IOException;
        String getData(String username, String fileUrl)
        throws IOException, URISyntaxException;
        boolean revokeAccess(String username,
        String server)
        throws FileNotFoundException,
        IOException, URISyntaxException;

        String getBasicAuthProtectedData(String username,
```

```
        String fileUrl)
        throws MalformedURLException, IOException;
        String getOAuth2ProtectedData(String username,
        String fileUrl)
        throws IOException;
        String getLocalData(String filePath)
        throws FileNotFoundException, IOException;
        String getOpenData(String url)
        throws IOException, URISyntaxException;

        String getAccessToken(String username,
        String service)
        throws IOException;
        String getDataWithAccessToken(String fileURI,
        String accessToken)
        throws IOException;
        void revokeToken(String token, String service)
        throws IOException, URISyntaxException;
}
```

Listing 5.1: AccessController interface

At the moment, the interface is implemented by the Access class which is presented in the following sections of this chapter.

5.2 Class structure

The class structure of AccessController is presented in the figure 5.1. The Access class implements the AccessController interface. The Access class uses four helper classes for retrieving data; BasicAuthHelper for HTTP Basic Authentication protected resources; OAuth2Helper for OAuth 2.0 protected resources; OpenDataHelper for open data resources; and LocalDataHelper for data in local file systems.

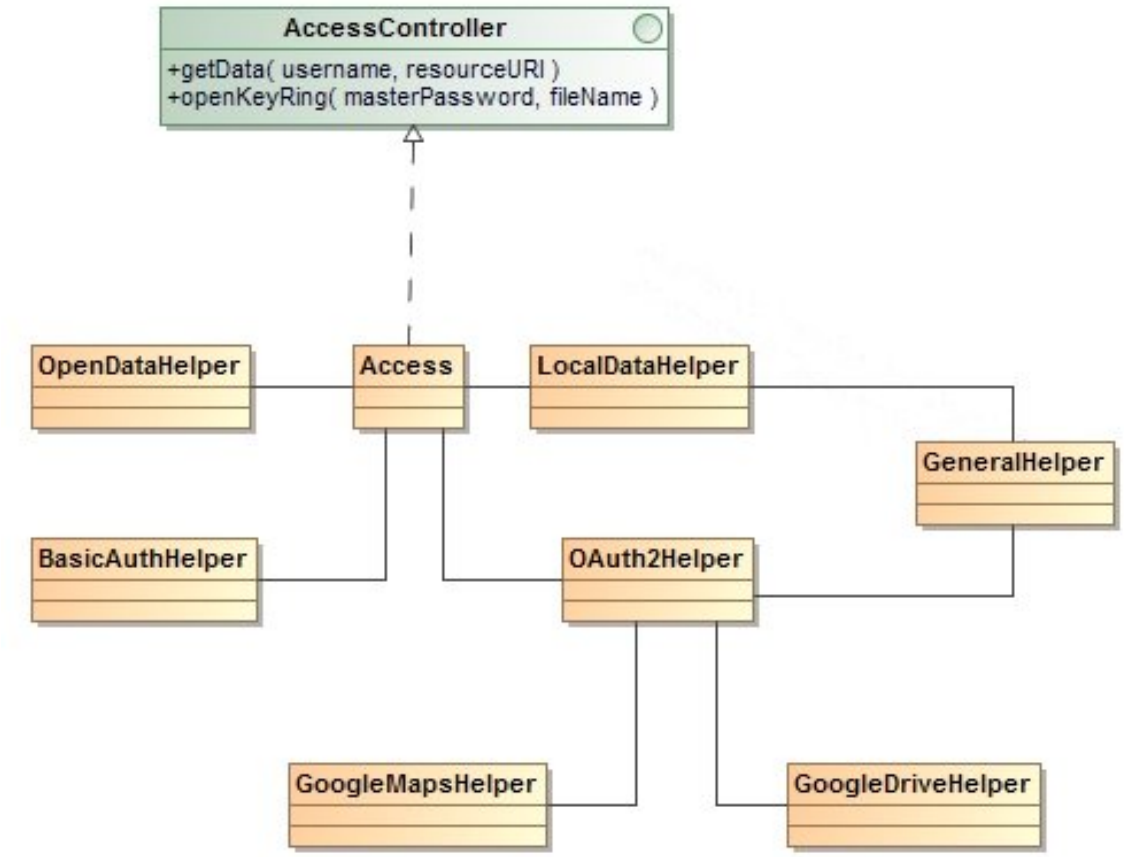

Figure 5.1: Class structure of AccessController.

The Access class needs the KeyRingHelper class to open an access to a keyring file. The KeyRingHelper class, for its part, uses the KeyRing class.

5.3 KeyRing class

Keyring files are accessed via a KeyRing object. The KeyRing class is a renamed version of KeyChain.java; a Java class developed by Sun Microsystems, Inc. The class allows storing multiple credentials or passwords in a central password store. Access to this central store is controlled through a master password. A keyring entry includes a username, server and password. The class uses the KeyStore class of Java's security package to store credentials and encrypt them with SHA-1 algorithm. The actual store for the KeyStore class can be any OutputStream object and in the case of AccessController the store is a FileOutputStream object. [6]

5.4 Retrieving data

The listing 5.2 shows the procedure of resolving the type of the authorization of the resource server. The resolvation is done by the Access class.

```
@Override
public String getData(String username, String fileURI)
        throws IOException, URISyntaxException
{
        if (OAuth2Helper.isOAuth2Protected(fileURI))
        {
                return getOAuth2ProtectedData(username,fileURI);
        }
        else if (BasicAuthHelper.isBasicAuthProtected(fileURI))
        {
                return getBasicAuthProtectedData(
                username, fileURI);
        }
        else if (fileURI.contains("http://"))
        {
                return getOpenData(fileURI);
        }
        else
        {
                return getLocalData(fileURI);
        }
}
```

Listing 5.2: Resolving the authorization technology of the reource server

If an OAuth 2.0 protected resource is requested, the authorization server of the resource server is resolved in the OAuth2Helper class as shown in listing 5.3.

```
public static String getFile(String username,
        String fileURI, KeyRing keyRing,
        String keyRingFile) throws IOException
{
        List<String> scopeAndAuthServer =
        getScopeAndAuthServer(fileURI);
        String scope = scopeAndAuthServer.get(0);
        String authServer = scopeAndAuthServer.get(1);

        if (authServer.equals(GOOGLE_AUTHSERVER))
        {

                return GoogleHelper.
                getFile(username, fileURI,
                Arrays.asList(scope), authServer,
                REDIRECT_URI, CLIENT_ID,
                CLIENT_SECRET, keyRing, keyRingFile);
        }
        else
        {

        }

        return null;
    }
```

Listing 5.3: Resolving the authorization server

In case the authorization server belongs to Google, the service where the resource locates, is resolved as described in listing 5.4. After that the resource is retrieved from the specific service with a Credential object which includes an access token.

```
Credential credential = getCredential(username,
scopes, authServer, redirectURI, clientId,
clientSecret, keyRing, keyRingFile);

if (scopes.get(0).
equals(OAuth2Helper.DRIVE_SCOPE))
{
        return getDriveFile(credential, fileURI);
}
else if (scopes.get(0).
equals(OAuth2Helper.MAPS_SCOPE))
{
        return getMap(credential, fileURI);
}
else
{

}
```

Listing 5.4: Resolving the requested Google service

If the request resource is protected with HTTP Basic Authentication then the getFile method of the BasicAuthHelper class is called. The method Base64 encodes the username and password, uses the encoded string as a property when opening a connection to the server and gets an input stream from the URLConnection object. The encoding and getting the input stream are presented in the listing 5.5. Finally the input stream is transformed into a String object and returned to the Access class.

```
String userPassword = username + ":" + password;
String encoding = Base64.encode(userPassword.getBytes());
URLConnection uc = url.openConnection();
uc.setRequestProperty ("Authorization", "Basic " + encoding);
InputStream is = null;
try
{
        is = uc.getInputStream();
}
```

Listing 5.5: Opening a connection to a resource server protected with HTTP Basic Authentication

If the getData method of the Access class resolves that the requested resource is open data, then the getData method of the OpenDataHelper class is called. The method is presented in listing 5.6. In the method the URI object is constructed from the protocol, path, query and other parts of the URL object. Then the URI object is transformed into a URL object. The input stream of the URL object is opened. The return value of the method is a String object which is transformed from the stream.

```
public static String getData(String fileUrl)
                throws IOException, URISyntaxException
{
        URL url = new URL(fileUrl);
```

```
        URI uri = new URI(url.getProtocol(), url.getUserInfo(),
        url.getHost(), url.getPort(), url.getPath(),
        url.getQuery(), url.getRef());

        url = uri.toURL();

        return GeneralHelper.
                        readInputStreamAsString(url.openStream());
}
```

Listing 5.6: GetData method of the OpenDataHelper class

Finally, if the URI of the resource did not match to any of the previous conditions the resource is resolved to be in the local file system and the getData method of the LocalDataHelper class, which is stated in listing 5.7, is called. In the method the requested file is read to a FileInputStream object. Then the stream is transformed into a String object which is returned to the Access class.

```
public static String getData(String filePath) throws IOException
{
        FileInputStream fis = new FileInputStream(filePath);
        return GeneralHelper.readInputStreamAsString(fis);
}
```

Listing 5.7: GetData method of the LocalDataHelper class

An access to a new type of resource location could be added by creating a new condition in the getData method of the Access class and implementing a helper class for this new access type. The helper class must have a method which returns the requested data as a String object.

5.5 Obtaining credentials for OAuth 2.0

In case where there are no credentials in the keyring file for the OAuth 2.0 protected resource server, in which the resource requested in the call of the 'getData' method exists, the helper class of the requested service will obtain new credentials (figure 5.2).

The helper class tries to retrieve a refresh token by calling the 'getPassword' method of the KeyRing class but because the credentials do not exist the method returns value 'null'. The helper class resolves the URL of the authorization server and calls the 'authorizationCommandLineUI' method of the OAuth2Helper class for retrieving the authorization code from the user. The helper class exchanges the code to an access token and a refresh token with the authorization server. Finally, the helper class stores the refresh token in the keyring file. The procedure of the GoogleHelper class for authorizing the user and exchanging the code is described in the listing 5.8.

Instead of redirecting an authorization code from the authorization server to the

client, AccessController asks the end-user to type the code for the client. This approach is selected because directing the code to the client requires the device, on which the client is launched, to act as web server. Therefore, the redirect URI is set to ' urn:ietf:wg:oauth:2.0:oob' instead of 'http://localhost'. [8]

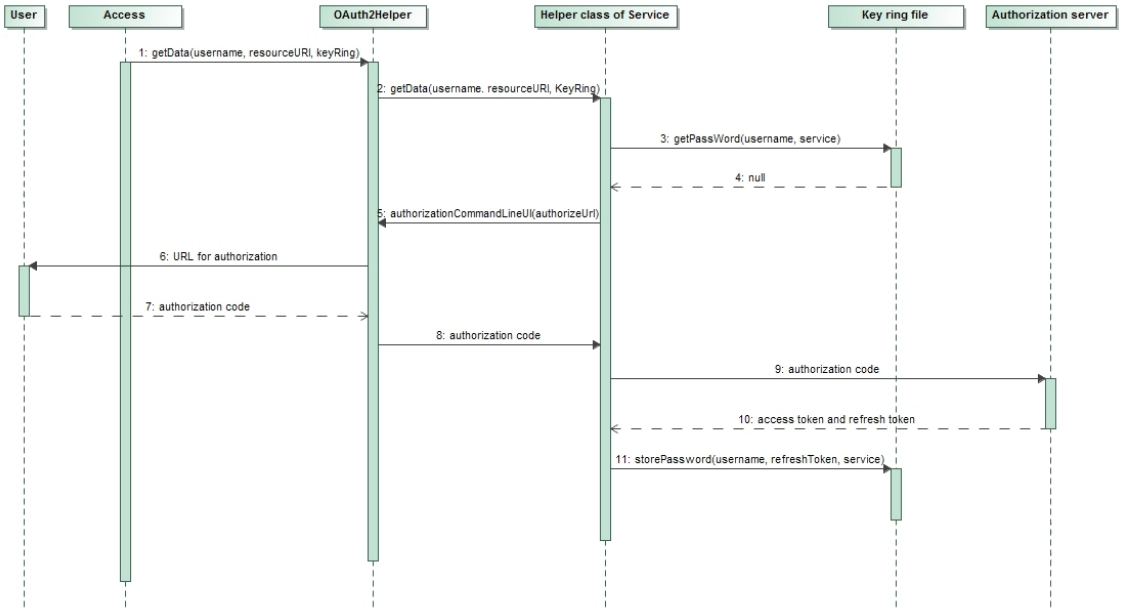

Figure 5.2: Obtaining credentials for OAuth 2.0.

```
String authorizeUrl = new GoogleAuthorizationCodeRequestUrl(
authServer,clientId,redirectURI,scopes).setState("").build();

String authorizationCode = OAuth2Helper.
authorizationCommandLineUI(authorizeUrl);

credential = exchangeCode(authorizationCode, clientId,
clientSecret, scopes, redirectURI);

keyRing.addPassword(username, scopes.get(0),
credential.getRefreshToken().toCharArray());
keyRing.store(new FileOutputStream(keyRingFile));
```

Listing 5.8: GoogleHelper's procedure for authorization and the authorization code exchanging

Because obtaining credentials is implemented with Google's own libraries, obtaining OAuth 2.0 credentials for another service provider requires creating a new helper class. The new helper class uses the OAuth2Helper class and the helper class is called in OAuth2Helper's getData method.

5.6 Obtaining credentials for HTTP Basic Authentication

If BasicOAuthHelper gets a value of 'null' when calling KeyRing's 'getPassword' method, then it asks the user for the password for the combination of the username

and folder on the HTTP server (figure 5.3). BasicAuthHelper stores the combination of username, password and folder in the keyring file for the future use.

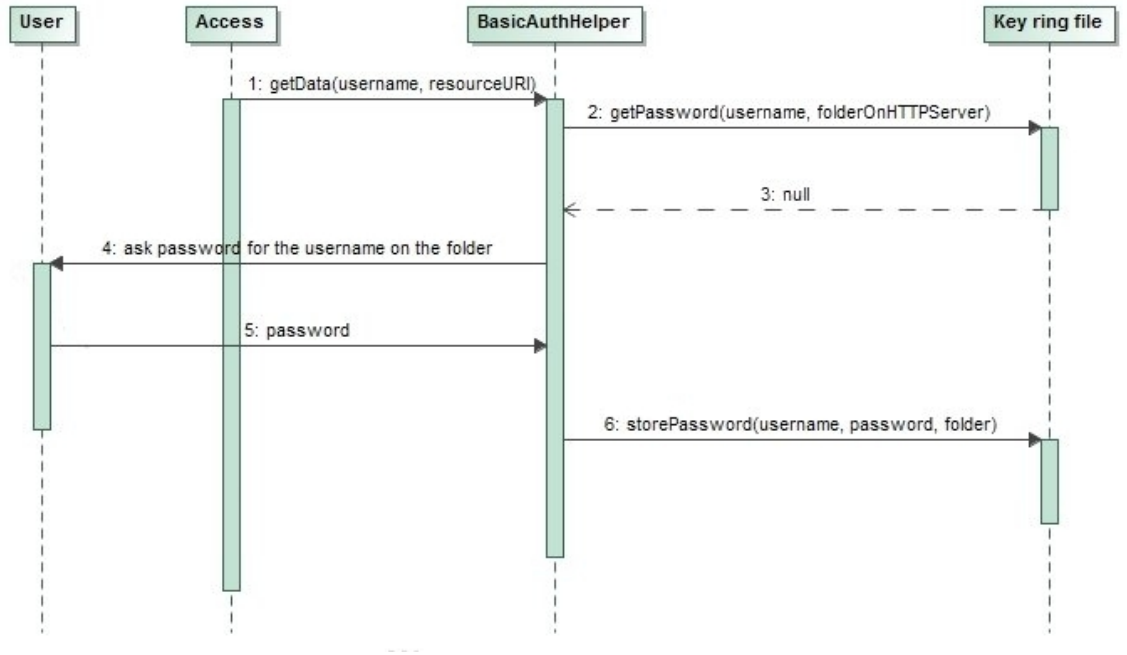

Figure 5.3: Obtaining credentials for HTTP Basic Authentication.

5.7 Access revocation

From the perspective of a client application, the AccessController module's access revocation works analogically for both OAuth 2.0 and HTTP Basic Authentication protected resource servers. An access is revoked by calling the 'revokeAccess' method with the username and server as its parameters.

Listing 5.9 presents the method of the Access class for the access revocation. If the server, to which the access is revoked, is protected with HTTP Basic Authentication then the method only deletes the specific entry in the keyring file. In case of an OAuth 2.0 protected server, the specific refresh token is first revoked by sending a HTTP request to the revocation end-point of the authorization server. After the token revocation the keyring entry is deleted.

```
@Override
public boolean revokeAccess(String username, String server)
        throws FileNotFoundException,
        IOException, URISyntaxException
{

        if (OAuth2Helper.isOAuth2Protected(server))
        {
                String scope = OAuth2Helper.
                getScopeAndAuthServer(server).get(0);
                String token = keyRing_.
                getPassword(username, scope);
```

```
                if (OAuth2Helper.getScopeAndAuthServer(server).
                        get(1).equals(
                        OAuth2Helper.GOOGLE_AUTHSERVER))
                {
                        String url = OAuth2Helper.GOOGLE_REVOKE +
                        token;
                        getOpenData(url);
                        keyRing_.removePassword(username, scope);
                        keyRing_.store(
                        new FileOutputStream(keyRingFile_));
                        return true;
                }
                else
                {
                        return false;
                }
        }
        else if (BasicAuthHelper.isBasicAuthProtected(server))
        {
                keyRing_.removePassword(username, server);
                keyRing_.store(
                new FileOutputStream(keyRingFile_));
                return true;
        }

        return false;
}
```

Listing 5.9: Access revocation

Contrary to the specification of OAuth 2.0, the token revocation of Google does not require sending also client credentials beside the token to be revoked. This means that the revoker of the token does not prove its identity and therefore a malicious third party can revoke the token.

5.8 Granting access for untrusted applications

The AccessController module has a feature for granting a temporary access for untrusted third-party applications. The granting happens by calling the 'getAccessToken' method with parameters 'username' and 'service'. The OAuth2Helper class resolves the authorization server from the 'service' parameter and calls the concerned helper class (listing 5.10).

```
public static String getAccessToken(String username,
        String service, KeyRing keyRing, String keyRingFile)
        throws IOException
{
        String scope = OAuth2Helper.
        getScopeAndAuthServer(service).get(0);
        List<String> scopes = Arrays.asList(scope);
        String authServer = OAuth2Helper.
                getScopeAndAuthServer(service).get(1);
        if (authServer.equals(
        OAuth2Helper.GOOGLE_AUTHSERVER))
```

```
        {
                return GoogleHelper.getAccessToken(username,
                scopes, authServer, REDIRECT_URI, CLIENT_ID,
                CLIENT_SECRET, keyRing, keyRingFile);
        }
        else
        {
                return null;
        }
}
```

Listing 5.10: The method for retrieving an access token

If there is a keyring entry for the username-service pair the 'getAccessToken' method retrieves an access token by sending the refresh token to the authorization server. Finally, the method returns the access token as Java String object. In case where there is no keyring entry the method asks the end-user to authorize himself/herself on the authorization server and then exchanges the authorization code for refresh and access tokens. The method stores the refresh token in the keyring file and returns the access token as a Java String object. Listing 5.11 shows the method for retrieving an access token for a Google service.

For an untrusted third-party application to retrieve data from OAuth 2.0 protected server there is the 'getDataWithAccessToken' method which takes the URI of data and an access token as its parameters. The owner of the access token can revoke the access by calling the 'revokeToken' method with the access token as the parameter, or revoking the refresh token with the 'revokeAccess' method.

```
public static String getAccessToken(String username,
        List<String> scopes, String authServer,
        String redirectURI, String clientId,
        String clientSecret, KeyRing keyRing,
        String keyRingFile)
        throws IOException
{
        Credential credential = getCredential(username, scopes,
        authServer, redirectURI, clientId,
        clientSecret, keyRing, keyRingFile);
        return credential.getAccessToken();
}
```

Listing 5.11: The method for retrieving an access token for a Google service

At the moment, AccessController can retrieve access tokens only for Google's services. Granting access tokens of another service provider is implemented by adding a getAccessToken method in the helper class the service. The method simply uses the credential retrieval method of the class and returns the access token.

5.9 Exceptions and recovering

Every method of the AccessController interface throws one or two exceptions. The 'openKeyRing' method, for example, throws an IOException if the keyring file could

not be created or read. The exceptions of the 'getData' method are thrown by the specific methods (getBasicAuthProtectedData', getOAuth2ProtectedData', getLocalData' and 'getOpenData') which 'getData' calls, and therefore those exceptions are covered in the following paragraphs which describe the exceptions of these specific methods.

The 'getBasicAuthProtectedData' method throws a MalformedURLException if the URL of the folder is malformed. The method throws an IOException if a connection to the URL cannot be opened, reading the user's input fails or storing the password in the keyring file fails. In case where the stored password or the password typed by the user is wrong, the method catches an IOException, deletes the stored password (if any) and calls the method itself what makes the method to ask the user for a new password.

The 'getOAuth2ProtectedData' method throws an IOException if the data or credential could not be obtained, the authorization code could not be exchanged for tokens or storing the password in the keyring file fails. If the refresh token in the keyring is revoked the method catches an IOException, deletes the refresh token in the keyring and starts an authorization flow.

The 'getLocalData' method throws a FileNotFoundException in case the file the client application requested was not found. A IOException is thrown by the method if the file could not be read.

The 'getOpenData' method throws a URISyntaxException if the syntax of the requested file's URI was not correct. The method throws an IOException if the connection to the URL of the file failed.

The methods 'revokeAccess' and 'revokeToken' throw an IOException in case they received no response from the revocation end-point. They also throw a URISyntaxException because they call the 'getOpenData' method.

The 'getAccessToken' method throws an IOException if the authorization code could not be exchanged for tokens.

The 'getDataWithAccessToken' method throws an IOException if the data could not be obtained.

6. CASE STUDIES

This chapter depicts the applications which were developed for testing the functionality of the AccessController module. Soil Sample Mapper is an example case of using AccessController in a trusted application. In section 6.2 there is described how the module can be used for granting access for untrusted applications. The last section of the chapter presents AccessControllerTask which is an Apache Ant task for access controlling and retrieving data.

6.1 Soil Sample Mapper

Soil Sample Mapper (SSM) is an application for visualizing soil sample data on maps of Google Maps service. The application was developed exclusively for testing the functionality of the AccessController module and it is not intended to be used in actual researches. SSM uses the AccessController module to retrieve XML files soil sample data and maps from different resource servers.

In the example use case SSM produces maps from two researches. The data of the other research, research1.xml, is located in Google Drive service and the data of the other research, research2.xml in a HTTP server's folder which is protected with HTTP Basic Authentication. SSM also includes information about the soil texture of samples. The information is retrieved from the site www.dbpedia.org which is an open data source. Google Maps and Google Drive are protected with OAuth 2.0. The function of SSM is demonstrated in the figure 6.1.

For the example use case, a Google account, with username 'soilsamplemapper', was created. In addition, a new user with username 'SSM' was created in the HTTP server. In the example use case there are no credentials for either Google's services or HTTP Basic Authentication stored in the keyring file.

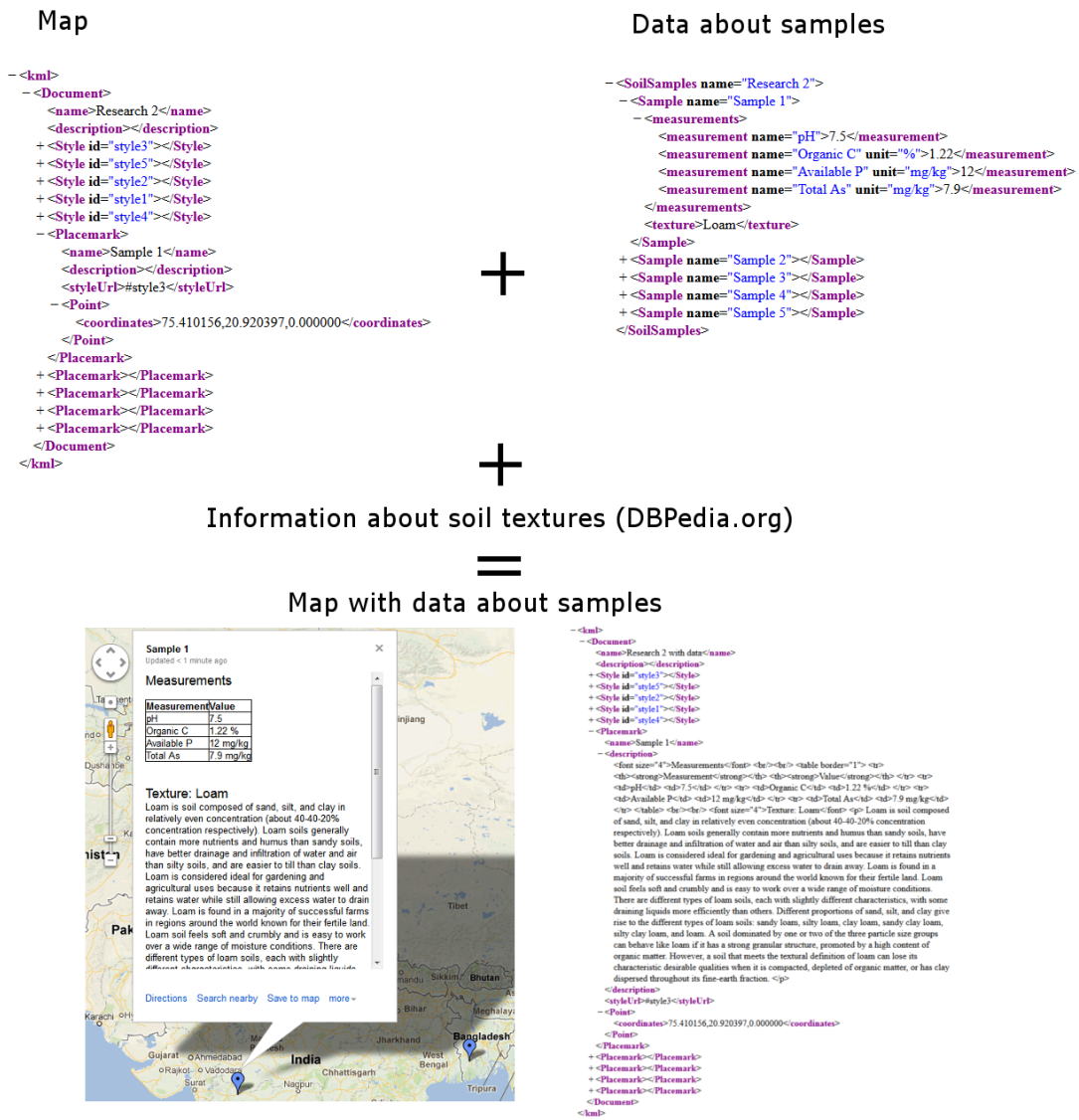

Figure 6.1: The function of Soil Sample Mapper

SSM retrieves its soil sample data from Google Drive and a HTTP Basic Authentication protected server. However, AccessController provides options to get the data also from the local file system or an open online site. These types of resource locations were not selected to implement into SSM because they do not offer any new testing value and SSM is supposed to be a brief application.

6.1.1 Structure of the data transmission

Figure 6.2 depicts the parties of SSM's data transmission and figure 6.3 details the phases of the transmission. Listing 6.1 shows the program code of Soil Sample Mapper.

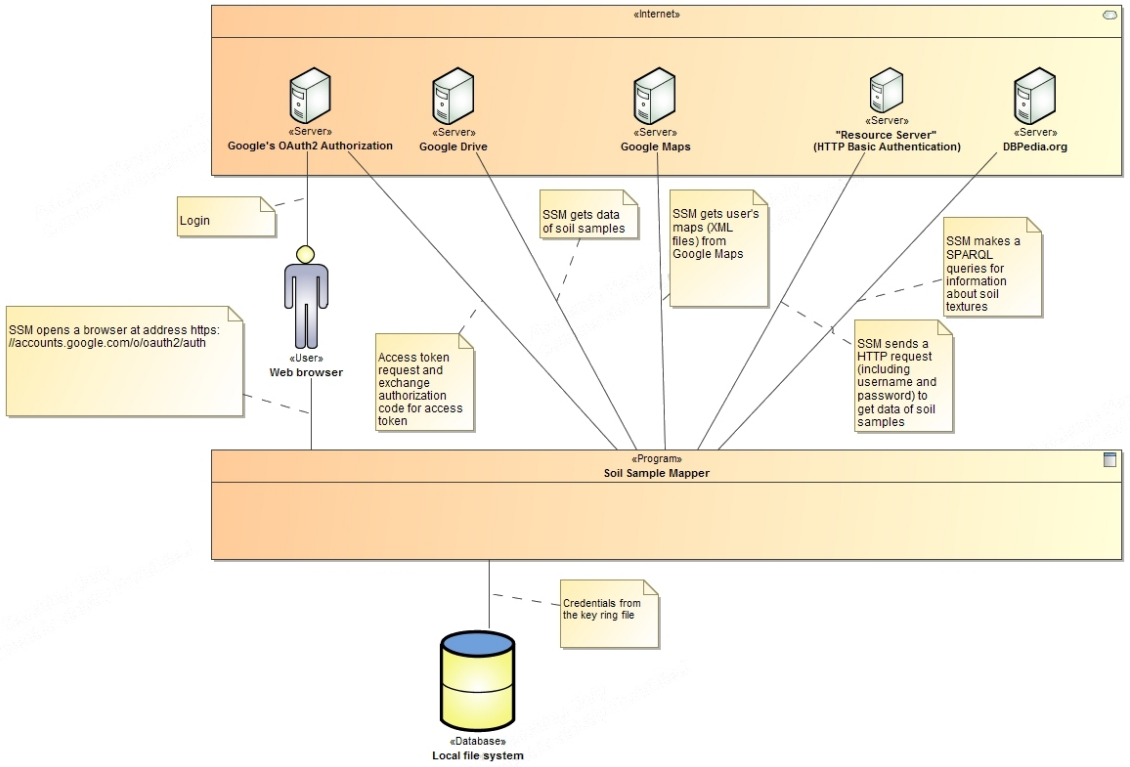

Figure 6.2: The parties involved in SSM's data transmission.

First SSM opens an access to the keyring file by calling the 'openKeyRing' method of AccessController. SSM requests for the 'research1.xml' file in Google Drive with the method call 'getData(''soilsamplemapper", ''GoogleDrive/file=research1.xml")' where the first parameter is the username of the Google account and the second the URI of the data. AccessController resolves from the URI that the requested data is in Drive service and therefore tries to retrieve a refresh token from the keyring file with the method call 'getPassword(''soilsamplemapper", GoogleDriveScope)'. Because there is no refresh token stored AccessController asks the end-user to authorize himself/herself on the Google's authorization server.

After a successful login the user types the authorization code and AccessController exchanges the code for a refresh token and access token. Then AccessController retrieves the file, stores the refresh token in the keyring file and returns the file to SSM.

Next SSM requests AccessController for the map file by making the method call 'getData(''soilsamplemapper", ''GoogleMaps/map=research1")'. Again, there is no refresh token and AccessController retrieves and stores tokens in an identical way as in the case of Drive. The XML data of the 'research1' map is returned to SSM.

```
AccessController ac = new Access();
ac.openKeyRing("masterPassword", "keyRing.txt");

String research1Data = ac.getData("soilsamplemapper",
```

```
"GoogleDrive/file=research1.xml");

String map = ac.getData("soilsamplemapper",
"GoogleMaps/map=research1");

MapMaker.makeMap(map, research1Data, "research1Map.kml", ac);

String research2Data = ac.getData("test",
"http://www.cs.tut.fi/~mustonep/SoilSampleMapper/research2.xml");

String map2 = ac.getData("soilsamplemapper",
"GoogleMaps/map=research2");

MapMaker.makeMap(map2, research2Data, "research2Map.kml", ac);
```

Listing 6.1: SoilSampleMapper class

In the next phase SSM retrieves information about the soil texture (e.g. silt or loam) of the samples. This is done incrementally in a loop for each sample. The data for a sample is retrieved with the method call 'getData(null,URLForSoilTexture)' where the value of the username parameter is 'null' because is no authentication in DBPedia.org's server. The URI parameter includes the URL of DBPedia.org's server and a SPARQL query for data of a soil texture. AccessController retrieves the soil texture data and returns the data to SSM. This phase is done in the MapMaker class. Listing 6.2 is a sample code from the MapMaker class and it shows how the textures are retrieved.

```
Document research = DocumentHelper.parseText(researchData);

List<Element> textureElements = research.selectNodes("//texture");
List<String> textureInfos = new ArrayList<String>();

for (Element textureElement : textureElements)
{
        String texture = textureElement.getStringValue();
        String query1 = "SELECT ?abstract " +
        "WHERE { " +
        "{ <http://dbpedia.org/resource/" +
        texture + "> " +
        "<http://dbpedia.org/ontology/abstract> ?abstract . " +
        "FILTER langMatches( lang(?abstract), 'en') }" +
        "}";

        String URI = "http://dbpedia.org/sparql?" +
        "default-graph-uri=http://dbpedia.org" +
        "&format=application/rdf+xml"+ "&query=" +
        query1 + "&timeout=0&debug=on";

        Document textureDoc = DocumentHelper.parseText(
        ac.getData(null, URI));
        String textureInfo = textureDoc.
        selectSingleNode("//sparql").
        getStringValue();

        textureInfos.add(textureInfo);
}
```

Listing 6.2: Getting information about soil textures

In the final phase of creating a 'Soil Sample Map' Soil Sample Mapper adds the data of soil samples (from research1.xml) and information about soil textures (from DBPedia.org) into the placemark elements of the 'research1' map. The result map is stored in the local file system.

In the example use case the second file of soil sample data is stored in the folder of HTTP server and the folder is protected with HTTP Basic Authentication. SSM calls AccessController's 'getData' method with the username parameter 'SSM' and the URI parameter which is a HTTP URL to the file in the folder of the HTTP server. AccessController resolvers that the folder is Basic Authentication protected and tries to get a password from the keyring file. Because there is no password stored AccessController asks the user to type the password. If the password was correct AccessController retrieves the 'research2.xml' file, stores the password and returns the file to SSM.

Next, SSM requests for the second map in the same way as with the first map. This time, however, there is already a refresh token for Google Maps stored in the keyring file, and hence no authorization is needed. AccessController obtains an access token by sending the refresh token to the authorization server. The 'research2' is retrieved and returned to SSM by AccessController.

The process of retrieving information about soil textures and creating ‘Soil Sample Maps’ is identical with the first research (research1).

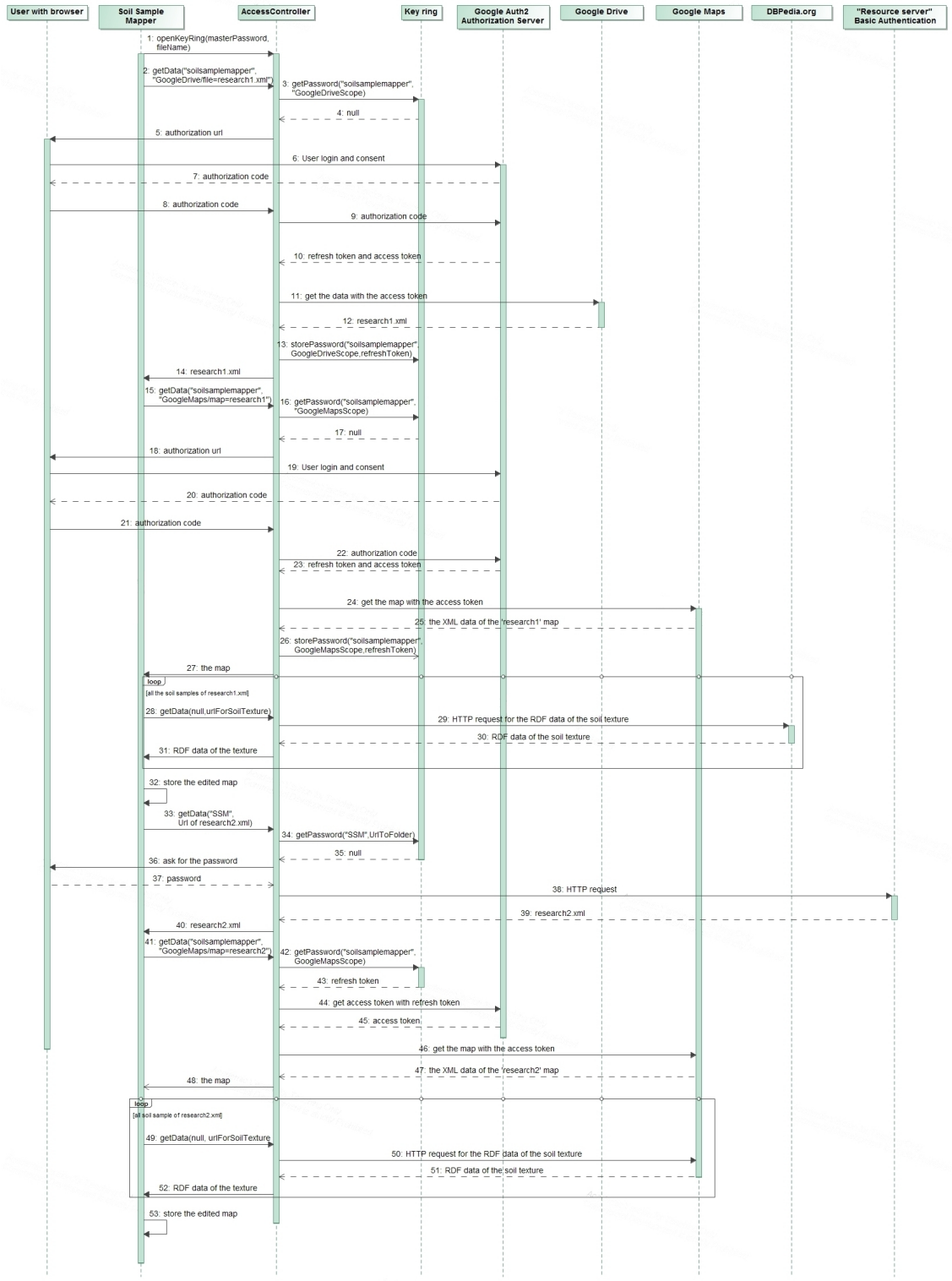

Figure 6.3: Phases of data transmission between SSM, servers on the Internet and the local file system.

More soil sample data and maps could be processed in the same application. If

the soil sample data is retrieved from the same Google Drive account, in this case from the account of user soilsamplemapper, the refresh token in the keyring will be used and no authorization is needed. This same situation concerns also retrieving maps from Google Maps service.

6.1.2 Technology stack

The figure 6.4 shows the technology stack of Soil Sample Mapper application. SSM retrieves data from resource servers with the AccessController module. SSM processes XML data with the Document and Element classes of Dom4j package. The FileWriter and File classes are used for writing the result maps in the local file system. Keyring files are stored in the local file system. AccessController uses HTTPS or HTTP as its connection protocol.

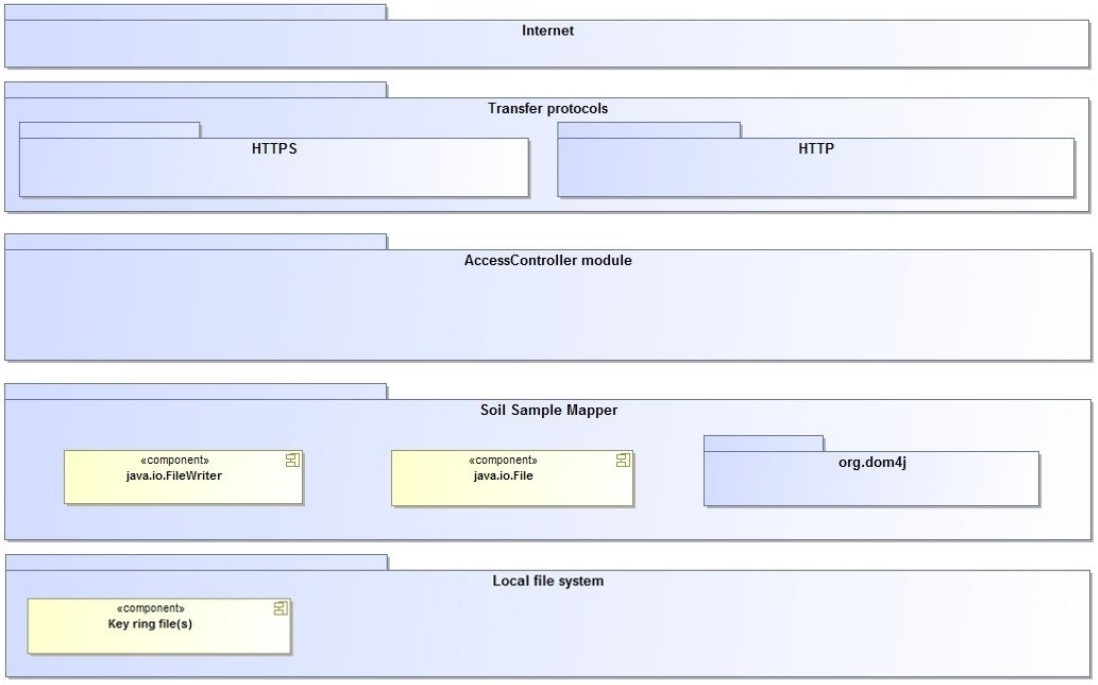

Figure 6.4: Technology stack of Soil Sample Mapper.

6.1.3 Tool and device environment

The IDE used in the development of the application was Eclipse Classic 4.2.1, release Juno (4.2) SR1. The version of Java Development Kit was 7 Update 11 for Windows x64. The application was developed on Windows 7 Home Premium (64-bit) operation system.

6.2 Untrusted application

AccessController's feature of granting access tokens was tested by creating an application which is created (or run) by the owner of the resources and grants access tokens, and an "untrusted" application which uses the granted tokens. In other words, the feature can be used in case where the owner of the resources does not want share his/her key ring file and/or login credentials with the untrusted application. Instead of sharing that information, the owner grants a temporary access by giving access tokens to the untrusted application.

The granting application obtains the access tokens by calling the getAccessToken method of AccessController as described in listing 6.3. The owner can define to which service (specified by the scope of token) the token grants access to. The owner is not able to decide the expiring time of tokens but that depends on the configuration of the authorization server.

```
AccessController owner = new Access();
owner.openKeyRing("keyRing.txt", "masterPassword");
String tokenDrive = owner.getAccessToken("soilsamplemapper", "GoogleDrive");
String tokenMaps = owner.getAccessToken("soilsamplemapper", "GoogleMaps");
```

Listing 6.3: The application that grants the access tokens

Listing 6.4 shows how an untrusted application can use AccessController's getDataWithAccessToken method for retrieving data with access tokens. The application can retrieve unlimited amount of data during the life-time of the token.

```
AccessController ac = new Access();
String tokenDrive = "ya29.AHES6ZQJSU3M0MDE0-p5s38-S5Mr9NuGbEfzs2__o1YpLhM";
String tokenMaps = "ya29.AHES6ZQv79sjLLmEBzLx98L4503NO4SOEkFrrCf0FOAQ_7A";
String data = ac.getDataWithAccessToken("GoogleDrive/file=research1.xml", tokenDrive);
String map = ac.getDataWithAccessToken("GoogleMaps/map=research1", tokenMaps);
```

Listing 6.4: An untrusted application which uses access tokens for retrieving data

After granting the access tokens the owner can revoke the tokens with AccessController's revokeToken method. In the case of listing 6.3, the owner could make a call 'owner.revokeToken("ya29.AHES6ZQJSU3M0MDE0-p5s38-S5Mr9NuGbEfzs2__ o1YpLhM")'. However, neither the AccessController module nor the authorization server cannot provide a feature to observe if the untrusted has already used the token.

Neither using access tokens nor revoking them requires any client identification. Therefore, a malicious third-party application or user, which has hijacked an access token, can retrieve data with the token or revoke it.

6.3 Apache Ant task

An Apache Ant task, called AccessControllerTask, makes it possible to use the AccessController module in Ant pipelines. The task was packed to a jar archive

with Eclipse's Fat Jar Plug-In. In addition to the Java file, the archive also includes the jar archives of the AccessController module and ant library's version 1.7.1.

6.3.1 Example pipeline

Listing 6.5 illustrates an example pipeline of using AccessControllerTask. In the first target's taskdef task the name of the task is defined to be accessController. In the second target the defined task is used to retrieve data. The input tasks asks the user for the name and master password of the keyring file. The next element, accessController, includes four inner elements, called accessResource. The first inner element retrieves open data and therefore it does not have a username attribute. The retrieved data is stored into the location specified in the outputfile attribute. The second inner element gets data from a server protected with HTTP Basic Basic Authentication. The user is asked for a password for the server if the password does not exist in the keyring file. The next two elements retrieve data from servers protected with OAuth 2.0. If there is no access tokens for the servers in the keyring file the user is prompted to authorize himself/herself in the authorization server.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="acTest" basedir="." default="access">

  <target name="definition" description="Define AccessControllerTask" >
    <taskdef name="accessController"
      classname="AccessControllerTask"
      classpath="accessControllerTask.jar"/>
  </target>

  <target name="access" description="Access some resources"
    depends="definition">
    <input
          message="Please enter the name of the keyring file:"
          addproperty="keyring"
    />
        <input
          message="Please enter the master password for the keyring file:"
          addproperty="masterpassword"
        >
        <handler
          classname="org.apache.tools.ant.input.SecureInputHandler" />
        </input>

    <accessController keyRingFile="${keyring}"
      masterPassword="${masterpassword}">
      <accessResource resourceURI="http://dbpedia.org/data/Silt"
        outputFile="test1.xml"/>

      <accessResource username="SSM"
        resourceURI=
          "http://www.cs.tut.fi/~mustonep/SoilSampleMapper/research2.xml"
        outputFile="test2.xml" />

      <accessResource username="soilsamplemapper"
```

```
        resourceURI="GoogleMaps/map=research1"
        outputFile="test3.xml" />

      <accessResource username="soilsamplemapper"
        resourceURI="GoogleDrive/file=research1.xml"
        outputFile="test4.xml" />
      </accessController>
  </target>
</project>
```

Listing 6.5: Example of using AccessControllerTask

As using AccessController in a Java application, also in an Ant pipeline AccessController uses passwords or refresh tokens in the keyring for retrieving data from servers from which there was previously retrieved data. Naturally, the keyring file is opened for the particular task and it cannot be accessed in other tasks without entering a valid master password.

6.3.2 Java class of the Ant task

A Java class, called AccessControllerTask.java, implements the Apache Ant task. When an AccessControllerTask is executed in a pipeline, the processor calls the execute method of the class. At the beginning of the method if the user entered the name and master password of the keyring file, the keyring file is opened by calling the openKeyRing method of AccessController. The inner accessResource elements are processed in a loop which is presented in listing 6.6. If the value of the revokeAccess attribute of AccessControllerTask element is true then every access is revoked in a loop.

```
for(Iterator<AccessResource> it=accesses_.iterator(); it.hasNext();)
{
        AccessResource access = (AccessResource)it.next();
        String username = access.getUsername();
        String URI = access.getResourceURI();
        String token = access.getAccessToken();

        try
        {
                String data = null;
                PrintWriter out = new PrintWriter(access.getOutputFile());
                if ((token == null || token == "") && URI != null && URI != "")
                {
                        data = ac_.getData(username, URI);
                }
                else if (URI != null && URI != "")
                {
                        ac_.getDataWithAccessToken(URI, token);
                }
                else
                {
                        out.close();
                        throw new BuildException("URI is a required attribute");
                }
```

```
                out.write(data);
                out.close();
                log("Retrieved the resource " + URI);
        } catch (IOException e) {
                e.printStackTrace();
        } catch (URISyntaxException e) {
                e.printStackTrace();
        }
}
```

Listing 6.6: The loop which processes the inner elements

The inner elements are implemented as a public class called AccessResource. When an inner element is processed, a new instance of the class is created by calling the createAccessResource method (stated in listing 6.7) of the AccessControllerTask class. In the method the new instance is added to accesses_ list.

```
public AccessResource createAccessResource()
{
        AccessResource access = new AccessResource();
        accesses_.add(access);
        return access;
}
```

Listing 6.7: The method for creating an instance of AccessResource

New inner elements can be created by adding a new class inside the AccessControllerTask class and implementing a 'create' method, such as createAccessResource. New attributes for accessControllerTask element or inner elements can be implemented by adding a private variable (e.g. String username_) and the setter and getter methods (e.g. setUsername and getUsername) for the private variable.

7. ANALYSIS

This chapter presents the results of the thesis and summarizes the methods applied to obtain these results.

7.1 Methods

Two different methods were used to evaluate the AccessController module; the case studies and the evaluation framework and criteria. The case studies covered using AccessController in a trusted application in which the user trusts his/her refresh tokens and passwords for storing, untrusted application which uses access tokens granted by the owner of the resource for retrieving, and Apache Ant task. The evaluation framework and criteria were mainly constructed from reviews of SSO systems and password managers.

7.2 Results

This chapter answers to the evaluation criteria presented in the list in the section 4.4. The criteria are referred in parentheses, i.e. ”(criterion 6)”.

The most significant result of this thesis is that the access to several protected resources can be controlled with the following exceptions. Firstly, the access restrictions to the resource or an end-user's password for either an OAuth 2.0 or HTTP Basic Authentication protected server cannot be controlled via the client program or the AccessController module. Secondly, an access to OAuth 2.0 protected resource cannot be granted for a scope under authorization server which is different than the server user logged in.

AccessController offers an almost analogical access to the resource regardless whether the resource server is protected with OAuth 2.0 or HTTP Basic Authentication, or the resource is open data. The only difference between OAuth resources and the other kind of resources is that OAuth resources are unaccessible via HTTP URLs while the other resources are accessed via URLs. This is because the URLs of OAuth 2.0 protected resources include an identification code which cannot be used by the end-user because of the matters of usability.

Passwords and refresh tokens are hashed with SHA-1 algorithm and stored in a keyring file which is created with Java's KeyStore class and secured with a master password. An entry of a keyring file includes a username, password or refresh token,

and a folder of a HTTP server or a scope of a OAuth 2.0 protected resource server.

7.2.1 SSO features

AccessController creates a new keyring entry for every new scope (or resource server) of the particular user. Hence, the AccessController module grants only the minimum access to the resources of the user. However, AccessController does not utilize the single sign-on feature of an authorization server. On the other hand, AccessController itself is a SSO system because the master password of a keyring file may grant access to multiple servers.

The AccessController module supports the application first scenario of SSO systems: it first checks if there is a password or refresh token for the specific username and server stored in the keyring file. If there is not then AccessController asks the end-user to login into an OAuth 2.0 authorization server or enter a password for HTTP Basic Authentication. The module does not offer an option to store credentials into a keyring file before trying to access the resource. Therefore, the module does not support the login first scenario. (criterion 1)

The AccessController module can logout a client application from both OAuth 2.0 and HTTP Basic Authentication protected resource servers. The logout method of AccessController deletes the entry in the keyring file and in case of a refresh token it also revokes the token. However, an end-user's signing off on an authorization server does not affect either the lifetime or validity of tokens. If the keyring file is deleted or the master password lost a revocation by a portal of an authorization server is the only way to revoke the refresh tokens. (criterion 2)

The situation where an end-user revokes a refresh token of a client application in an online portal of an authorization server is also considered in the AccessController module. The module detects a revoked token when retrieving data from the resource server, and asks the end-user to login on the authorization server.

7.2.2 Password manager features

The AccessController module can be used as a local (desktop) password manager. One of its benefits as a password manager is that an end-user can control where the credentials are stored.

Credentials are secured with a master password (criterion 4) but there is no option to use a key file instead of a master password or combine the key file and password methods (criterion 5). If the master password (and encryption of the credentials) is strong enough the credentials stored with AccessController are secure. Disadvantage of AccessController is that only passwords and tokens are encrypted in a keyring file, and usernames, servers and scopes are unencrypted. Therefore,

username-server/scope combinations are unsecured information. In addition, because AccessController works locally anyone who has access to the computer (actually to the folder of the keyring file) is also able to read the keyring file. But because credentials are stored in a file the user has full control over the location of the credential store (criterion 8).

There are also some other disadvantages of desktop password managers compared to online managers: no credential synchronization between computer, accessiblity from any computer (criterion 7), or one time passwords or any other multi-factor authentication (criterion 6).

7.2.3 Other features

AccessController does not specify the type of connection between the client application and the authorization server or resource server but it depends on servers (criterion 3). However, the module supports both HTTP and HTTPS protocols.

The AccessController module stores only the scope of the resource server of the requested resource into the keyring file. Therefore, the module provides minimum access to resource servers (criterion 11).

AccessController has a feature by which an end-user can grant access tokens to an untrusted third-party application. The owner of the access token can control the access by either revoking the particular access token, or all the access tokens granted by a refresh token by revoking the refresh token. A remarkable disadvantage of this feature is that the owner cannot affect the lifetimes of the access tokens he/she grants (criterion 10). Furthermore, the owner is not able to control, or even monitor, how many times the third-party application access the resource with the access token.

8. CONCLUSION

The main goal of this thesis was to investigate how access to resource servers can be controlled in an application that utilizes both open and closed Linked Data. The thesis also aimed at finding out a way to store and manage credentials for resource servers. To answer the research question a module called AccessController was developed.

The AccessController module meets the most relevant requirements set to the module. It can be used in a trusted application for retrieving data and storing credentials. The module is also able to grant access tokens for an untrusted application and the untrusted application can use the module to retrieve data with these access tokens. It was very straightforward to use AccessController as library when implementing an Apache Ant task for access controlling.

The AccessController module supports only a few types of resource servers with closed data. However, this support can be extended by creating new helper classes. When creating a new helper class for a service which uses OAuth 2.0 for authorization, the new helper class will be called in the OAuth2Helper class. To retrieve data from a resource server protected with some other authorizarion technique, a completely new helper class must be created. A new Google service could be added into the GoogleHelper class and it could use GoogleHelper's method for obtaining credentials.

The most significant feature which AccessController is lacking, is controlling the expiring time of access tokens. However, this is caused by the functionality of OAuth 2.0. The feature can be implemented into AccessController when there is a suitable authorization technique available.

A possible solution to control the lifetimes of granted access tokens with the current version of OAuth 2.0, is a process which revokes the token after the time period decided by the granter of the token.

On the whole, the AccessController module responded well to the needs of this thesis and it can be further developed by adding new features.

REFERENCES

[1] Berners-Lee, T. 2009. Linked data [WWW]. [Accessed on 13.2.2013] Available at: http://www.w3.org/DesignIssues/LinkedData.html

[2] Berners-Lee, T., Fielding, R., Frystyk, H. 1996. Hypertext Transfer Protocol – HTTP/1.0 [WWW]. [Accessed on 20.2.2013] Available at: http://tools.ietf.org/html/rfc1945

[3] Bizer, C., Heath, T., Berners-Lee, T. Linked data - The Story So Far [PDF]. [Accessed on 14.2.2013] Available at: http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf

[4] Chadwick, D.W., Inman, G., Siu, K.W.S. & Ferdous, M.S. 2011. Leveraging Social Networks to Gain Access to Organisational Resources [PDF]. [Accessed on 14.10.2012] Available at: http://delivery.acm.org/10.1145/2050000/2046653/p43-chadwick.pdf

[5] Cobden, M., Black, J., Gibbins, N., Carr, L., & Shadbolt, N. 2011. A Research Agenda for Linked Closed Data (vision paper) [PDF]. [Accessed on 13.2.2013] Available at: http://eprints.soton.ac.uk/272711/3/position.pdf

[6] George, B. Sun Microsystems. 2005. KeyChain (SwingLabs API). [Accessed on 24.4.2013] Available at: http://download.java.net/javadesktop/swinglabs/releases/0.8/docs/api/org/jdesktop/swingx/auth/KeyChain.html

[7] Google Developers. 2012. Google OAuth Client Library for Java. OAuth2 [WWW]. [Accessed on 11.3.2012] Available at: http://code.google.com/p/google-oauth-java-client/wiki/OAuth2

[8] Google Developers. 2012. Using OAuth 2.0 for Installed Applications [WWW]. [Accessed on 30.10.2012] Available at: https://developers.google.com/accounts/docs/OAuth2InstalledApp

[9] Haishi's Blog. 2012. A survey & comparison of SSO solutions [WWW]. [Accessed on 3.1.201] Available at: http://haishibai.blogspot.fi/2012/03/survey-comparison-of-sso-solutions.html

[10] Hardt, D. 2012. The OAuth 2.0 Authorization Framework draft-ietf-oauth-v2-31 [WWW]. [Accessed on 18.12.2012] Available at: http://tools.ietf.org/html/draft-ietf-oauth-v2-31

[11] Lodderstedt, T., Scurtescu, M., Dronia, S. 2012. Token Revocation (draft-ietf-oauth-revocation-00) [WWW]. [Accessed on 29.2.2013] Available at: http://tools.ietf.org/html/draft-ietf-oauth-revocation-00

[12] Nikokov, I. 2006. Web Single Sign On Systems [WWW]. [Accessed on 31.12.2012] Available at: http://www.cesnet.cz/doc/techzpravy/2006/web-sso/

[13] Pinola, M. 2012. Which Password Manager Is The Most Secure? [WWW]. [Accessed on 10.02.2013] Available at: http://lifehacker.com/5944969/which-password-manager-is-the-most-secure

[14] Richardson, L. 2010. Developers Enjoy Hypermedia, But May Resist Browser-Based OAuth Authorization [PDF]. [Accessed on 13.10.2012] Available at: http://delivery.acm.org/10.1145/1800000/1798377/p4-richardson.pdf

[15] Sun, S-T., Pospisil, E., Muslukhov, I., Dindar, N., Hawkey, K. & Beznosov, K. 2011. What Makes Users Refuse Web Sing Sign-On? An Empirical Investigation of OpenID [PDF]. [Accessed on 14.10.2012] Available at: http://delivery.acm.org/10.1145/2080000/2078833/a4-sun.pdf

[16] Wang, R., Chen, S. & Wang, X. 2012. Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services [PDF]. [Accessed on 15.10.2012] Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6234424