UNIVERSITY OF OSLO
Department of Informatics

A Semantic Approach for context-aware Authorization in Enterprise Systems

Master thesis

Martin Folkeseth

Autumn 2013

A Semantic Approach for context-aware Authorization in Enterprise Systems

Martin Folkeseth

Autumn 2013

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor **Josef Noll**, for his guidance throughout this thesis, for including me in his work and for encouraging me to continuously improve my skills. Josef has much experience which he is glad to share, and is also a person who takes student opinions seriously, and for that I am grateful.

I would also like to thank my fellow students for their opinions, feedback and good times both on- and off campus.

Finally, I would like to thank my family, friends and my good colleagues at Netlight for all their support and good times as well.

Abstract

This thesis deals with the privacy of the user profile in a context-aware situation. The mobile phone has become an identifier in the digital world, and can help us to get services tailored towards our needs in the given situation. Such a situation may be a service offer at the airport, train station, or any other locations. The approach in this thesis is to implement the core concept of such context-aware service provisioning for a travel scenario. The key issues of our approach is the business applicability to an enterprise environment, as well as establishing access to the user profile based on trust and link the user profile to context-aware services. We will use the context of the user and the situation of the context as the attributes contributing to establishing trust value, and implement service provisioning using enterprise frameworks and Microsoft OData. We will then review the approach and the implementation and compare it to other solutions on the market.

List of Abbreviations

PA Profile Attribute

SA Security Attribute

TA Trust Attribute

TAV Trust Attribute Value

CA Context Attribute

CAV Context Attribute Value

ETA Evaluated Trust Attribute

TL Trust Level

Contents

List of Figures

Chapter 1

Introduction and technology evaluation

1.1 Problem statement

Most people have a mobile phone, or some kind of digital device on their person at all times. With mobile computing becoming more important in our lives every day, it has also become an important part of our security. In one day, one might visit the bank, Facebook, Email, cloud storage and train time tables, just to mention a few. The mobile phone has in many ways become the *identifier* in our digital daily lives.

The mobile phone contains information about location, information about availability, travel plans, as well as activity and calendar. In order to create personalized services, the functionality may be enhanced by user preferences. To keep all information the phone and make them available on the phone for service providers, will require an huge amount of data transfer, reduce battery capacity, delay in service capability, as well as lack of backup of the information when the phone is lost or experience technical faults. Current trends in the industry shows that an approach where a user profile is created for each application is turned into a more common approach where only relevant parts of that profile are provided to a service provider for his/hers service or application.

The structure of the user profile and how to store it has not been standardized yet, but we assume that semantic technologies will help us to get structured representation of these data. Contextual information, being time and location, as well as activity and temperature, are refereed in many research papers as being the contributer to context-aware profiling, but little ha been done on the contextual user profile.

By combining all the different application-specific user profile into one user profile, the challenge of only providing relevant information to the service in charge, is one of the dominating challenges of the future. This information should be context sensitive and should reflect the demand for privacy of the user. An example of such context-aware profile is where a user comes to a certain location and he/she wants information on service offers. If the users are completely unknown in the place, they might only

be interested in getting recommendations for the further on travel, being the way to the next airport gate or bus stop, where as the users feel more comfortable, he might agree to get information on sushi breakfast.

The main challenges here is to build the users trust for context-aware situations and how to link this trust settings to the access of the user profile. In our approach we called up an attribute-based authorization, where the attributes contribute to access to certain parts of the user profile. One of the main challenges that we are addressing is to bring this into an enterprise service environment. Most of the enterprises user the Microsoft framework, thus a major point of this thesis is to apply the Microsoft framework to context-aware personalized provision.

1.2 Goals

The goal of this thesis is to address the challenges in the following areas:

- Presenting a scenario of context-aware personalized profile.
- Define a functional architecture, which takes into consideration privacy demands of the user and the context-aware service provisioning offers.
- Evaluate the enterprise framework for applicability to this functional architecture.
- Implement user profile, trust architecture, and context-aware authorization into an enterprise frameworks.
- Evaluate the approach of using an enterprise framework for context-aware and enterprise frameworks.

1.3 Approach

The approach follows the goal description, being described in chapter before. Where each and every bullet of the goals is tackled on its own. A scenario will be established in order to exemplify the challenges of context-aware profiling. The functional architecture is set up combining a scenario and a non-technical implementation description, taking into account a distributed computer profile and access to that user profile. Security and privacy is taking into account through the review of the state-of-the-art authorization models, leading to a context-aware authorization model for semantically structured data. Our approach of using enterprise framework, thus the state-of-the-art review and suggests framework components to implement the functional architecture. Through that we will be able to communicate user preferences by receiving context information and respond with service offers that are structured semantically.

The implementation of the core components will provide us with an answer to what degree the framework is able to supply fully working

authorization implementation. Based on the experience of this work, we will the conclude with recommendations on to which extend the enterprise framework can be used for context-aware authorization and personalized user provisioning.

1.4 Structure of thesis

Following the approaches described below, the thesis is described as follows; Chapter 2 creates an overview of the thesis, by introducing the scenario, establishing key requirements, as well as analyzing possible technologies and concepts for the implementation. Chapter 3 reflects on other researchers proposal for state-of-the-art authorization/authentication models, with main focus on context-awareness. Chapter 4 will expand expand the scenario to a detail description and make a selection of which parts of the scenario to be implemented, as well as present a implementation simulation with simulation data and expected results. Chapter 5 makes a selection of technologies to be used to create our framework for context-aware authorization for semantically structured data. Chapter 6 describes our implemented framework in detail. In chapter 7 we evaluate our approach of the thesis and implementation. Finally, chapter 8 will conclude on our results for creating a enterprise framework for context-aware and personalized services. Chapter 8 will also present recommendations for future work and provide additional information of how OData can be used in practice, in other scenarios.

Chapter 2

Expanding problem statement

2.1 Introduction

In this thesis we wish to combine context information and user profile information in order to create a context-aware user profile, and to provide location-aware services to the user. In order to have a privacy-aware solution, our basic assumption is that not all of the information of the user is going to be in the cloud, but some of the confidential information will be located on the mobile device. In order to have such distributed profile information, we need to look at available security mechanisms. Our main focus for such context-aware peronized information is on the access control. We aim at using context-aware access control authorization model for semantically structured data, will explain this in more detail later.

In this chapter we will expand the problem statement and will start by creating a detailed scenario in section 2.2, where we create a more detailed description of the user experience. In section 2.4 we will present the different components we view as essential in such a system. From the scenario and system overview we will reflect on what we see as key requirements, which will be covered in 2.3. Finally we will present concepts and technologies that, to the best of our knowledge, can be used to fullfill our scenario and key requirements.

In order to provide a broad understanding of the topic and the related technology aspects, we will start will a brief presentation of technologies and concepts that might be relevant. We will then establish key requirements and identify the core technologies wich we are going to use in our solution. These requirements and technologies will be described in section 2.5 and 2.6.

2.2 Scenario

In ordet to give an overview of the context-aware user provisioning we will create scenario, indicating the user experience when combining his/hers profile data with context information.

Our starting point of providing personalized services is the assumption that the user will have a user profile, typically in the cloud. Typical

information in this profile is information the user has stored to help the system to present information to the user that is considered relevant, such as personal preferences, travel schedule etc. Updates of the user profile are seen either through a direct interaction with the user or through automated algorithms or through recording user reactions to situation-aware context.

When a user walks into an arbitrary airport, he/she may want information relevant to their current situation. This person may have for instance arrived by plane or public transport, in other words he may be going to a hotel/home or on a flight. Our expectation is that the TV's, which are located throughout the airport, will not only display generic information, but also information being relevant for the specific user in the vicinity. This scenario assumes that users carry their mobile device at all times, which contains, or has the ability to obtain, contextual information. Such information is typically the UserId, time and location, but can also be information such as temperature, weather condition, the users travel plan or its further on destination. The phone will automatically connect to TVs, carrying a local hotspot, throughout the airport and send the context information to this hotspot. The backend system of the hotspod will then combine context information from the users phone, context information from itself to create a service request to the cloud, requesting access for parts of that users profile, which is relevant to the users current action. If for instance the user has sufficient time to eat before he/she can travel furhter, the local food service may ask the cloud for the users food preferences, relevant for restaurants within a geographical distance. The users food preferences, or other information in the users profile, such as information that may be used for payment or social networks, is protected by a security policy, specified by the user itself or agents acting on the users behalf. Access to the specific parts of the users profile needs to be authorized by the cloud for the requesting serivce to be able to retreive this information. Based on the users information retreived by the requesting service, available service offers and other open data public information, the TV-screen will then create personalized information to the users.

In our situation, an example may be provided for a user called Bob, arriving at Gardermoen Airport and are later on going to Los Angeles. There has been some flight delays, and Bob has got time to spare. For this example we will assume that Bob would like to be provided with information on how he may use his spare time, based on his personal preferences. But before Bob can be provided with useful information, there has to be some information about him in the cloud. Therefore, we have sub-divided Bob's scenario into two parts, where the first part is profile management, and the second part is the interaction with services surrounded by Bob. Both of these parts consists of several steps which we will cover in the following lists, and will be followed by a figure for graphical representation. We will start with profile management, assuming that Bob has access to his user profile via a front end, i.e mobile app or web site.

1. Bob accesses his user profile.

2. The front end fetches Bob's user profile from the cloud.
3. He creates a new attribute.
4. He attaches a policy to this attribute.
5. He saves the attribute, and the front end will update the profile in the cloud.

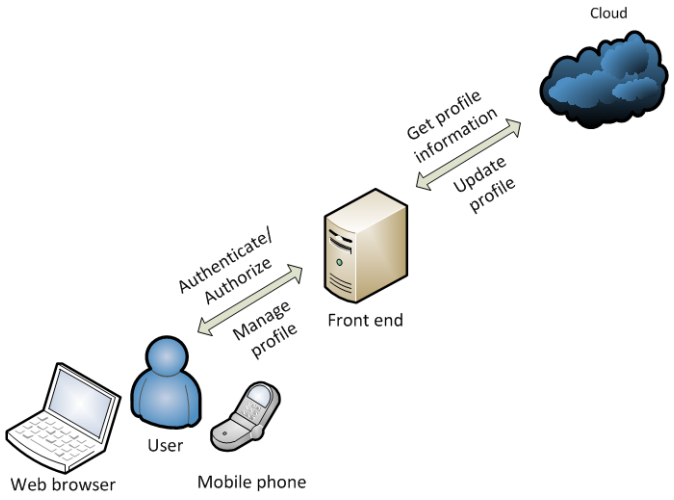

Figure 2.1: A vizual example of the process where a user manages his/hers profile information

In this figure we have chosen a stand-alone front-end to make it clear that there are several options available. The front-end can be an own web server, mobile app, part of the cloud itself or others.

When Bob arrives at Gardermoen Airport, the system will interact by following the below steps. We assume that his phone is within reach of a hotspot and the phone is ready to discover services.

1. His mobile phone detects a hotspot, which is locally attached to a TV.
2. The phone connects to the hotspot.
3. The phone gathers contextual information from it's sensors.
4. The phone sends the context information to the hotspot.
5. The hotspot receives context information from the phone.
6. The hotspot gathers context information from it's sensors.
7. The hotspot sends the phones and it's own context information to the service provider, or backend system.
8. The service provider receives the context information from the hotspot.

9. The service provider constructs a request, containing a query for the current users profile information and the context information.
10. The service provider sends the constructed request to the cloud, which contains the user profile information.
11. The cloud fetches the user profile.
12. The cloud authorizes access to the relevant information in the users profile, based on context information from the request.
13. The cloud constructs a response message, containing authorized parts of the users profile.
14. The cloud sends the response to the service provider.
15. The service provider receives the response from the cloud.
16. The service provider establishes contact with other public data stores.
17. The service provider fetches information from these public data stores based on what information it knows about the user, context and profile.
18. The service provider creates a response message to the hotspot, containing suggested activities for the user.
19. The hotspot publishes the suggested activity to the TV for the user to view.

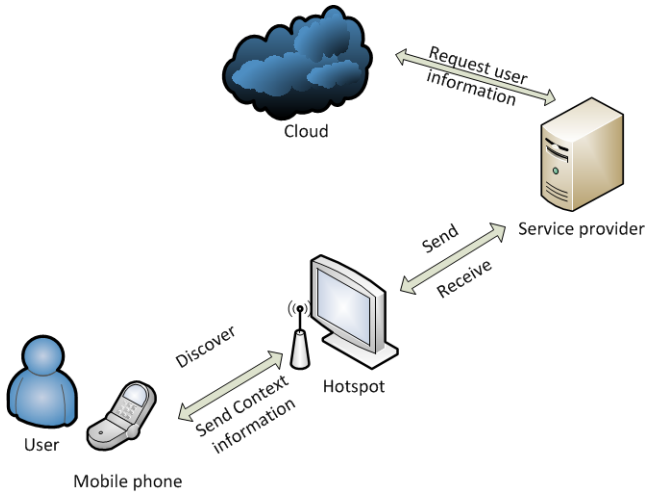

Figure 2.2: Shows the process for requesting a personalized service for a user

Ultimately, the system will consider the users time table, like how much time does he have before the flight leaves, has he already walked past the

security checkpoint, where is he located, etc. The system might also like consider what the users preferences are, like what kind of food does this person like, does he like shopping, is he looking for something in particular, and last but not least, what is this user *not* interested in.

2.3 Key requirements

In this section we will make a list of what we view as the key requirements based on the scenario. Like we did in the previous section, we have divided the scenario in two parts, the profile management part, followed by the interaction part, from our example, where Bob is at the airport. We will create a corresponding list to the above section steps and describe our view of requirements attached to that particular step. We will start with the profile management part.

1 A front end with a form of authentication, authorization and registration. We will also need a place to store the user profile.

2 The ability to communicate between services and the cloud, depending on implementation.

3 The ability to manage user profile from front-end in point 1.

4 The ability to create policies to restrict access to the users profile information.

5 View point 2.

Following is the key requirements for the interaction part:

1 Discovery for mobile devices.

2 Authentication certificate so that the mobile phone will be able to authenticate through the hotspot.

3 Mobile context sensors like GPS, clock, calendar and access other internal or external resources such as travel schedule.

4-5 Ability to establish communications between devices and trust.

6 Context sensors like described in 2.

7-11 Ability to establish communications between devices, like 4-5.

12 Security policies for user data and context-aware authorization.

13 The ability to create custom messages between services.

14-15 View point 4-5.

16-17 Ability to establish contact with external open data stores.

18 The ability to do automatic reasoning on user information.

19 The ability to publish graphical information on a TV-screen.

From the above lists there are many aspects to consider. As we stated in our problem statement, our main focus will be context-aware access control, where we focus on authorization. Considering this, we have selected the most important part of such a scenario in this thesis to be the context and user profile exchange between services and the cloud.

2.4 High level system view

Based on the key requirements resulting from the user scenario, we will now suggest the high-level view of our context-aware service provisioning system.

In this section we will evaluate the scenario and create an overview of a suggested system and its components. In Figure 2.3 we present

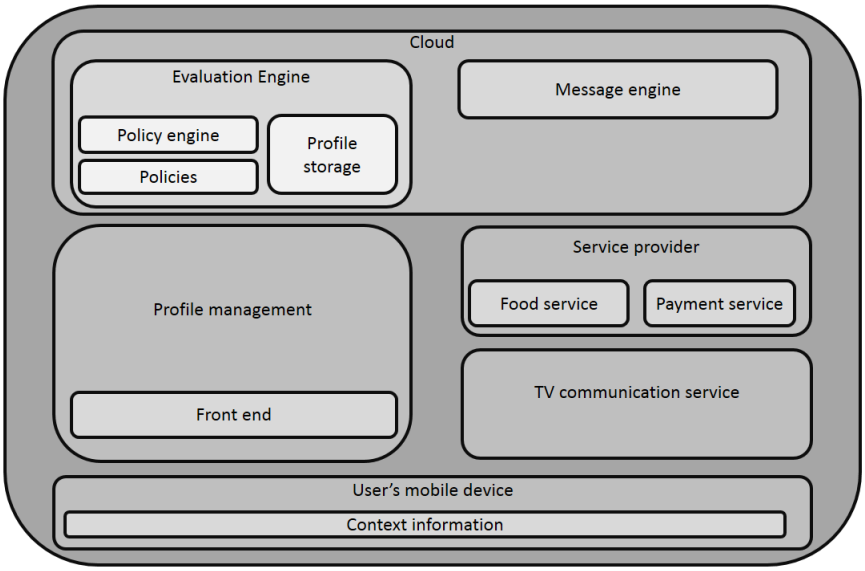

Figure 2.3: High level system overview with the five main components we consider bein the main participants in the scenario

our suggested system. It has four main components: *User's mobile device* *Profile management*, *TV communication service*, *Service provider* and the *Cloud*. These components are structured hierarchically in the order of how they communicate, meaning that if a user walks into the airport, the mobile phone connects to the TV hotspot via the TV communication service, which again forwards the request to a service depending on the users action, i.e. food service, which sends a request to the cloud.

The user's mobile device is the device acting on the users behalf, meaning that it is the users mobile phone that actually interacts with the system, but it is the user who moves between locations for the mobile device to be able to discover hotspots. The user is also the one who is in

charge of updating their profile in the cloud by interacting with the phone or other parties, such as a web site. The profile can also be updated by other agents or reasoning algorithms acting on the users behalf.

The profile management is the service used by the user to update his/hers profile. Depending on the implementation of the profile service, it can be present in the cloud or locally on the mobile phone. The front end part of the profile service is the user interface the person interacts with. The front end provides simple authentication and authorization in form of a username and password, or other mechanisms for the user to be able to access their profile. When a user wants to make changes to his/hers profile, the profile management exchanges information with the cloud to make the update.

TV communication service is the service that the users phone detects and interacts with when the phone is within range of the TV's local hotspot. It is used as the communicator between the users mobile device and the service the device want to use, depending on the users context. This service is also used to display relevant information on the TV screen onse a service has retrieved information from the users profile in the cloud.

The service provider layer of the model consists of available services the user can use. This is the component that requests information from the users profile based on the context, and exchanges information with other open data stores to be able to evaluate what information that may be relevant for the user in the current context.

The cloud is the main component is the most important part of the system. This component is in charge of storing and updating the user profiles and communication with other components, such as the profile management and services. When these services wants access to these user profiles, every request will be evaluated in respect to the policies users has attached to their information.

2.5 Access Control and data structure concept overview

To create a system where users can move between locations and the system can securely retrieve information from the user profile, we not only need components. We also need concepts for these components to use. In this section we will touch on different kinds of concepts. For our vision of implementation, not all of these concepts will be put to use, but will be mentioned in order to create as broad understanding of them as possible, and to be able to distinguish them from others. Our focus is on the security aspects of the service, thus we will focus on authentication, authorization, context information, as well as related access controls, being Role-Based Access, Attribute-Based Access or Context-Aware Based Access. We will then conclude the section with a short introduction to the Semantic Web and Linked Data, which we see as the two most promising approaches to generate the Context-Aware personalized services.

2.5.1 Authentication

Authentication is the process of identifying an individual. This is to ensure that the individual is who he/she claims to be, and is usually done with a username and password, but may also include third-party verification such as SMS, key generators and other methods. This is usually called third-party authentication. There are several other options to username and password, such as smart card and biometric authentication.

We can divide authentication methods into two categories *strong* authentication and *weak* authentication. Weak authentication is usually regular username and password, or just password, usually referred to as *something you know*[1]. However, passwords can also be divided into weak and strong password. Weak passwords are usually where we can choose our password freely without any patter restrictions, though it is becoming more and more common to use password complexity requirements [2]. The most common complexity requirement is by having a password which is 7 characters long and consist of upper- and lowercase letters, digits and alphanumeric characters.

Third-party authentication is usually associated with strong authentication. In recent years, more services are implementing this kinds of security, like Google, Microsoft, Blizzard entertainment, Citrix and BankID. The third-party itself can be you, *something you are*, biometrics, or *something you have*. A device third party can be a E-mail, SMS, security dongle or other types of One Time Password (OTP) tokens[3], where as biometrics can be facial recognition, fingerprint or eye-scanner. Even though biometrics is easy to use and can seem high-tech, devices are more often used with high security because biometrics has more faults, like bad algorithms resulting in authentication even if this is not the case, but when you loose your device, you will immediately know that there can be a security breach. There are also several other actions of security one can use to provide better security, companies often use address limitation, which means you will have to be within the company's network in order to be able to access data or use it's services. In general, by providing several steps for authentication, leads to better security.

When we talk about better security, there is also the argument of usability. We will not dig deep into this topic, but rather mention that this also needs consideration. If there is too much security, there will usually be lower usability. One action that has been frequently more used in order to solve such problems is Single-Sign On (SSO)[4]. To put it simple, SSO is a method of access control when upon authentication to a centralized authentications server, the user receives a security token which can be used to access other services attached to that authentication server.

[1] http:/ /www.novell.com/ documentation/ nmas31 /?page=/ documentation/ nmas31/ admin/ data/ a53s8fw.html, accessed October 2013

[2] http://technet.microsoft.com/en-us/library/cc786468(v=ws.10).aspx

[3] http://www.networkworld.com/newsletters/2007/0326id2.html, accessed October 2013

[4] http://en.wikipedia.org/wiki/Single_sign-on, accessed October 2013

In this thesis we will assume that there has been trust established between all service. This is a concept that is used so that services can communicate freely. This does not mean that they are free to impact each other as they see fit however. This is where authorization comes in, and which will be covered in the next section.

2.5.2 Authorization

Authorization follows after the procedure of authentication. This is the function of specifying how much access the user has to a specific resource. To be clear, Authentication verifies the users identity, where as authorization determines *what kind of access* or *how much access* we have to the resource. These resources can be anything, a web page, folder, even physical access to a part of a building. Through the process of authentication and authorization, the system may then access parts of the users profile. Following we present the three main categories of of authorization, namely Role-Based Access Control, Attribute-Based Access Control and Context-Aware Access Control.

2.5.3 Role-Based Access Control (RBAC)

Role Based Access Control (RBAC) defines access control by using a set of roles to determine access to a specific resource[23]. This is a widely used method of access control. A simple but yet useful example is a blog site. Here the viewers of the blog can be categorized as *user* or *guest*, but the editor of the blog can be categorized as a *administrator* or *writer*, where the administrator has write privileges and the user has read privileges. Such Role-Based access is a very static model and does not provide any grains of access control, which makes it harder to create more specialized security. This may also lead to scenarios where users may get more access than they actually need, because there may not exist a role for their purpose.

2.5.4 Attribute-Based Access Control (ABAC)

Attribute Based Access Control (ABAC) defines access control by the use of policies which combines attributes together [23]. Here we will use the terminology of subject and resource. When a subject wants to gain access to a resource or set of resources, the policy attached to that specific resource has requirements. An example of a subject can be a user, and application or a process, and a resource can be a web service, data structure or a system component, such as a folder. Policies can be defined as follows:

$$can_access(subject, resource) \leftarrow (UserID(subject) = ResourceOwner(resource))$$

The above ABAC policy states that a user can only access the specified resource if the user is the owner of the resource. Attribute Based Access Control can also be applied to Role Based Access Control with correctly constructed policies:

$$can\_access(subject, resource) \leftarrow (Role(subject) =' Manager')$$

$$\wedge(Name(resource) =' ApprovePurchase')$$

As shown in this example, if the user is a Manager he/she can access the ApprovePurchase web service. While policies in these examples are used to define access to a certain resource, where their meaning can be extended in defining complex relations of services in certain context being allowed to access specific parts of the user profile. In our case Bob is a restrictive person, so he will only give access to his profile where he is comfortable, example giving his home country, Norway.

2.5.5 Context-Aware Access Control (CAAC)

From our understanding, Context-Aware Access Control has several names, Context-Sensitive Access Control (CSAC)[22][19] and Context-Based Access Control (CBAC)[13] are some of them[8][9]. In order to try and distinguish these from one another, we found that CBAC is terminology mostly used for networking and that CAAS and CBAC are quite similar. Both of these concepts involves mobile computing and is used for access control by the use of contextual information. Our research shows that the term *Context-Awareness* are most widely used with mobile computing, we ended up on Context-Aware Access Control.

Our understanding of Context-Aware Access Control is that it is an extension of ABAC, where attributes are not only a set of static attributes, but dynamic attributes such as time and location. This is what we will refer to when we talk about CAAC in this thesis.

CAAC and ABAC provide both flexibility and good fine-grained access control, however, one needs to have a good model for such access control, as these rules and policies can become quite complex, and if so, can have bad scalability.

Having discussed authentication and authorization in access and context-aware in access, we will now address developments in Internet technology, allowing the access and the relation of data.

2.5.6 Context information

Mobile devices always knows something about itself or its environment. The device usually knows where it is, what time it is, and even the temperature in its location. This kind of information is usually called context information. Dey et al.[15][1]

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

In other words, context can be every piece of information we can gather about the environment that is actually useful to the application. However, Dey and Abowd proposed a set of four context types that would help to categorize and application as context-aware and non-context-aware. By their definition, a context-aware application contains information about Location, Identity, Time (date) and Activity. Location being the physical position of the entity (user), Identity being the identifier of the entity (UserID 3), Time being the time of interaction, and Activity being the trigger of the interaction.

Context information can be an important part of access control. From our example where Bob is at the airport and want useful information on what to do when he has spare time before his flight leaves, the back-end system may be able to view parts of his profile based on his location. However, context does not necessarily need to be associated with security. There are already a lot of mobile apps which uses contextual information to help the user, for example mobile apps used for public transportation which commonly uses both time and location.

How context information can be used in an authorization model will be covered in detail in chapter 3.

2.5.7 Semantic Web

The semantic web[5] is an extension to the current web that promotes common data formats. The web has expanded too quick in order for the technology to be able to follow. The current situation is that the web only consists of documents with content understandable by humans. The vision of the semantic web is to bring structure to meaningful content of web pages. The web pages would be structured by adding machine-readable tags to the human-readable information, then machines will be able to understand the information and reason on that information.

A web link can essentially point to anything, but in the semantic web, a link, or a URI[4] is used to identify an object. By using such a structure, we can distinguish objects with same name, but different purpose. Consider placing an order of business cards. You might want the cards delivered to you office address, but you want the regional office to take care of the bills (i.e delivery:address and billing:address).

So, what can semantics actually do for us? As mentioned, the semantic web consist of data and relations between these data. If the whole web were semantic, it could help us with a lot of things. The system could reason and check the consistency of data. Consider you did a lookup on a person you wanted to get in contact with. You may not know the persons first name, but you know that he/she works in the anthropology department at the National Museum of Natural History in Washington D.C, lives around the area of Dupont Circle and has a son studying History at Georgetown University. In the current web, you can't be sure that the person you find actually is the person you are looking for, but the semantic web can do reasoning upon all employees in that research department, check multiple resources, and compare the data you provided. Taking it one step further,

one could easily imagine a microwave checking the manufacturers site for optimal cooking parameters.

2.5.8 Linked data

Linked data was introduced by Tim Berners-Lee in 2006[6]. Berners-Lee created four rules, or expected behavior, for linked data to help the web grow:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL, where RDF* describes links between things on the web and SPARQL is used to query these).
4. Include links to other URIs. so that they can discover more things.

The example of linked data he describes is primarily meant for RDF, but from our knowledge of OData, the rules described by Tim Berners-Lee are also applicable. The rules of identifiers and relationships are also a core part of the OData protocol, which we will cover more in detail in 2.6.2.

2.6 Technologies and tools

In this section we will touch briefly on the technologies and tools that we consider may potentially be a part of our implementation. We will early on select a platform and focus on selection of technologies on this platform. However, we acknowledge that there are other platforms out there, so we choose to mention these briefly as well. Finally we will talk about a selection of technologies that are platform independent.

2.6.1 Platform

We knew early on that there is a variety of platforms that can potentially be used to solve the problem from section 1.1, create a context-aware authorization model for semantically structured data. We accept that there is too much work to learn everything about the world. We have chosen the Microsoft platform, because to the best of our knowledge, there has yet to be created an implementation from our scenario on this platform and this thesis is focused on a business perspective. Most corporations use this platform [5][6][30].This platform is currently used on 81.64% desktops and 45.8% servers worldwide[7]. With these numbers, we consider Microsoft to be a reasonable choice of platform.

[5] http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0, accessed August 2013

[6] http://www.forbes.com/sites/quickerbettertech/2013/05/06/why-most-businesses-will-keep-buying-microsoft, accessed August 2013

[7] as of August 2013

2.6.2 OData

OData is a application-level protocol for interacting with data via RESTful[8] web services [17]. It provides facilities for machine-readable descriptions, sets of data entities and their relations, filtering and querying over data, CRUD-operations, custom logic and vocabularies. OData is designed to work over a variety of data stores, but is mainly used for relational databases. URIs in a OData scenario may represent an object or a collection of objects.

The first version of OData was released in Febuary 2009[17] and have since then being put to use by huge communities on the web[9]. Among these are well known web-sites such as StackOverflow, eBay, TechEd and Netflix. The purpose of OData according to the OASIS OData Technical Committee:

> *There is a vast amount of data available today and data is now being collected and stored at a rate never seen before. Much, if not most, of this data however is locked into specific applications or formats and difficult to access or to integrate into new uses.*
> *- OASIS OData Technical Committee*

Entities in OData has a semi-semantic structure where each object has its unique URI, where semi-semantic means that the data is not stored semantically, but is rather presented semantically, by using a schema between the database and the application. Each entity has a set of data attached to it and also relations to other entities, navigation properties. Figure 2.4 show an example of an OData entity structured in XML. The <*id*>-tag shows that entities unique URI. The <*link*>-tags shows relationships to other entities, and the <*properties*>-tag shows data that describes this entity.

Atom

For publishing data, Microsoft created the OData Atom Format, which is an extention of the Atom Syndication Format[28] and the Atom Publishing Protocol (AtomPub)[18]. In short terms OData Atom Format extends these standards for representing and querying data. The published Atom data contains metadata, a description of the Entity Data Model (EDM) and Common Schema Definition Language (CSDL)[10].

Entity Framework

OData needs somewhere to store the data, and this protocol has out-of-the-box support for several relational databases, such as Microsoft SQL Server, Oracle and DB2. There is a possibility of supporting other data

[8] http://searchsoa.techtarget.com/definition/REST

[9] http://www.odata.org/ecosystem, accessed Febuary 2013

[10] http://www.odata.org/documentation/odata-v3-documentation/common-schema-definition-language-csdl/, accessed Febuary 2013

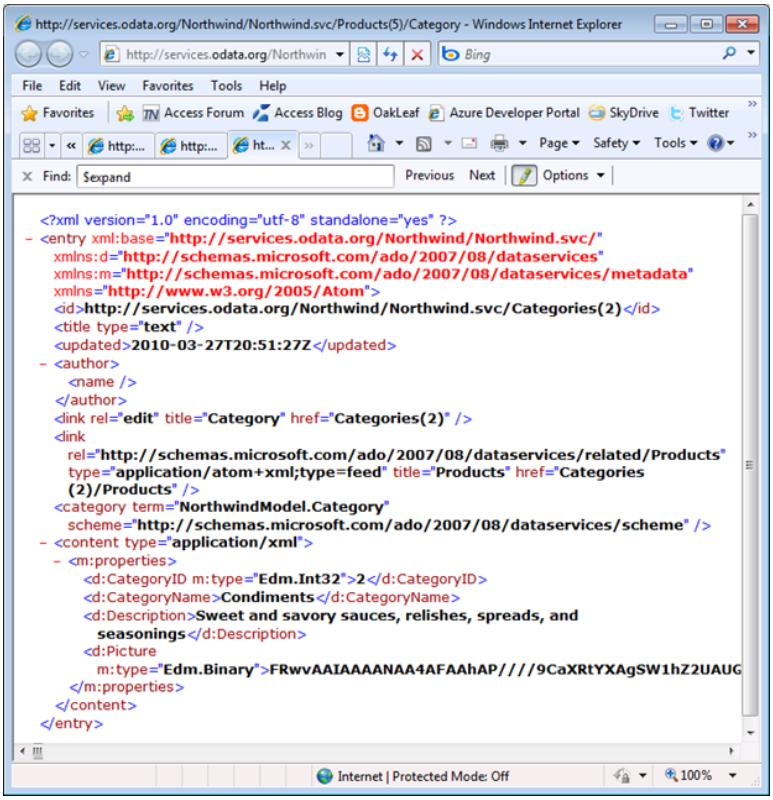

Figure 2.4: Example of OData response in AtomPub format

stores, which can be done by implementing an interface in the Entity Framework (EF). OData mainly uses EF as an interface to manage the data in a relational database. EF is a part of the Microsoft toolbox and is widely used in development processes using the .NET framework.

There are several options available if we want to work with Entity Framework: *Code First*, *Model First* or *Database First*. In Code First we create classes and generate the database from that class structure. In Model First, we can create a visual database diagram and generate classes and database from that diagram. The last option is Database First, where we want to use an existing database and generate a class structure from that database.

One thing that is particulary interesting about EF is that it also provides a scheme and connection string between the application and database. When you want to access data from the database, the returned data will be a collection of entity classes, which can be used directly in the application without worrying about writing queries, SQL injection and class mapping. EF takes care of that part. However, if desired, it is possible to map classes manually or use thirt-pary frameworks to perform the task.

LINQ

LINQ is a query language developed by Microsoft and provides a strongly typed query language which does type checking at compile time. In time

of writing, it is already implemented and used by a variety of objects withing the .NET framework. If your data source does not already support this query language, one can easily implement this by implementing an interface.

2.6.3 Semantic web technologies

RDF

Resource Description Framework (RDF) is used to express relations between objects, in RDF terminology called *triples*. These triples are expressed in XML and provides meaning to our data. If we for example have have two objects, "John" and "Maggie", we can give these two a relation (property) by expressing that John *is son of* Maggis or Maggis *has son* John. This gives us a way of describing the data processed by machienes. RDF uses URIs to encode information about a document, and by using URIs, we can be sure that these concepts are not just words but unique definitions that everyone can find on the web. Putting this comcept in practice, we can define that a field in a database "address" is of a type address using URIs rather than just the phrase address.

RDF does not have to be defined just as triple stores. This concept can be extended with named graphs [11]. Like triple stores, these named graphs are identified using an URI, and allows for extended description of the triple, such as context, created time, owner and other metadata.

SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) is a query language, standardized by W3C, designed to query RDF. SPARQL is recognized as one of the key technologies in semantic web.

Ontologies

How do we know whether an address in database 1 and address in database 2 is the same thing. This is where the Ontology comes in, which basically is a collection of information. Ontologies provides the ability to create classes, subclasses and the relations among them. We can assign properties to these classes and subclasses can inherit these properties. With the use of ontologies, we can express that *if John is child of Maggie, and Maggie is child of Robert, then John is grandchild of Robert*. The machine might not truly understand this information, but it can look at the relation between these objects and perform reasoning on that information. An example of an ontology language is OWL.

SWRL

Semantic Web Rule Language (SWRL) is a W3C standard that can be used to construct rules and logic. Rules consist of antecedent and consequent.

[11] http://www.w3.org/2004/03/trix/, accessed September 2013

This means that when a condition in antecedent is fulfilled, the condition specified in consequent must also hold. Here is an example: If *if John has parent Peter* and *Peter has brother James*, *James is uncle of John*.

2.6.4 GData

The Google Data Protocol[12] is a REST-inspire technology for reading, writing and modifying information on the web, and can be used in largely the same way as OData. This technology is used for most of Google's available applications. Google provides a large variety of APIs that can be used to create custom consumer applications against a Google account. However, Google does not provide a provider API so that programmers can create their own REST services based on GData technology.

2.6.5 XML

XML[11] is a W3C standarized markup language which defines a set of rules for encoding documents in a format that makes the data human-readable and machiene readable. XML follows a structured standard, it lets its users create arbitraty tags consisting of parameters and values.

XSD

XSD or XML Schema[33][10] is a W3C standatdized schema language for XML. XSD is used to express a set of rules in which an XML document that is attached to this specific schema must fullfill in order to be valid. By attaching schema to an XML file, some software can read this schema and give the developer feedback on what can be the next element in the structure.

2.6.6 Policy languages

Our research shows that there are two policy languages available: WS-Policy and XACML. WS-Policy is a policy language, and W3C standard, that allows web services to use XML to describe requirements for security, quality of service, data integrity and access control in a specific domain. A WS-Policy consists of a collection of policy alternatives, which again is a collection of policy assertions. A policy assertion describes a requirement, capability and other properties of a behavior. Microsoft and Microsoft Research maintains a framework, Web Services Enhancement, and tools, Samoa, to create such secure web services[13].

eXtensible Access Control Markup Language (XACML) is a access control policy language implemented in XML, which is maintained by OASIS[14]. These policies describes how to evaluate authorization requests according to the rules specified. XACML is primarily used as ABAC, where

[12]https://developers.google.com/gdata/docs/2.0/reference, accessed November 2013
[13]http://research.microsoft.com/en-us/projects/samoa/, accessed September 2013
[14]https://www.oasis-open.org/, accessed November 2013

the policies describes what attributes is needed in order to access a resource with the specified action. These XACML policies can consist of policy sets and policies, which defines subjects, resources and action. Subjects being the requirement to be fulfilled in order to gain access to the specified resources, where action specifies what access that is given by fulfilling the subject requirement to that resource. This is a policy language that is easy to understand and can create complex policies, but require a lot of markup to describe simple policies.

Chapter 3

Authorization models

3.1 Introduction

In this chapter we will mainly focus on state-of-the-art authorization models available on the market. Since the early millennium, there has been done extensive research on this topic and some research institutions have even created some frameworks for context-aware authorization models. In the early 2000, our research shows that even though this has been a hot topic, only recent years there has been a desire to put these concepts into practice. The following section will provide what we consider to be the most applicable authorization models to our research.

3.2 State-of-the-art authorization models

Costabello et al [14] proposed an CAAC-model for authorization, based on graph stores that consists of triples. Triples are composed of a subject, predicate and object. Such a triple describes how a subject is related to the object (Bob likes Sushi), usually in a RDF format [34][13]. This authorization model is based on a *S4AC* ontology, which is used for fine-grained access control policies for RDF data. In addition to this ontology, they use a vocabulary they call PRISSMA to model the context of a user, and SPARQL to communicate between the mobile device and actual data store. Between the mobile device and data store, there is a pluggable component called Access Control Manager (ACM).

Resources are organized as graphs, with RDF and is identified by URIs. Each graph describes owner and the resource itself. The main component of the access policy is the S4AC model, which defines the constraints that must be satisfied in order to be able to access a graph.

They define their context in three dimensions: User, Device and Environment. The user is the actual user of the system, the device is the device used by the user to communicate with the system, and the environment is the context information such as location, time and activity. The user (with help from the software on the mobile device) creates a SPARQL query, the mobile device accesses the context and sends this query into the system. The query is picked up by the ACM, which filters the

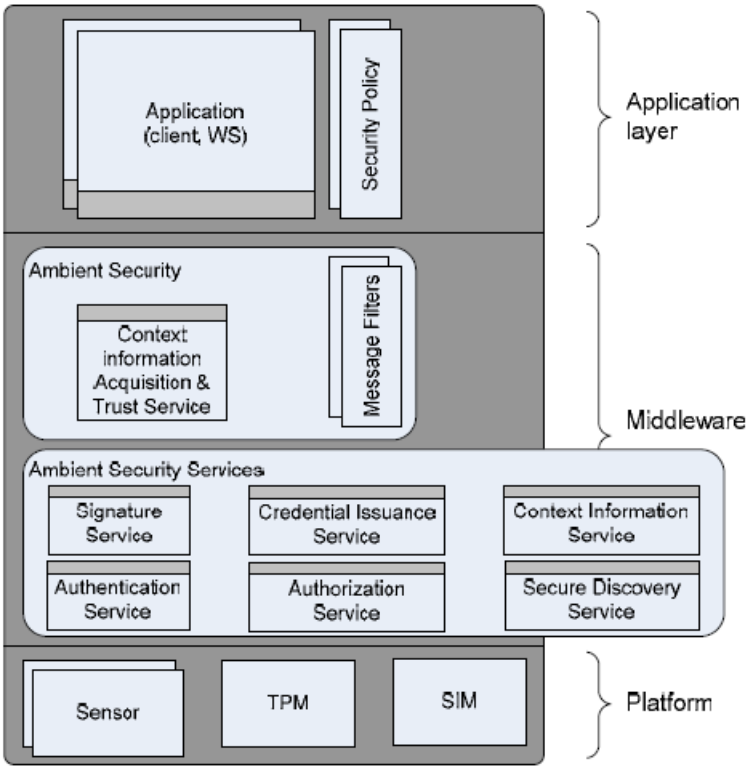

Figure 3.1: Overview of the MOSQUITO Context-Aware Access Control framework

query. The ACM first checks the integrity of the contextual information (future work) and selects the policies that is effected by the query and evaluates them. The evaluation returns the set of named graphs the user has access to and the query is only executed on these. After the execution of the query, the result is returned to the user.

Walter et al.[26] proposed the MOSQUITO framework that can be used for Context-Aware Access Control. In addition to authorization, it also covers security, encryption, data integrity and trust. Like Bhatti et al's[8] proposal, it is focused on a SOA based architecture, which means that it consists of services that can work together across different devices/computers/services in a network without human interaction. Here we will mostly talk about the authorization part of the framework, but will also mention briefly the other components of the framework in order to pinpoint where the authorization process fits in.

As we can see in Figure 3.1 consists of three layers. The first layer is the *Application layer*, which is located on the device. This layer contains two components, the application itself, and the *Security Policies*, which will be our main focus here. Objects are secured by these policies [19][12][9][34]. XACML is used to define access control rules that take context into account. SAML is used to define what context information that is required in order to validate the rule. The second layer is the *Middleware*, and is the most complicated layer. This layer is used to exchange and encrypt messages

(SOAP), check for data integrity, service discovery and to be designed tasks. An example of a task can be to pull a service for information every thirty minute, get current device location, etc.

The third layer, *Platform*, is the hardware layer. This is the interface that is used to get context information like location and temperature. Like mentioned above, we are going to go into detail about the authorization process of this framework and touch briefly the authentication process to create a overview. When a device comes into contact with a service it want to exchange information with (CRUD operations), they exchange metadata information to agree on the authentication. When they have agreed, the device sends the request containing action, credentials and context information. When the middleware has performed the trust evaluation, it forwards the request to the service.

When the back end receives the SOAP request, it requests the authorization service. The authorization services XACML engine is used to apply context-aware policies and returns a decision that is enforced by the policy. If the policy requires additional information, the service will send a similar request to its Trusted Third Party (TTP). Once the policy have all the information it needs to make a validation, it will grant access or deny access.

In 2004 Hu and Weaver [21] proposed an authorization scheme that is based on CAAC, implemented in .NET. The vision of this authorization scheme is to withhold access until it is needed. Therefore no authorization level (i.e Administrator) can give access to potentially everything. Their scheme is based on an extension to RBAC to consider context and is divided into three main components. These components are authentication engine, authorization engine and context service. The authentication engine has its traditional role with authenticating the user and provides the user with an identity and security token in return. The authorization service provides administrators with the possibility to create access rules based on the WSE framework's WS-Policy[9]. These policies, in addition to specific users or roles, also defines contextual information that needs to be validated in order to be able to access the resource specified in the policy. This provides, according to Hu and Weaver, flexible fine-grained access control that is evaluated at runtime. All resources or set of resources are bound by these policies. Finally, the context service manages a repository of all context definition. In practice, the authorization engine contacts the context service each time it needs to evaluate a policy that requires contextual information.

Toninelli et al. [35] introduces a CAAC framework that is designed for ad-hoc collaboration between mobile devices. They state that permissions based only identity/role is not sufficient in such spontaneous coalitions and such introduce the concept of context-aware access control. The argument is that in such scenarios, entities may need to share services with other unknown entities and maybe other entities which may not be sufficiently trustworthy. In addition to this, entities may need to change roles (from consumer to provider or vise versa), and new policies needs to be applied. They propose a semantic context-aware access control framework where one can both, define policies and where policies are able

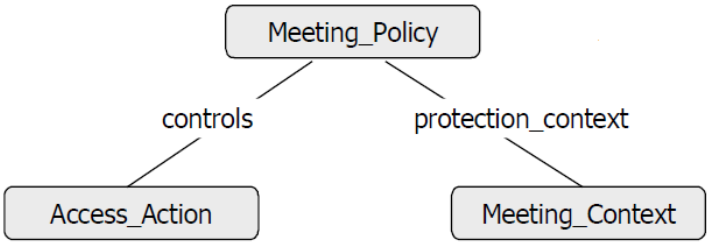

Figure 3.2: Model for OWL-based policy specification fod ad-hoc collaboration between mobile devices

to adapt to changing situations.

The access control model consists of an ontological approach, OWL[29], based on Description Logic (DL) for context/policy classification and Logic Programming (LP) for dynamic adaption of policies. Access control policies can be viewed as one-to-one associations between contexts and allowed action, referred to as *protections contexts:* ProtectionContext(context, allowedAction). Here the entities can only perform these actions when that context is active, as well as all entities that share the same context are allowed to operate on that same resource with the assigned action.

They define context as "all characterizing information that is considered relevant for access control". Toninelli et al. divide context into four elements: identity, time, location and action. However Toninello et al. makes a more clear distinction between the user and context information by dividing the context into two parts, the actor part and environment part. The actor part is the roles, identities or security credentials. The environment part contains all information about the environment, such as time, location or other available resources.

The policy model consists of three distinct phases: policy specification, policy refinement and policy evaluation. The policy specification is where administrators define the OWL-based policies, also called aggregation rules. These policies consists of ontological associations between actions and protection contexts as seen in Figure 3.2. The protection context has static or variable values. The variable values are set by the use of LP, which is returned to DL. These policies are only used as definitions and cannot be used in the real world. The policy refinement phase is where these aggregation rules gets instantiated and adapted to the particular state of the world. The evaluation phase is where protection contexts gets evaluated against current state of the context elements. Note that refinement and evaluation may be triggered by a resource context change, i.e, the resource changes location.

Samuel et al. [31], inspired by the Katrina Hurricane in 2004, present a theoretical mechanism for adaptive access control. Similar to [25][8][7][27], this model uses an extension of RBAC with context. When a user needs access, depending on if the system allows it, will be given a temporal role that is used to access a specific object. The goal of this system

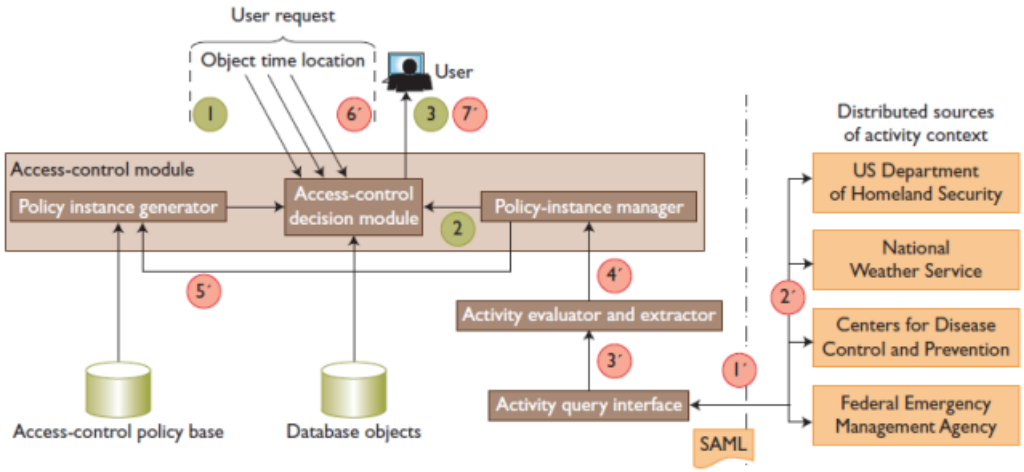

Figure 3.3: Access-Control policy adaption for critical situations

is to be used as a CAAC-model on a day-to-day basis, but is attached to distributed sources that contain information about crisis or potential crisis. Considering this, they purpose to divide policies into two categories. Normal Constraints (NC) are policies used on a day-to-day basis and Critical Constraints are used when there is recorded a crisis situation. As well as constraints, users are also split into two categories, Weakly Enforced Users (WEC) and Strongly Enforced Users (SEU).

Figure 3.3 shows the system as a whole. However, like mentioned above, there are two user cases: access control under normal circumstances (marked with green labels) and access control for crisis management (marked with red labels). The policies are created and stored in the Access-control policy base (ACPB). The Policy Instance Generator (PIG) creates an instance of the policy with the NC (default).

When a user wants to access an object, he/she sends a request to the Access-control decision module (ACDM). This request contain time, user location and reference to the object the user needs access to. The ADCM will evaluate access in conjunction with the Policy Instance Manger (PIM). If access is granted, the ADCM will return the object to the user.

Depending on the setup of the system, the Activity Query Interface (AQI) will get information from the distributed sources of activity context. If a crisis is detected, the AQI will forward the crisis context to the Activity Evaluator and Extractor (AEE). AEE sends context parameters to PIM, which requests a different policy instance from PIC, based on the activity context value and the CC in the policy. The PIG will then create a new instance of the policy and load it into the ACDM. When a user then sends a new request, access will be granted based on the users contextual information and class in the system (depending whether on the user is a WEU or SEU).

Chapter 4

Technology selection

4.1 Introduction

In this chapter we will look at what options we have from section 2.6 and make a selection out of the presented technologies. Since we early on selected Microsoft as our platform, some of these choices are straightforward, but where there are several options, especially in platform independent technologies, we will evaluate these side-by-side.

4.2 Selection of key technologies

4.2.1 Integrated Development Environment

Even though it is fairly easy to imagine the Integrated Development Environment (IDE) to use with .NET, there are actually several open source alternatives to Visual Studio. Fortunately for us the university provides us with educational licenses for Microsoft products, and since we have most experience with Visual Studio, the choice has landed on Visual Studio 2012 (VS2012). The actual choice of Visual Studio version could be 2008, 2010 and 2012, as the newest .NET framework is supported in all these version. At this point it is a matter of personal preference and we have chosen 2012 because that is what we are most familiar with at this point.

4.2.2 Web server

There are several web server softwares available, but we have considered the two most known ones, Internet Information Services (IIS) and Apace. Apache has modules that, to our knowledge, should be able to run .NET applications. However, the core of this system is based on Microsoft technologies and we can use IIS at no additional cost. Another reason for choosing IIS is that VS2012 comes with a web server called IIS Express, which is a development server that is almost identical to the regular IIS server, and therefore make our development process easier.

4.2.3 Data structure

Our research shows that there are several official an unofficial libraries that can be used to work with RDF-stores from the .NET platform. When we say unofficial we talk about libraries or plug-ins that is not available in Microsoft's NuGet packet manager. W3C has a list of available options [1].

The one most known to us is the dotNetRDF Library [2] which is an open source API for working with RDF, using the .Net framework. This library is currently being worked on and was last updated 6th of May 2013. This is also available through the NuGet packet manager in VS.

LinqToRdf is another Semantic Web framework for .NET. This framework provides functionality both for querying RDF databases using LINQ, This Semantic Web framework translates LINQ queries to SPARQL queries, which again is used to retrieve data from RDF files. It also provides a UML-style surface for creating RDF files. Unified Modeling Language (UML) is a modeling language used for object-oriented software systems [3].

RDFSharp[4] is hosted at CodePlex, which is a well known resource for .NET developers. This library offers a toolbox for working with RDF models and supports triplets, graph, among other things, as well as the ability co execute SPARQL queries on graphs and triple stores.

Open Anzo project[5] is featured with quad store and semantic web middleware platform, for creating RDF, OWL and SPARQL applications. Rather than being a REST protocol, the middlewarel provides a set of services for replication, notification, model, authentication, query, update. Anzo supports a Service Oriented Architecture (SOA), and currently provides APIs for .Net, Java and JavaScript. Additional storage architectures may be supported by implementing interfaces. A major architectual feature og this project is the support of offline use. This is a part of the client API and all graph changes are cached in the local replica automatically. We are only mentioning this project briefly, but it has a lot of other features, such as SPARQL query engine, reasoner and Command Line Interface (CLI). Cambridge Semantics Inc. offers the Anzo software suite that is a set of tools to be used together with Anzo.

We also found other tools for .Net programmers to work with RDF and OWL. The ones we found was Drive and ROWLEX. Unfortunately these projects are note in development anymore and has been taken down from the web.

We have chos OData for the data structure part in this implementation. Although this protocol is no longer maintained by Microsoft, it was originally a product created by them using the own frameworks such as .Net and Windows Communication Foundation (WCF). From our analysis on Semantic Web and Linked Data we think that OData fills these requirements. Every entity has its own distinct URI and data is linked

[1] http://www.w3.org/2001/sw/wiki/.Net, accessed October 2013
[2] http://www.dotnetrdf.org/, accessed October 2013
[3] http://en.wikipedia.org/wiki/Unified_Modeling_Language, accessed October 2013
[4] http://rdfsharp.codeplex.com/, accessed October 2013
[5] http://www.openanzo.org/projects/openanzo/wiki, accessed October 2013

between entities. With the mapping scheme between data storage and the software itself, makes the data easy to work with. OData also makes it easy to to change the client and server software. The OASIS Technical Committee provides several alternatives both for client and server. Some of the presented libraries for server, besides .NET, are well known commercial programming languages such as PHP, Node and Java, which are also available on the client side[6]. Databases such as MySQL and Azure data are also supported out-of-the-box. We are using Microsoft for this implementation because our focus lies on the enterprise, but there are also possibilities for free and open-source solution from a research perspective. OData also supports the JSON format out-of-the-box, which we find most interesting, if in the future a client wants to make asynchronous calls to our application with a JavaScript framework such as jQuery. From the frameworks mentioned above, there are several ways to use .Net to communicate with RDF stores, but in recent years, both Microsoft and W3C has presented experiments on how to make these two standards work togeather. We will cover this in more detail in section 8.2.2.

4.2.4 Data storage

To store our data we have chosen Microsoft SQL Server (MSSQL). MSSQL is also a part of VS2012 off-the-shelf and therefore also makes our development process easier. This technology and the ones mentioned above is not the primary part of the implementation and therefore we want to stay focused on the problem at hand, without running into to many technical difficulties.

4.2.5 Policy language

In the research phase of this thesis, there were two policy languages that was continuously used throughout the implementations, namely WS-Policy and XACML. From what we gathered WS-Policy was the Microsoft alternative, and from digging around the web we found that WS-Policy was part of a framework called Web Services Enhancement (WSE), maintained by Microsoft research. It is also shipped with a tool for Visual Studio to create policies. In the following table we will present our evaluation of these two policy languages:

Framework	Availability	Integration	Scalability	Usability
WS-Policy	X	X		X
XACML			X	

Our evaluation shows that WS-Policy is the best choice of policy language in our implementation. WS-Policy is available through and maintained by Microsoft research, easily integrated with our choice of development environment and comes with its own user interface. However, it seems that XACML has more scalability and we are more

[6]http://www.odata.org/libraries/ accessed October 2013

free to choose how to use the policy language. We found two main options for how to work with XACML in .NET, Axiomatics[7] and a framework called XACML.NET[8]. Axiomatics is an enterprise software company with main offices located in Sweden. Their software for fine-grained access are available through payment and even has an API for programming .NET. XACML.NET is a XACML framework for C#, however this implementation seems old and like it is not maintained or developed further. Based on these analysis, we have chosen to create the policy part of the implementation using WSE and WS-Policy.

4.3 Selection of context used in authorization

If we go back to the scenario in this thesis, time and location may be two important pieces of context information, whereas the temperature of the weather outside may not be as important. Although all this information may be considered context in general, temperature will not be considered context for this implementation. We believe in Dey and Abowds selection of primary context attributes[15] and we also feel that i would seem unnatural to build a context-aware application without these primary attributes. Therefore we will consider using three out of the four primary context attributes: UserId, Time and Location as part of our context in this implementation.

However, while these four attributes may be sufficient for building a context-aware access control scheme, our implementation has multiple communication points, i.e the mobile device and the service provider. We believe it is more secure to gather some information from the service provider as well. Imagine if someone were to setup their own WiFi with correct prefix and manages to set up communication to the cloud. In a worst case scenario, these infiltrates gets context information from the mobile device, requests the cloud and can potentially get every piece of information from the requesting users profile. We believe we can avoid this by also gathering context information from the service attached to the service provider. From the service we want to gather all four pieces of the context information. For this impelmentation we think it is more natural that it is the service provider that decides on the action. This action however, will be assumed by the service provider, based on the users context. If we use this information as part of the security we believe that in a worst case scenario, the *fake* service will only be able to gather context information about the user and nothing from the users profile.

To further secure data integrity in this system, we will also gather some information about the service provider. We would like to know the service providers ID and IP-address. The evaluation process of the cloud can therefore check if all the actors in this request are who they say they are.

We would also like to provide the user with some key options they can use in order to force behavior. What this means is to give the user an option

[7] https://www.axiomatics.com/, accessed October 2013
[8] http://mvpos.sourceforge.net/, accessed October 2013

to set a security threshold if they prefer at any time. This means that if the system categorizes the context as *very secure*, the user will be able to force the system not to get more information than what they consider *moderate*. Since this is not a part of the context itself, this will be mentioned more in detail in section 6.

Chapter 5

Implementation description

This chapter will start by creating a detailed scenario of how users interact with our system of context-aware personalized services. We will then analyze each aspect of these scenarios and put these into use cases, followed by a selection of these use cases, and finally provide implementation description with simulation data and expected results.

5.1 System description

In this section we will pick up where we left of in chapter 2, where a person is out traveling. When this persons mobile device comes into contact with the system, the device will immediately start sending context information to the service provider. The service provider asks the cloud for information about the user and uses that information to again provide the user with relevant information based on that profile. Further details are provided in section 2.2 and 2.3.

The scenario will be detailed through use cases and the section will finish with a selection of these use cases, indicating what we see as the main challenges to be solved in this scenario. The selected use cases will then become the basis for the definition of the system that will be implemented.

5.1.1 Scenario

In this scenario we focus on showing users information based on their user profile, on the go. Following is a more detailed description of what is described in section 2.2. We sub-divide the scenario in two parts, *profile management* and *on-the-go*.

The main activities in this scenario are defined as follows:

1. Registration / Authentication
2. Manage objects stored in the user profile
3. Discover services
4. Exchange context information with these services

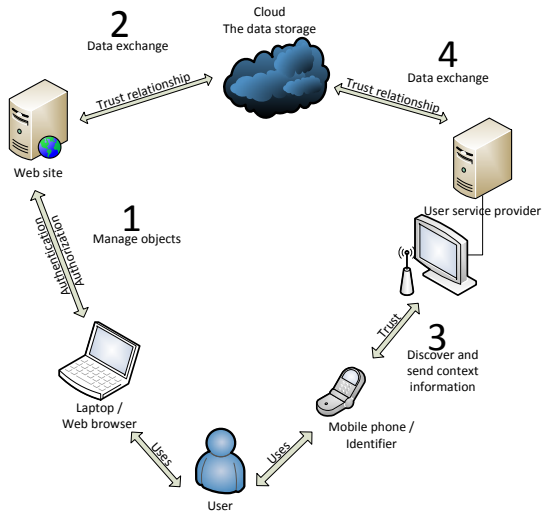

Figure 5.1: Detailed system description for scenario upon profile management and getting personalized services

5. Access objects in user profile from a service provider

We will start with the home-scenario, describing point 1 to 2 where the user updates their user profile. Later in the on-the-go-scenario, describing point 3 to 5, where a user is moving between locations and the mobile phone discovers services.

Profile management

The main goal of the at home scenario is to create preferences for service offers being abroad. These preferences are defined though a user profile, which can be accessed through a web page, mobile app or other services. The web page or mobile app can be seen as a front end for the cloud user profile. A standard user registration system such as username and password, password on SMS, or other methods may be used to get access to the user profile.

To get a system profile, the user needs to register in the system. The profile service lists existing objects registered on the user and provides functionality to register new objects, as well as modifying and deleting existing ones. A user may be able to add food preferences, hotel preferences or travel preferences, but may also be able to add information that is more secure in nature, like a payment card or other similar things. When the users add information to their user profiles, this is to be viewed as an initialization process. When the system has a set of information about the

user, requesting services will be able to reason on this information, not only for further use, but also to add to the users profile. Each of these information is linked through policies defining how the information of this object can be accessed, and what kind of access that is given by fulfilling each policy.

From the clouds front end point-of-view, when a user wants to manage their objects, one of two actions can be performed: *manage* or *read* objects. The front end views a read request when a user wants to list objects or view information about a certain kind of object. A manage request is viewed by the front end when a user wants to create, update or delete an object. Before front end can forward the request from the user to the cloud, the user needs to be authorized by the front end. After the user has been authorized, the request is forwarded to the cloud. Authentication, authorization and object requests to the front end from the user is marked in Figure 5.1 by *1*.

When the request is received by the front end from the user, it is forwarded to the Cloud. The cloud receives the request an first identifies that the request sender is the front end. The front end has special read and write permissions for all objects for the current user, and therefore the request will be approved by the Cloud. The Cloud will manage the current object request with data store, save the changes and return an approved request to the front end. The front end will then show the user the latest changes to objects. The interaction between the front end and the Cloud is marked in Figure 5.1 by *2*. A front end such as a web site may be part of the cloud, but an app is naturally stored on the mobile phone itself.

5.1.2 On-the-go

The main goal of the on-the-go is for the user to be able to exchange information with the system without physical interaction. Interaction between the user and the system is through the mobile device, which holds contextual information about the environment, time, location, activity and other contextual information.

A user walks into an area which runs this system, carrying a mobile phone with wifi turned on. This area can be a train station, airport, buss station, or any other location which is a part of the system. The mobile phone discovers the system via a WiFi prefix and tries to connect to it. All the mobile phones registered in the system will have a certificate, or some other method of identifying that it is registered and a valid component in the system. The mobile phone will send this certificate to the system via WiFi. If the certificate is approved, the mobile phone will now start to gather contextual information from its sensors. This contextual information consist of the ID, time, location and action, the action being the key of what action the user wants to perform, and in this automatic encounter, retrieve relevant information to the user about food services, transportation and so on. After all contextual information is gathered by the mobile phone, it will send this information to the service provider. This interaction is marked in Figure 5.1 by *3*.

When the service provider receives request from the mobile device, it

will start gathering its own context information. The service provider will always only need to read information from the Cloud, so it will send a read request. This read request contains what information the service provider will need about the current user, and all the gathered context information. When the cloud receives the request form the service provider, it will ask the data storage for all the objects. As mentioned in subsection 5.1.1 all objects have policies attached to them which will need to be evaluated with respect to trustworthiness before the Cloud can return this data to the service provider. For each object, every one of that objects policies will be evaluated against the context information received by the cloud. If all requirements by the policy is met, that object will be part of the return message from the cloud back to the service provider. This interaction is marked in Figure 5.1 by *4*.

When the service provider receives object information from the Cloud, it will be evaluated. The service provider wants to show services and information to the user user, based on the preferences and other information that the user provided to his/her profile in the cloud. This information is intended to be shown to the user anonymously, meaning without showing name, user id or other identifiable information. This in particular because this information will be displayed in public. All of these objects will have an unique identifier, i.e an URI. To get a wider understanding about these objects, the service provider uses this object identifier and connects to other open data stores around the world to gather additional information about the current object. This information can be for example a restaurant location, metro time tables, hotel locations availability and check-in times etc... By combining user preferences and information open data stores, the service provider can construct services and information that may be relevant for the user. When the service provider have constructed this relevant information, it will present this information to a TV that is closest to the users location.

One other thing we would like to add to the scenario, is how more secure services, such as payment can treated in conjunction with the user. We use the example of payment or access to payment information being used for restaurant, kiosk or other services, such as ordering of entertainment tickets. The service of ordering, being connected to the cloud, will ask the user for payment information from the users phone. The users phone may then initiate a link between a payment provider and the established service, such that the payment for the service can be performed. The focus here would be on the final provision of information through the users mobile phone, rather than through information in the cloud on its own. However, we do not want to keep information like this in the cloud. This is an example of establishing a shared secret between the payment service and the users bank, a secure channel generated on the mobile phone.

5.2 Use case description

In this section we have divided the scenario into smaller parts and use cases. We first divided the two scenario parts into flowcharts and then created a separate use case for each process. In addition to the two scenarios, we have divided the Cloud into its own part because it is complex on its own and consist of multiple layers. These use cases have been created on the format presented by Skagestein [32], and we have used guidelines provided by IBM [20].

Home scenario

The flowchart in figure 5.2 shows the processes for the home scenario. Following are the use cases that goes for each main process. Orange marks start and end point, green marks web site processes (vertical lines as subprocesses, part of use cases) and purple color marks decisions.

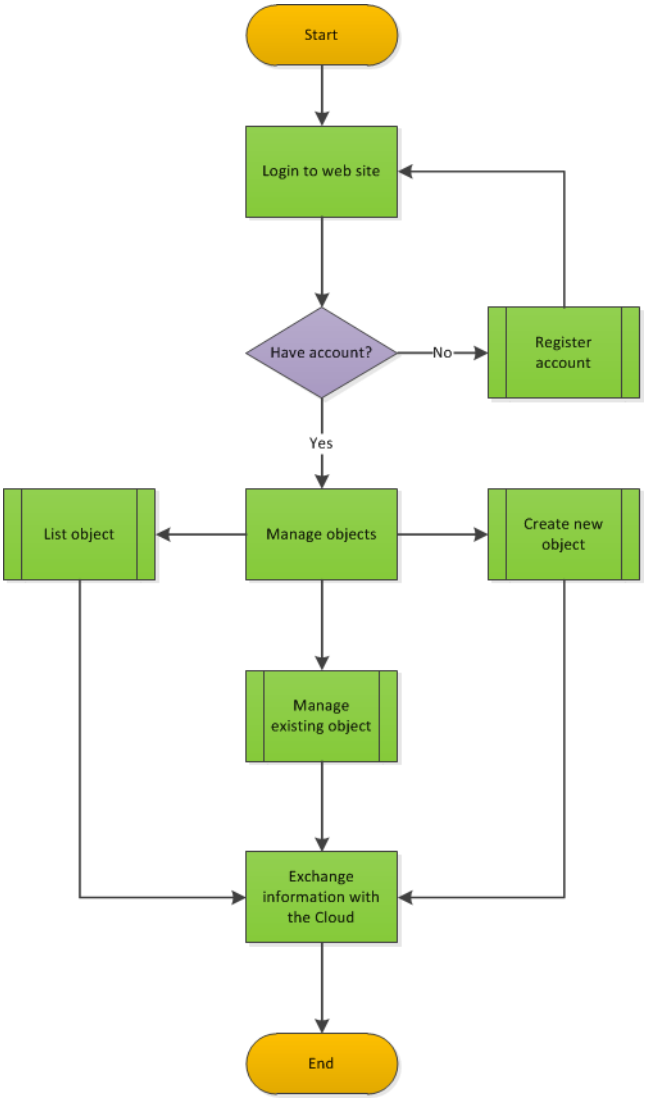

Figure 5.2: Flowchart home scenario where users manage their user profile

Use case 1	**Profile management**
Primary actor:	User
Preconditions:	• User has accessed the profile front end
Postconditions:	• User is logged in • User can access owned objects
Trigger:	User tries to manage objects

Main success scenario:

1. User accesses the front end
2. Front end requests user for username and password
3. User enters username and password
4. Front end authenticates user
5. Front end displays user profile preferences

Extensions:

2.a User is not registered in the system

1. User clicks on the registration link/button
2. User registers
3. User returns to step 5

Use case 2	**Manage owned objects**
Primary actor:	User
Preconditions:	• User is authenticated on his/hers user profile
Trigger:	User interacts with owned objects

Main success scenario:

1. Front end authorizes user
2. Front end presents existing objects to user
3. User performs CRUD operations on owned objects

Use case 3	**Exchange information with cloud**
Primary actor:	Front end
Preconditions:	• Trust relationship between front end
Trigger:	User makes request on the front end

Main success scenario:

1. Front end creates request with context and role information
2. Front end sends request to the cloud
3. The clouds middleware receives the request
4. Middleware exchanges data with the cloud policy engine
5. Middleware constructs a response message
6. Middelware sends response to front end

On-the-go

The flowchart in Figure 5.3 shows the process for on-the-go scenario. Following are the use cases that goes for each main process. Orange marks start and stop, green marks the processes of the mobile phone and blue marks the processes of the service provider.

Describe the figure and link it to the use cases

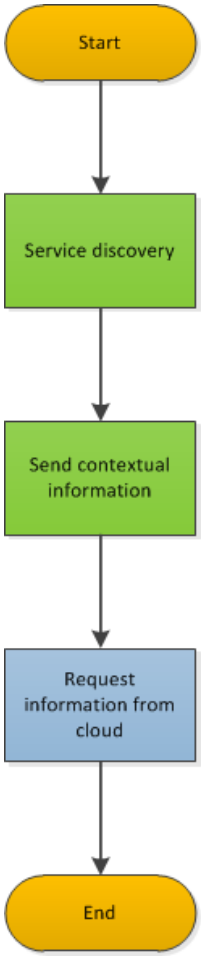

Figure 5.3: Flowchart service provider where the users mobile phone requests personalized services and authorizing access to their user profile by providing contextual information

Use case 4	**Service discovery**
Primary actor:	User/mobile device
Preconditions:	• Mobile device have WiFi turned on • Mobile device must be registered for the system
Postconditions:	• Mobile device is connected to WiFi

Main success scenario:

1. User walks into a system area
2. Mobile device discovers WiFi with prefix name
3. Mobile connects to WiFi

Use case 5	**Send contextual information**
Primary actor:	Mobile device
Preconditions:	• Mobile device is connected to WiFi
Trigger:	Every set interval

Main success scenario:

1. Mobile device gathers context information
2. Mobile device sends contextual information to service provider
3. Service provider sends request to cloud
4. Service provider receives response from cloud
5. Service provider performs actions according to request sendt by Mobile device

Use case 6	**Request information from cloud**
Primary actor:	Service provider
Preconditions:	• Trust relationship between service provider and cloud
Trigger:	Service provider receives context information from Mobile device

Main success scenario:

1. Service provider creates a request with context and role information
2. Service provider sends request to the cloud
3. The clouds middleware receives the request
4. Middleware exchanges data with the cloud policy engine
5. Middleware constructs a response message
6. Middleware sends response to the web site

Cloud scenario

The flowchart in figure 5.4 shows the processes for the cloud evaluation. Squares with no vertical lines represent a main process and will have its own use case. Squares with vertical lines represent subprocesses and will be part of the main processes use case. Orange marks the start end end, blue marks the processes in the cloud, green marks the policies and purple marks objects that are to be evaluated. For this flowchart, we will create only one use case. The reason for this is that even though the *Recieve request* and *Send response* may be categorized as processes, these processes are simple and therefore fits better into one whole use case.

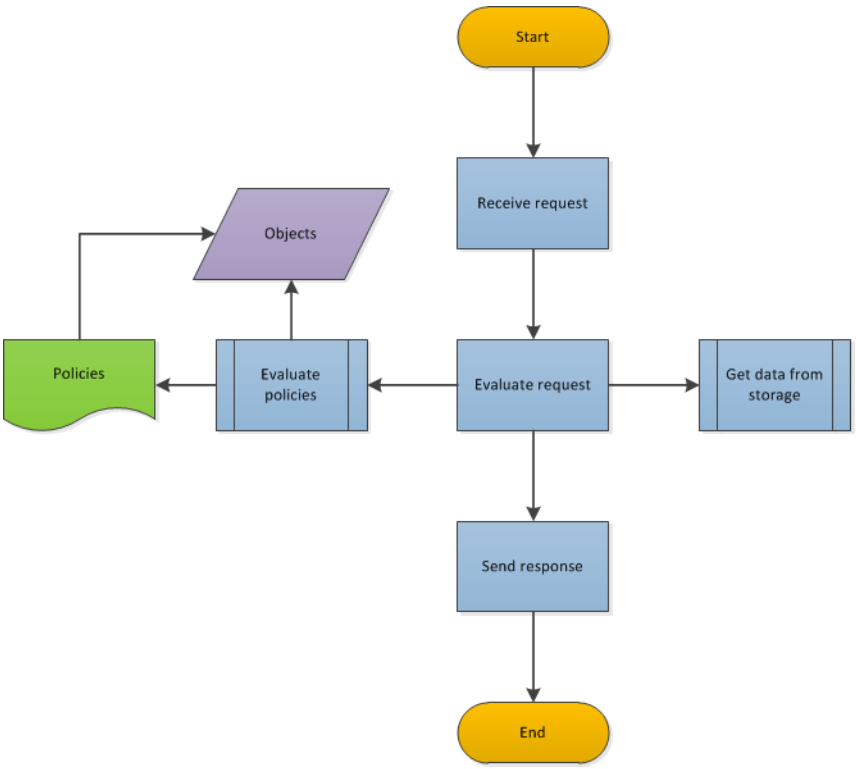

Figure 5.4: Flowchart evaluating access control in the cloud service

Use case 7	**Evaluation process**
Primary actor:	Cloud message handler
Trigger:	Message handler receives request

Main success scenario:

1. Message handler receives a request from the service provider
2. The request is divided into two parts. One part for context and one part for the data request
3. The request is sent to the evaluation engine
4. Evaluation engine retrieves relevant objects from data store
5. Evaluation engine gets policies for each object
6. Evaluation engine evaluates context data against each policy
7. Evaluation engine returns each valid object to message handler
8. The message handler constructs a response message
9. The message handler sends response to the service provider

Data variations:

7a. Object requestor is Web site

1. Actions performed may be CRUD

7b. Object requestor is Service provider

1. Actions performed may only be read

5.2.1 Selection of use cases

In section 5.2 we described all use cases that is relevant for our scenario.

1. Use case 1 - Profile management: This use case can easily be implemented out-of-the-box with the selected .NET framework if created as a web page. For mobile apps, Android and iOS frameworks are available.
2. Use case 2 - Manage owned objects: To achieve this functionality, we need to have communication with the cloud.
3. Use case 3 - Exchange information with the cloud: This use case and use case 2 go hand-in-hand.

4. Use case 4 - Service discovery: To implement this functionality we could have an app on the mobile phone that is specifically created to talk to this system.

5. Use case 5 - Send contextual information: For this use case, the app in use case 4 could gather information from mobile phone sensors or other built-in applications.

6. Use case 6 - Request information from cloud: For this functionality, the service provider needs to be able to communicate with the cloud .

7. Use case 7 - Evaluation process: This use case needs to consist of multiple layers. Message handler, evaluation engine and data storage.

As we can see from the above list and evaluation, some use cases can be fairly easy to implement with off-the-shelf-products and some requires larger amounts of work to implement. The scope of this thesis is context-aware access control authorization, and with this in mind, we believe that in order to make an implementation that reflects our problem statement, we choose to implement use case 5, 6 and 7.

5.3 Scenario simulation

At the above section, we have evaluated the system and selected use case 5,6 and 7. Use case describes how context information is gathered from the mobile phone, sent to the service provider, which requests information about the users profile in the cloud service. Use case 6 describes where we send a request for the users profile information in the cloud with context information and get back parts of the user profile based on that information. To be able to determine what information the service provider is allowed to see, we need to evaluate the request and context, which is described in use case 7. To create and evaluate our implementation, we will create a simulation of a selected user experience. We will first describe this user experience in detail and then we will make a selection of the attributes that will be evaluated.

5.3.1 User experience

In this user experience we have selected two users, Bob and Cathrine. Both of these users will experience the same scenario and have the same information in their user profile. However, Cathrine will have a more trustworthy profile than Bob. Bob is more concerned about security. The result of this difference between profile security is that, even if the users are in the in the same situation, Cathrine will get more personalized information because she allows the service provider to view more information about her profile.

In this simulation, both users will arrive at Gardermoen airport in Norway and has got time to spare before their flight leaves. We assume that they want suggestions of what to do with this spare time. They each carry their mobile phone, which is registered in the system. Bob and Cathrine both have created their user profiles with security policies, by logging into their account using their credentials at the cloud front end. When the users enters the airport, their mobile phone will immediately start searching for service providers. When a service provider is found, the mobile phone will start the authentication process. Once authenticated, the mobile phone will start gathering contextual information and send these to the service provider. For the service provider to be able to display personalized information to Bob and Cathrine, the service provider will generate a request, containing the request itself for profile information and the context information. The cloud will request user profile information locally and tell the policy engine to evaluate each policy attached to the profile information against the received context. For each of the policy information that the policy engine evaluates as approved will be returned to the service provider.

5.3.2 Profile structure and security

Eralier in this thesis we selected the context-aware authorization as the main concern of this implementation, and therefore the main focus of this implementation will be on the evaluation phase. In this evaluation phase, access to certain parts of the users profile is the data we will evaluate access to. The profile is structured semantically as shown in figure 5.5. It starts at the top node *Profile*, which is the root of the profile itself. This root node will have a set of children attached to it, representing each an individual type of data like the users *Food preferences* or information needed to access the users *Sosial networking* information. This hierarchical structure will continue with more specific information of for instance the users food preferences, like *Sushi* or *Pizza* and will follow this structure further with more specific information like *Pepperoni pizza*.

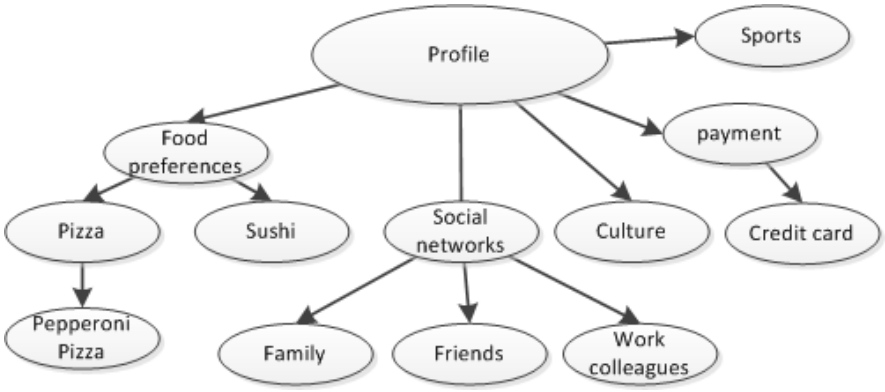

Figure 5.5: Generalized example of a semantically structured user profile

Each of these nodes or statements will be called *Profile Attributes*

(PA) and the first node alone provides some information about the user. By knowing that the user has food preferences, the service requesting information about the users profile will already know that the user may be looking for places to eat. Therefore, the implemented security will start at the first nodes, like Food preferences Sosial networks and so on.

Each of these PA's will have a policy attached to it, as illustrated in figure 5.6, referred with a number. This policy tells the evaluation engine what PA it is representing, what is the criteria to be fullfilled and what access is given by fulfilling this criteria. This implementation will be limited to requests for reading data and therefore all policies will be specified to give read access. The policy criteria is that the requestor of this information, a service, needs to have a *Trust Level(TL)* of equal to or greater than the specified criteria or *Security Attribute(SA)*. The SA is defined in ascending order between 0.0 and 1.0, where higher values represent higher security, meaning that a value of 0.8 is higher security than 0.5. How the trust level is calculated will be covered in more detail in section **??**.

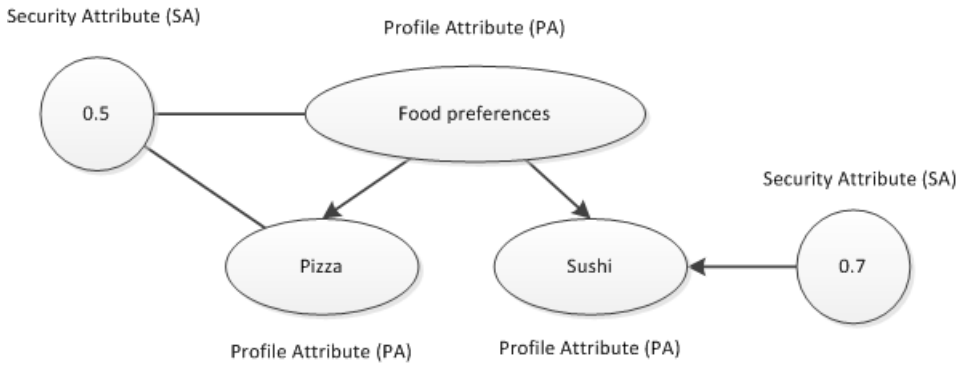

Figure 5.6: Example of a users PA attached with a SA

What we view as the beauty of semantics is the ability inherit this SA further down the hierarchy. If a user has set his/hers Food preferences PA with a SA of 0.5, all children of that PA will have the same SA, unless specified otherwise. This is illustrated in figure 5.6 where Food preferences has a SA of 0.5, which is followed by Sushi. Pizza on the other hand is specified with a SA of 0.7, which is again applied to all of this nodes children. It is important to note that digging through the user profile is done iteratively. This means that if a child has a lower SA than a parent, this information will be unavailable because the iterative process will start at the parent.

Context information

Our selection of context in this implementation consists of the four main context attributes [15], ID, Time, Location and Action. Even though there are only four attributes, they can describe a lot of information about the context. In a generalized setting these can be some of, but are not limited to, the following:

- ID: The users unique identifier in a system, profile, IP-Address, MAC-Address or device name.
- Time: Date and time, time of day or time intervals.
- Location: GPS coordinates, place (hotel, airport), country, continent, county or coordinates in a specific space.
- Action: The users current situation. What is the user doing. What is the users status. Is he/she unavailable, felling private, or available for interaction, and what do we expect the user is trying to achieve.

As shown in the above list, there are an unlimited things to consider when considering context. What context information that is considered relevant may vary for each application or for each service. If a user is trying to access train timetables, the ID may not be as relevant as time and location and when accessing a users profile information, ID may be highly relevant.

Based on our scenario where Bob and Cathrine visits Gardermoen airport in Oslo, Norway at the same time and both of them has two hours of time to spare, we have created a set of context attributes which we find relevant for a context-aware authorization model for access to the users profile.

- ID: Used to identify the correct user profile in the cloud.
- Time: This interaction will take place at 7:00AM, in our case before work hours.
- Location: The place of interaction will be at an Airport in Norway.
- Action: The user has two hours of time to spare and will be in need of something to eat or may be available to interact with colleagues or other known individuals.

Since the main focus of this thesis is based on the authorization model for access to a semantically structured user profile, we have made an assumption that we know the users action, namely that he/she wishes to get suggested places to eat and is available for interaction with known people within their social networks. To be able to determine access to the users profile, the following context attributes has been selected to prove the concept:

Figure 5.7 shows the context for the scenario users Bob and Cathrine. All of these nodes or *Context Attributes (CA)* has a value or *Context Attribute Value (CAV)* like Europe, Norway, Oslo, Gardermoen and so on. This context information will be used in order to gain access to how trustworthy the user finds the current context. Before we can show how this data is intended to be used in practice we will first describe how the users trustworthiness is structured in section 5.3.2 followed by how the this is related to calculating access to the users profiles in section 5.3.3.

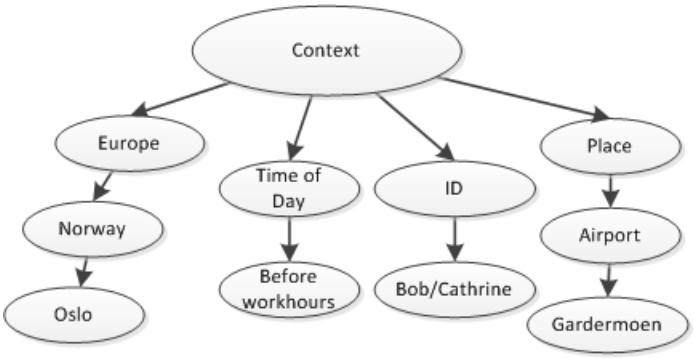

Figure 5.7: Contextual information that will be attached to the requesting services request

User trust

When the cloud receives a request for Bob's food preferences and context information, the evaluation engine in the cloud needs to look at the relationship between the context provided and the trust of the user. Like the profile, the users trust is also structured semantically. An example of a user trust structure is shown in figure 5.8.

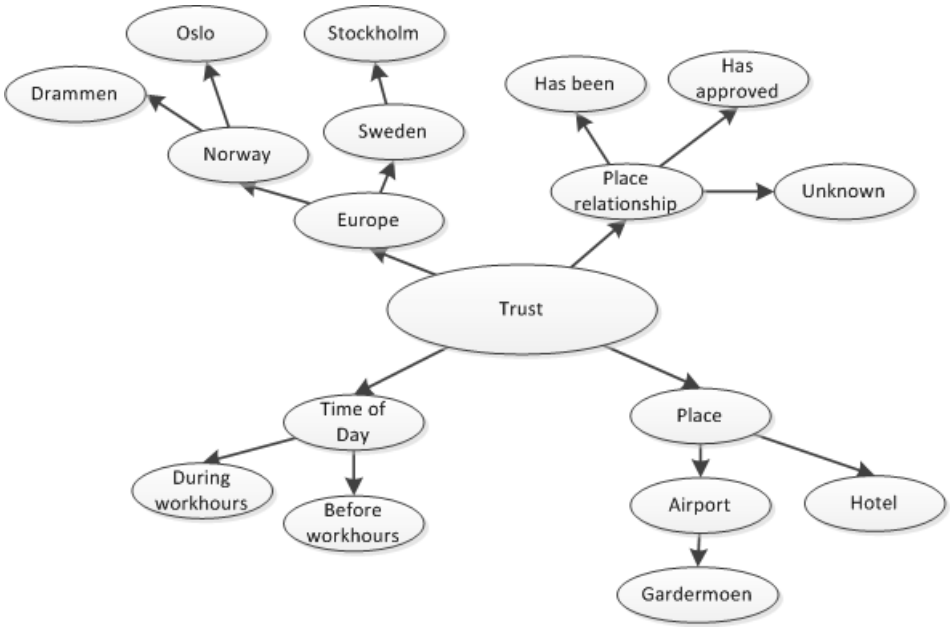

Figure 5.8: Example of semantically structured trust attributes for a user

The build up of a user trust structure and user profile has the same design, but they provide different purposes. In the trust, each of the nodes like Europe, Norway and Place is named *Trust Attribute (TA)*, and is assigned a *Trust Attribute Value(TAV)*. TAV is defined by a decimal value between 0.0 and 1.0, where higher value of TAV represent higher or better trust. As illustrated in figure 5.9, Bob has a trust of 0.7 when present in

Europe and Norway, but is more skeptical when in Sweden. The TAV is inherited by the TA's children, which means that by default every child of that TA inherits the TAV, unless specified otherwise. In the example provided by the figure, children of Norway will by default inherit the TAV og 0.5 where as children of Sweden will continue to inherit the value of 0.5. Bold nodes show TA's that are relevant for the context of this simulation.

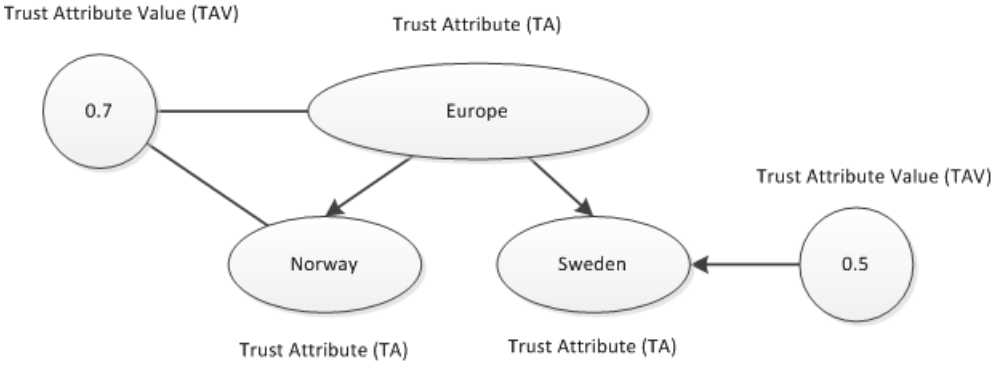

Figure 5.9: Snipping of Bob's trust relationship to Norway and Sweden

5.3.3 Calculating trust level

When the evaluation engine in the cloud receives the request from a service, it will calculate and delegate a *Trust Level (TL)* to that requesting service, which is used to be evaluated against the PA's SA. This trust level will be calculated by using the following process:

Figure 5.10 shows how the trust of a user is traversed upon when a requesting service requests access to the user profile. Based on the context information in the previous section, we will explain how this traversal works, and provide the relevant output data:

1. Start at *Trust*
2. Move to *Europe* - relevant
   (a) Move to *Norway* - relevant
      i. Move to *Oslo* - relevant
   (b) Move to *Sweden* - irrelevant
3. Move to *Places*
   (a) Move to *Airport* - relevant
      i. Move to *Gardermoen* - relevant
   (b) Move to *Hotel* - irrelevant
   (c) Move to *Train station* - irrelevant
4. Move to *Time of day*

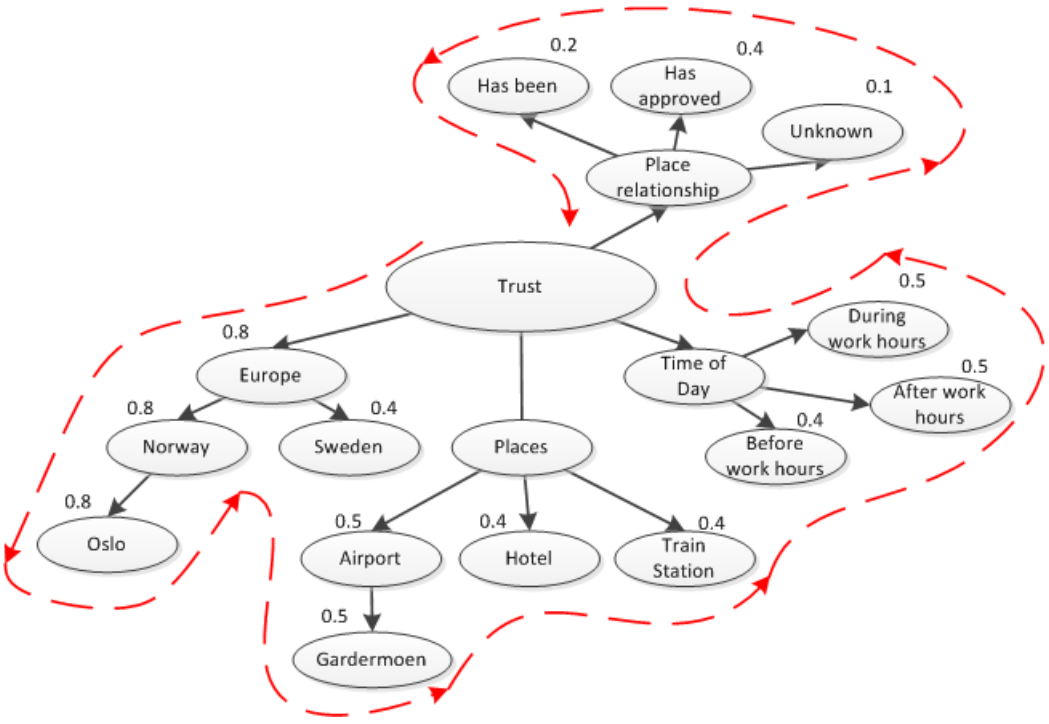

Figure 5.10: Figure showing how the users trust is traversed upon when getting relevant trust attributes based on context information

(a) Move to *Before work hours* - relevant
(b) Move to *After work hours* - irrelevant
(c) Move to *During work hours* - irrelevant

5. Move to *Place relationship*

(a) Move to *Unknown* - irrelevant
(b) Move to *Has approved* - relevant
(c) Move to *Has been* - irrelevant

Through this iterative process, a list of relevant trust attributes will be constructed, ETA. After the evaluation process, the ETA list will be narrowed down into one number because each of the PA's Security Attribute is only one number, Trust Level (TL). To make it simple, we will calculate the average trust level from these numbers by using the following algorithm:

$$TL = \frac{1}{n}\sum_{i=n}^{n} a_i = \frac{1}{n}(a_1 + a_2 + \cdots + a_n)$$

After the TL is delegated the requesting service, the time has come to traverse through the users profiles, based on that the TL is greater than or equal to the SA attached to the PA. The ideal path of the profile traversal is showed in figure 5.11 with Bob's profile. However, in order to be able

to retrieve all information of Bob's profile here, an TL of 0.95 or greater is required. When the profile is traversed, the traversal of child PA's with the lowest SA will be evaluated first because when a SA is reached that the delegated TL is unable to fulfill, further traversal of that PA's children will be stopped.

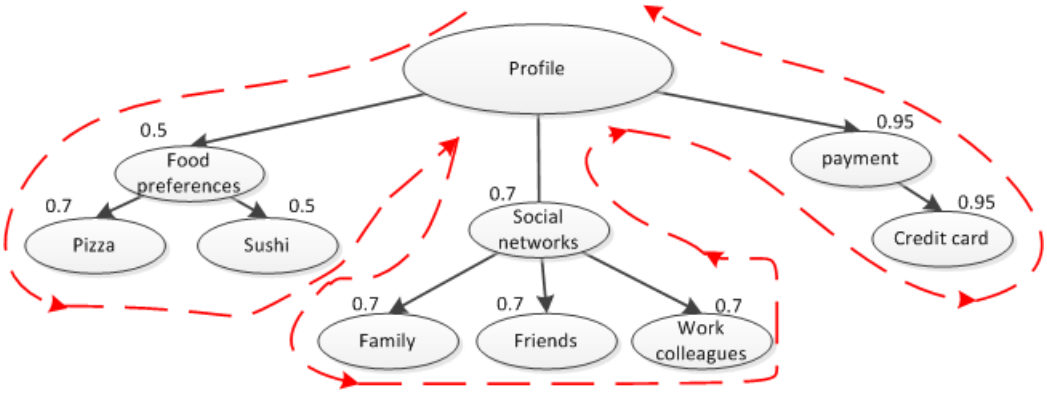

Figure 5.11: Figure showing the process of how the user profile is traversed upon when authorizing access

An example will be provided in order to clarify how this traversal works in practice: A service has been delegated an ETA of 0.5 and will now use this to get information about Bob's food preferences and social networks, starting with food preferences. It will start by visiting the PA Food preferences, where access is granted because TL is equal to that PA's SA. It will continue by traversing the child PA with the lowest SA first, which in this case is Sushi, which will also be granted. The next step is to traverse Pizza, where access will be denied because the TL is lower than 0.7. Backtrack will be done back to Profile, which will start to traverse the PA Social networks. The traversal will stop here because the TL is unable to fulfill the SA requirements for this PA. The same thing will happen for Payment.

The following list summarizes the traversal of Bob's profile with a TL of 0.5.

1. Start at *Profile*
2. Move to *Food Preferences* - access granted
   (a) Move to *Sushi* - access granted
   (b) Move to *Pizza* - access denied
3. Move to *Social networks* - access denied
4. Move to *Payment* - access denied

5.3.4 Simulation data

In this subsection we will break down the scenario of Bob and Cathrine into numbers which will be run in the simulation of the implementation.

They are both at Gardermoen airport in Oslo, Norway at 07:00AM in the morning, has two hours to spare before the flight leaves, and are both open for suggestions to eat breakfast and socialize. Since this thesis focuses on the authorization part of the scenario, the structure of context, profile and trust will be simplified in order to prove this concept, rather than focusing on the semantic structures of the data. We will first present the context information gathered for both of these users, since they are the same, and will follow by showing the profile and trust data for both users.

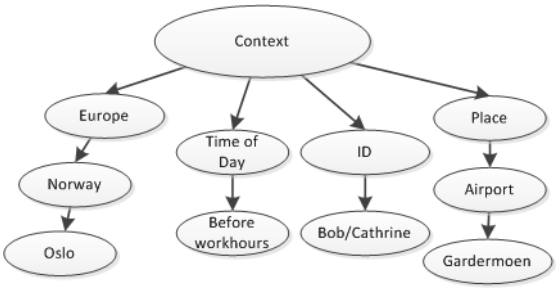

Figure 5.12: Contextual information that will be used in the simulation of this thesis

Figure 5.7 shows the context information that will be used in this simulation. The users find themselves in Europe, Norway, Oslo, the time of day is set to before work hours, the user is either Bob or Cathrine, and they are at Gardermoen airport.

Bob

Figure 5.13 illustrates Bob's profile. As we can see, he is a private person, which requires the context to be very trustworthy in order to view his user profile. Also, most of his profile follows the inherit of parent SA. Both Social networks and payment follow this pattern, but over at food preferences only sushi inherits the parents SA. For the pizza PA, Bob has chosen a SA of 0.7 instead of 0.5, which will be followed by default by pizzas potential children.

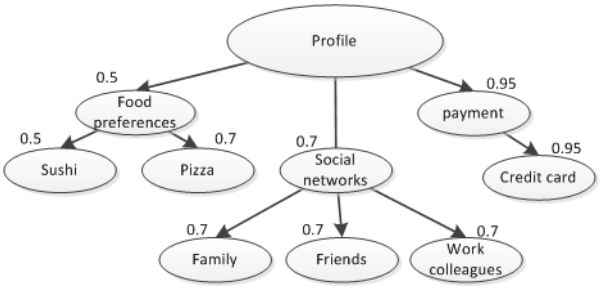

Figure 5.13: Bob's profile that will be used in the simulation of this thesis

As mentioned earlier, the users trust is structured in the same way as the profile, however, as shown in figure 5.14, which illustrates Bob's trust,

not all of these trust attributes makes sense to evaluate. They are in place to keep the hierarchical structure. Examples of these are the TA's Places, Time of Day and Place relationship. In contrast to Europe, these do not give any value to the evaluation. In our opinion, knowing that Bob has trust for Time of Day alone, does not tell anything about his trust to that time. In this case we need to dig deeper into the hierarchy for data that gives information, like Before work hours, After work hours and During work hours.

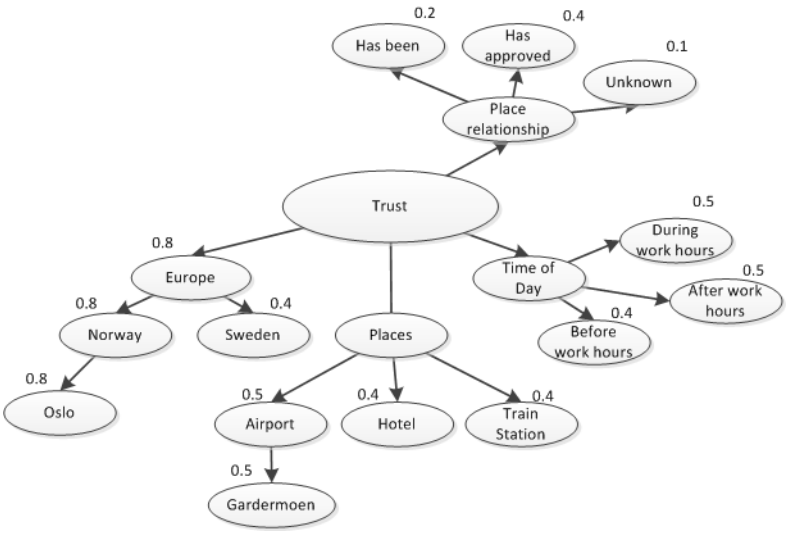

Figure 5.14: Bob's trust that will be used in the simulation of this thesis

Cathrine

Figure 5.15 shows Cathrine's profile. Cathrine is a more open person than Bob, which means that a lower trust level is required to gain access to her user profile. Note that information that are more secure in nature, like Payment has a high security level for Cathrine as well.

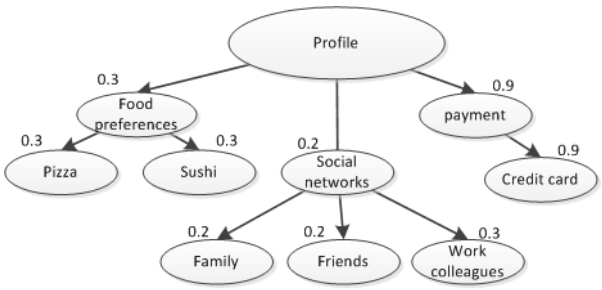

Figure 5.15: Cathrine's profile that will be used in the simulation of this thesis

Cathrine is also a more trustworthy person. She trusts her surroundings more, as illustrated in figure 5.16.

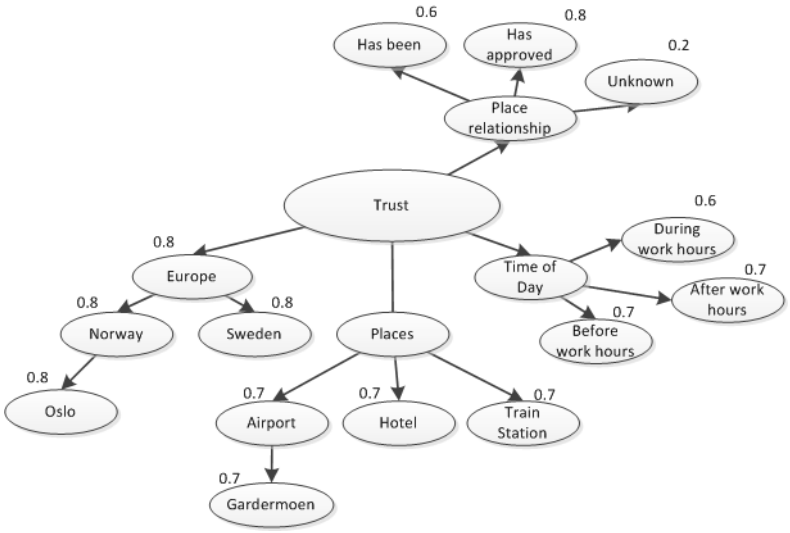

Figure 5.16: Cathrine's trust that will be used in the simulation of this thesis

5.3.5 Expected results

This section will analyze the users trust that is relevant based on the context in figure 5.7, where an assumption is made that Bob and Cathrine want suggested places to eat and is open for meeting people in their social networks. Our assumption is that it would make sense to use different contextual information when requesting different information from the users profile. We choose to use this generalized context for it to be easier to keep focus on the core part of the implementation, context-aware authorization. This section first presents the relevant trust attributes for each user based on the context, followed by a calculated TL for the requesting service and what information that will be returned to the requesting service.

Figure 5.17 shows Bob's trust attributes that is relevant for this simulation. The same is done for Cathrine in figure 5.18.

Trust level will now be calculated based on these selected trust attributes.

$$TLBob = \frac{1}{n}\sum_{n=1}^{n} a_i = \frac{1}{7}(0.8+0.8+0.8+0.5+0.5+0.4+0.4) = \frac{4.2}{7} = 0.6$$

$$TLCathrine = \frac{1}{n}\sum_{n=1}^{n} a_i = \frac{1}{7}(0.8+0.8+0.8+0.7+0.7+0.7+0.8) = \frac{5.3}{7} \approx 0.76$$

As shown from the above result, Bob's trust attributes delegates a TL of 0.6 to the requesting service. Cathrine is much more trustworthy and delegates a TL of approximately 0.76. Figure 5.19 and 5.20 shows what information that is returned to the requesting service for these two users. As illustrated here, Bob is more secure and does only return part of his

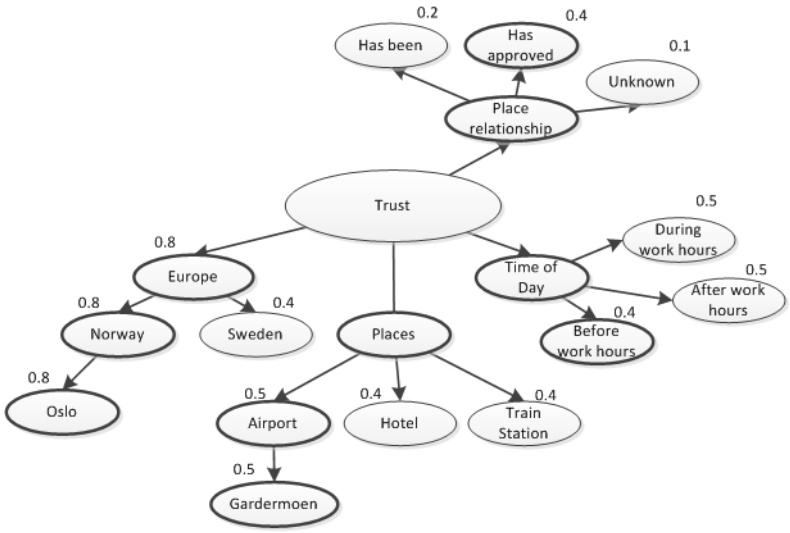

Figure 5.17: Bob's relevant trust attributes to the context information that is sent from the requesting service in the simulation of this thesis

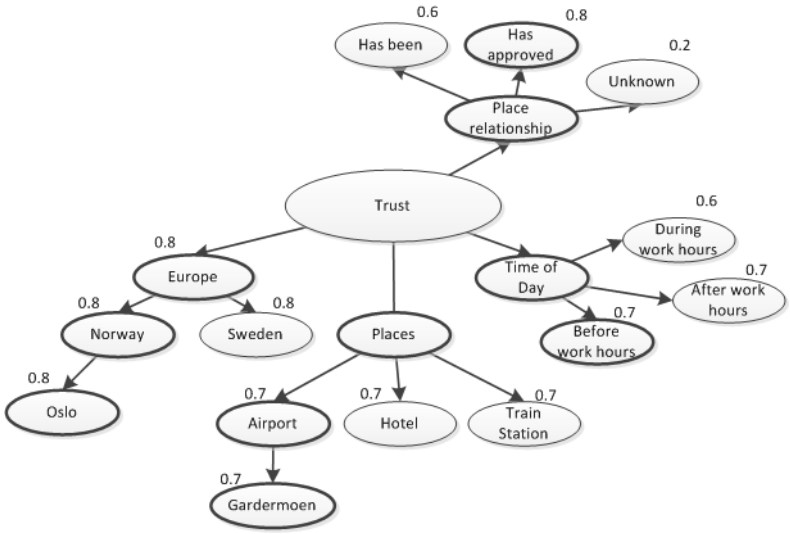

Figure 5.18: Cathrines's relevant trust attributes to the context information that is sent from the requesting service in the simulation of this thesis

food preferences. For Cathrine all food preferences are returned, as well as information about social networking.

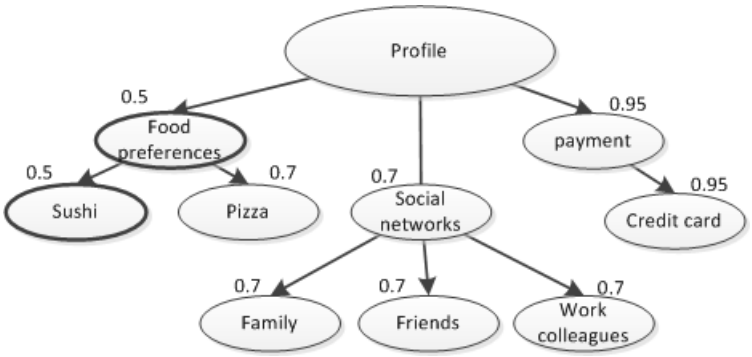

Figure 5.19: Information that is sent back to the requesting service of Bob's user profile in the simulation of this thesis

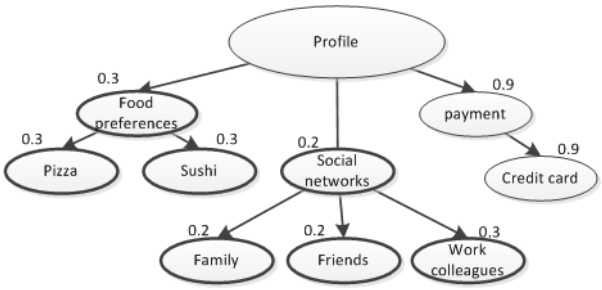

Figure 5.20: Information that is sent back to the requesting service of Cathrine's user profile in the simulation of this thesis

Chapter 6

Implementing access control

6.1 Introduction

Thus far we have talked about our problem statement, created a general scenario and a detailed scenario for our users, Bob and Cathrine. They are both registered users and each have separate profile, though with the same set of preferences, but with a different security requirements to their profile. The scenario of these two users is that they arrive at Gardermoen Airport in Oslo, Norway, and have time to spare before their flight leaves. We will provide these two users with options of what they can do with their spare time, based in preferences used in their store profile. With different security settings, both within their user profile and trust to their current environment, they will have different results to what will be displayed to them om a public screen.

We have also talked about how we can create such an implementation, with requirements to profile storage, how users can secure their profile and limit access to this profile, based on the users current context. We chose to use a cloud service to store the users profile and added policies to their information, which can be maintained by the users themselves. How the services the users encounters can get access the users profile based on their current situation.

In this chapter we will describe the implementation in detail, by following each step from where a food service provider sends a request for the users food preferences and until the service provider has been authorized access to the profile and is finally provided with that authorized food information.

6.2 Changes along the implementation

Earlier we made a selection on technologies and tools we would like to use in our implementation. Along the way of implementation we wither encountered issues that we were unable to solve by using these selected technologies and tools, or in some cases they did not meet our expectations. In this section we will go into detail about these changes throughout the implementation of this thesis.

6.2.1 Policy and policy language

In section 4.2.5 we stated that WS-Policy would be the best choice for our implementation, primarily based on the fact that this policy language was available through Microsoft framework (WSE) and the integration with our development environment. After trying to create the policy implementation using this language, we found that WS-Policy in the WSE framework was mainly used for better security between web web services and not for securing the actual data. WSE policies actually provides better security in addition to SSL. These are XML encryption, roles, certificates and so on. In addition to this it was not available off-the-shelf in VS2012. Here we ran into technical difficulties where we had to manually configure the development environment to make the tool use able.

So after discovering that WS-Policy was not an option for our implementation, we needed to take a closer look at our options, which was XACML. XACML is a flexible policy language where policies and sets of policies can be created, consisting of match criteria, specific resource, and what access fulfilling the criteria gives, whether it be write or read. Requests can also be created by specifying the same criteria as the policies and if the request and the giving policy match, access will be granted.

From what we found later on and during our research phase, we had two options, XACMLNET[1] and Axiomatics Policy Server (APS)[2], which are both based on XACML. XACML.NET is a Microsoft .Net framework, which provide functionality for writing, reading XACML policies and policy sets, reading and writing requests, as well as evaluating requests against these policies. APS is a enterprise software used for simple and complex XACML policies, with user interface both for the policy server and the software for creating policies, as well as framework for writing your own policy evaluator.

As our task was not to create a XACML implementation, we contacted Axiomatics, which was happy to provide us with a developer edition of their APS. We soon realized that this did not go as good as we had planned. Axiomatics was pretty busy, and was not able to help us as much, and along with insufficient documentation we realized that using this software would take up too much of our time. At this point we had to either create our own framework to use with XACML or use XACML.NET. The thing about XACML.NET is that is had not been updated since 2005, and when we tested small parts of that framework during our research phase, the framework was not able to read the policies it had created itself, in other words it was useless, and the site to where we could download the source code was taken offline.

Fortunately, jetBRAINS, a company that, among other things, creates tools for .Net development, has created a tool called DotPeek[3] that can be used to decompile .Net compiled code. This year, they also released

[1] http://mvpos.sourceforge.net/, accessed September 2013

[2] https://www.axiomatics.com/axiomatics-products-overview.html, accessed September 2013

[3] http://www.jetbrains.com/decompiler/, accessed October 2013

DotPeek version 1.1 which is able to decompile code to a Visual Studio project. We decompiled the XACML.NET source code and made alterations to suit our needs of creating policies, policy set, requests and evaluate these requests against each other. Out-of-the-box, the XACML.NET framework was a bit complex to work with, so we also created a policy engine on top of this framework to simplify some of the operations, like writing and evaluating policies.

A more detailed description of how this policy engine operates and fits into our implementation will be covered in section

6.2.2 Implementing a cloud service using OData

We knew early on in this thesis that we were going to use Microsoft's Open Data Protocol (OData) and our research showed that we had one option of using such an implementation. We first set up our database either by defining the structure of the database with programming classes or by creating the database first and then import the database in our development project. After this was done, we could expose the data with the following options:

1. Expose entities[4] statically.
2. Expose entities statically and write semi-dynamic queries on what to expose.
3. Expose data by using operations and actions that uses input to create more dynamic data retrieval.

Throughout the implementation we realized that the off-the-shelf implementation had several shortcomings to what we wanted to achieve in our implementation. First of all we discovered that OData off-the-shelf had an open approach, meaning that, by default it exposes all the data.

```
1 public class CoudDataService : DataService<DataContext>
2 {
3   public static void InitializeService(DataServiceConfiguration config){
4     config.SetEntitySetAccessRule("Users", EntitySetRights.AllRead);
5     config.DataServiceBehavior.MaxProtocolVersion = ↩
          DataServiceProtocolVersion.V3;
6   }
7 }
```

Listing 6.1: Example for exposing user entities with off-the-shelf OData product

The above figure shows an example on how to set access rules to entities, where in this case we say that we have read access to all users in the system. If we were to limit what users that are shown, we would

[4]Entity is a object that contains data, i.e. Employee, Order, http://www.odata.org/documentation/odata-v2-documentation/overview/, accessed October 2013

have to scale down to that specific user. We believe that this is a security risk in itself by assuming that we have access to everything, and have to prove otherwise, instead of vise versa. Also, our proposed system has a front end where users are able to update information about themselves and their entities, which means that not only do services need read access, they also need update, or even write access in some cases. If the system by default assumes that requesters have this kind of access by default and this has to be scaled down, we view this as a critical flaw in the software. And unfortunately there is no way of setting these entity access rights dynamically.

We thought that we might be able to solve the issue of "access by default" by using actions and operations. That way we could force requesters to prove that they have access to this user and that part of the profile, and then return them that way. Unfortunately, these actions and operations is only functionality in addition to the already exposed data. An example here would be to create an operation that gets information about Bob based on a set of parameters we pass to this operation. However, this is not an issue that cannot be solved by using custom HTTP headers to be used in semi-dynamic queries.

Our conclusion of using off-the-shelf OData is mainly to publish data you want to expose on the web in an easy way. We partly knew that this is the core idea of OData, but we assumed that this behavior could be manipulated in the way we wanted. It does not provide the dynamic functionality we needed, and in our opinion, provided a set of "back doors" that needs to be closed manually, which we believe is close to impossible[5]

After further research we found the OData Web API[6], which is used to create an OData service from scratch, with nothing implemented by default, and we can specify what functionality we want, rather then specify what we do not want. We have in this case not made a full OData implementation using this API, since this is not essential for this thesis, and can be considered to be a thesis of its own. We have however through the implementation made some observations to what impact this can have on the development process and some we consider being some of the shortcomings when implementing a OData solution based on the API.

We believe that using the OData Web API has impact the development progress on both the client side and server side. By using the OData Web API we need to implement the OData functionality we need from scratch, which takes a lot of time. OData features we will need for this implementation is to expose the data itself, once it it authorized to be exposed, navigation between the exposed data as well as actions that can be used by clients, in our case used to authize profile access. The client will also suffer from an OData Web API implementation. Rather than talking with a service, the client will have to serialize and deserialize request and response, because with a API implementation, the the OData provider will

[5]http://www.mcafee.com/us/resources/white-papers/foundstone/wp-pentesters-guide-to-hacking-odata.pdf, accessed October 2013

[6]http://www.asp.net/web-api/overview/odata-support-in-aspnet-web-api, accessed October 2013

be an application, rather than a service. We will go into detail about our server and client implementation in section **??** but will show an example, how the off-the-shelf OData and the API implementations are different from a clients point of view.

```
HttpClient client = new HttpClient();
client.BaseAddress = new URI"http://localhost:53984/odata/");

var response = client.GetAsync("Users").Result;
Console.WriteLine(response.Content.ReadAsStringAsync().Result);
```

Listing 6.2: Client implementation for how to request user entities form OData service implemented using the OData Web API

```
{
  "odata.metadata" : "http://localhost:53984/odata/$metadata#Users", "↩
      value":[
    {
      "Id" : 1, "FirstName" : "Bob"
    },{
      "Id" : 2, "FirstName" : "Cathrine"
    }
  ]
}
```

Listing 6.3: JSON response for user from service implemented using OData Web API

Code example 6.2 shows a small example on how to get all users from a OData Web API implementation and code example 6.3 shows what the request from such an implementation. Notice that what we retrieve from the API implementation is raw data that we need to serialize to objects before we can make further use of them programatically.

```
var cloud = new Cloud.CloudContext(new Uri("http://localhost:50668/OData.↩
    svc/"));
var users = cloud.Users;

foreach(var user in users)
{
  Console.WriteLiine("{0}: {1}", user.Id, user.Name);
}
```

Listing 6.4: Client retrieving all users form off-the-shelf OData service and printing names

```
1: Cathrine
2: Bob
```

Listing 6.5: Output from OData off-the-shelf client

As we can see from figure 6.4 it has the same amount of code as the API client, however, this code is already serialized into an object. The OData service reference also comes with class descriptions and does the serializing

for us. Even though these two examples has the same amout of code, in our last example, the class is already serialized into an object description the service also provides us with. As we can see from listing 6.5 in contrast to code example 6.3, these values comes directly from the object and are useable. We will not go into further detail about this but would only like to mention that navigation properties from the object, User in this case, is directly available through the programatic access in the entities.

6.3 Implementation

This section will address the implementation step-by-step according to figure 6.1. The process will start from the requesting service point-of-view by constructing a request and sending this to the cloud. The cloud will process this request by first evaluating and calculating the Trust Level for the requesting service. The delegated TL will be evaluated against the Profile Attributes Security Attribute, and if the TL is greater than or equal to the TL, that PA will be part of the response message. The response message will be constructed, containing the authorized PA's and the message will be sent back to the requesting service. The cloud service is divided into two main components as shown in figure 6.1. The message engine is responsible for communicating with services that requests information from user profiles. The evaluation engine is responsible for authorization of the requesting service to the user profile.

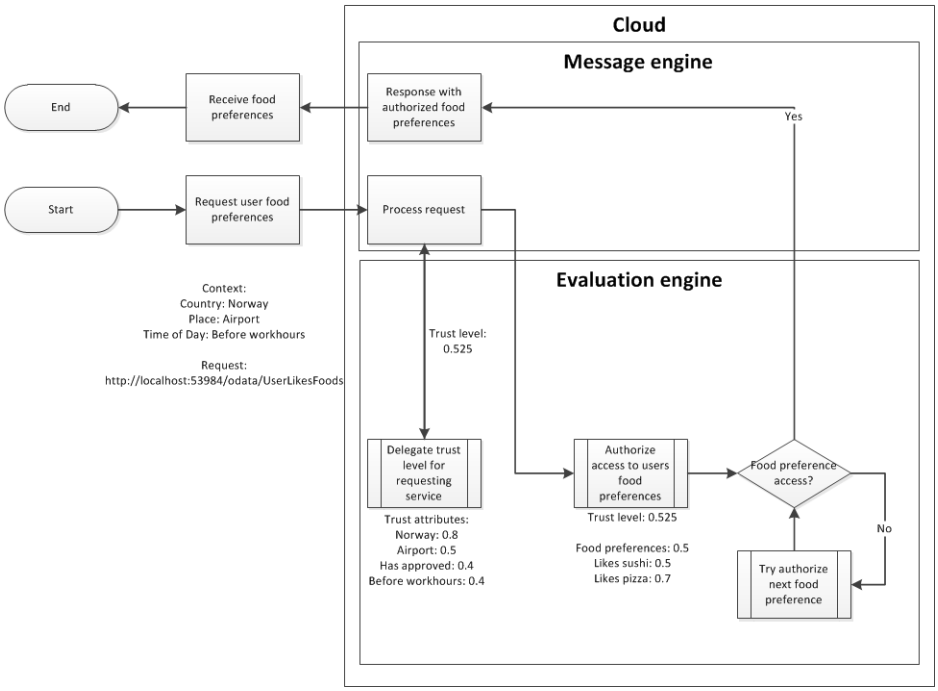

Figure 6.1: Flow chart for process when service provider request information about users food preferences and receives part of user profile

6.3.1 Send service request

Referring to the flow diagram in figure 6.2, marked with a bold border, is the process that will be covered in this section, namely the construction and sending of a request from the requesting service.

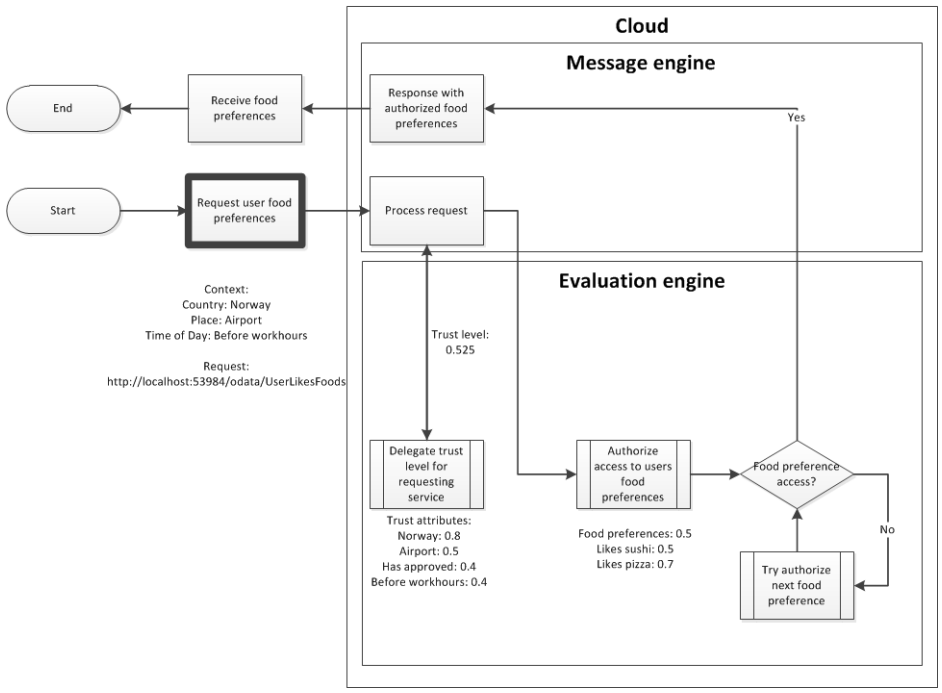

Figure 6.2: Flow chart shows how the implemented framework builds a request for requesting information about users food preferences

The message constructed by the requesting service contains two parts, the request itself and the context information, where the request is the information the requesting service would like to retrieve from the user profile and the context information being the context gathered from the users mobile device.

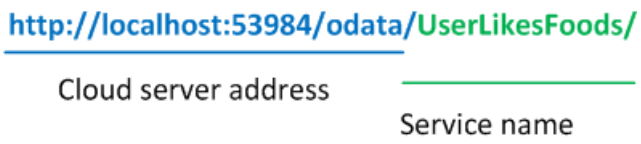

Figure 6.3: How the request is built up by dividing a URI into cloud address and service address

As shown in figure 6.3, the request part of the message consists of a URL. OData, whether it uses the OData Web API or the off-the-shelf solution, both of these options is a web service. Communication between the cloud and the requesting service is done over the HTTP protocol[16], and is followed by the address and port this server is set to respond to (in

blue). We choose to keep the option open for creating a profile management front-end in the cloud and has therefore created a route *odata* which is the part of this web service the other requesting services will communicate with. The next part of the URL is the service we are requesting (green), which is also the most essential part of the URL that will be changed for each request.

We stated early on in this thesis that the OData protocol lives true to the name. It is mainly used for open data, which means that passing parameters for each request is not trivial. However, for open data it is. OData comes with a module called *query options* which provides the requester the ability to filter the data as they see fit. Unfortunately for this implementation, these query options uses the argument part of the URL i.e. *odata/UserLikesFoods?$top=10* which rules out this usage and using these query options from the client is not a valid option, simply because this has to be done by the service itself and not the user. We still have to enforce this because we want to send contextual information as well as the request itself. To our knowledge, there are two options for passing the context parameters with the URL, using an OData action or via HTTP headers. The following tables present what we consider being the main pros and cons of either of these approaches, followed by an analysis of these and will finish off with a selection of approach.

Action

Good	**Bad**
Fully uses ODatas potential	POST message
Strongly typed data parameters	Breaks the semantics
Parameters is part of the main request	

HTTP headers

Good	**Bad**
GET message	Have to step outside the usage of OData protocol
Keeps semantics in tact	

The above tables show that using actions has a clear disadvantage. First of all, by using actions to pass contextual information as parameters results in that a POST[16] request needs to be done. The purpose of a POST request is to submit information or change the state of data in a service, this is not what what this cloud is supposed to do. Requesting services should only retrieve information from the cloud at this point. However if the requesting service in future work should add information to the cloud, POST should be used. So for retrieving information from the user profile, GET should be used.

One other thing that is good about using these HTTP headers is that it keeps the navigation properties in tact, which we consider to be important. An example is that by using actions the request URL needs to contain the function name, i.e. *odata/UserLikesFoods/FunctionName*. As mentioned

above actions are not made for GET requests, which means that by providing such a request URL, navigation like *odata/UserLikesFoods/FunctionName(1)* for retrieving the first record of the returned data does not exist, and even if it did, it conflicts with the purpose of the HTTP protocol.

The bad thing about using HTTP headers is that we have to step outside the OData protocol in order to use it for this specific purpose and actions provide strongly typed data, which means that whatever object or data structure one chooses to use as parameter can be user. However, by weighing these two against each other, we feel that the cost of strongly typed parameters are worth stepping outside the protocol by using HTTP headers. By doing this, we maintain the clean request URL's, semantics and avoid fighting the purpose of the HTTP protocol.

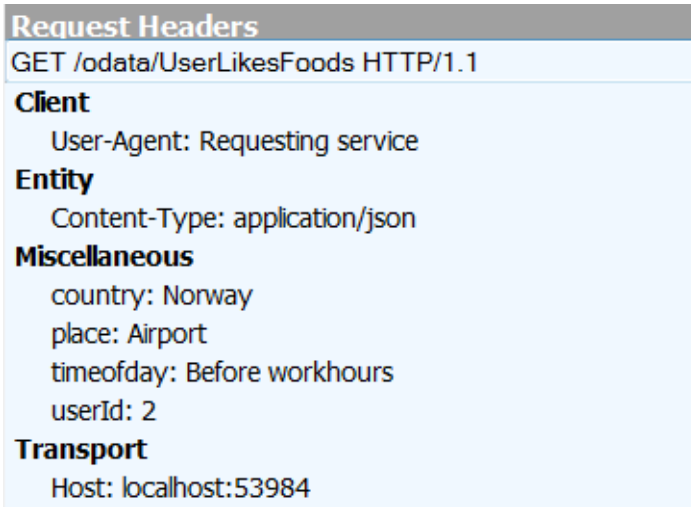

Figure 6.4: Request header with context information for requesting Cathrine's food preferences

A typical request for this implementation this implementation is illustrated in figure 6.4. We have chosen to simplify the context from being semantically structured to a key value pair because the main part of the implementation is the authorization process and these data are stored in a MSSQL database, which is a rational database where each table is attached to a class and object represents one row and the main part of the implementation is the authorization. An alternative would be to create one header with JSON[7], deserialize this to an array and run the path on a another OData service hierarchically, but we feel that this sidetracks too much with this implementations original purpose.

6.3.2 Evaluate trust

This section covers the next step in the process, as illustrated in figure 6.5, which is processing the request and calculating the TL which is to be

[7]http://json.org/, accessed November 2013

delegated to the requesting service for this request.

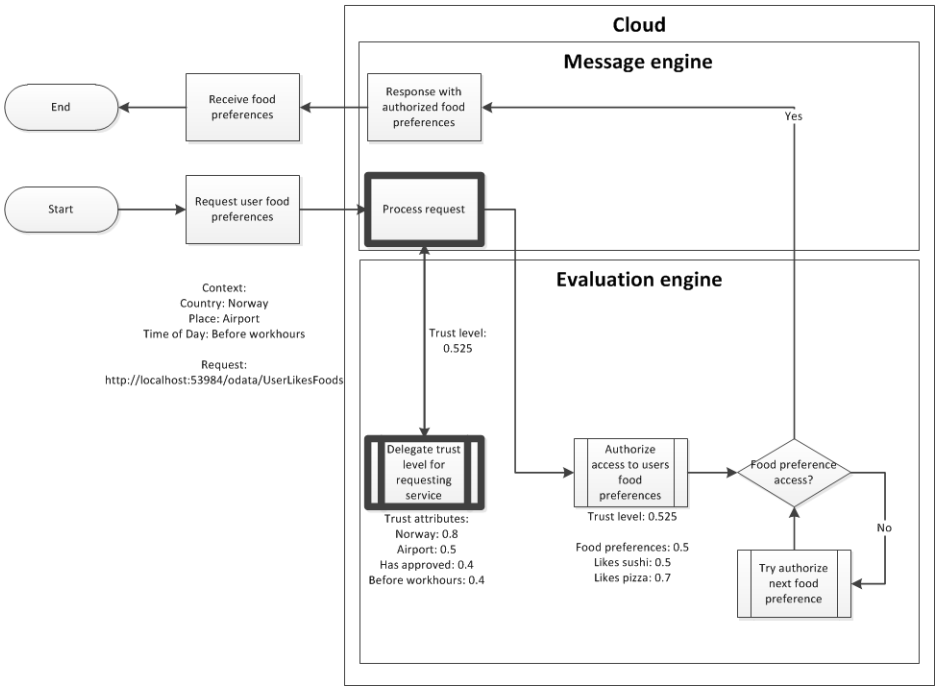

Figure 6.5: Flow diagram showing where in the request process trust level is calculated

The last paragraph in the previous section describes the request and how and why it has been simplified into key value pairs. A part from this, the implementation keeps true to its original purpose where it looks at the TA's TAV and calculates the average of these set of numbers into TL that can be evaluated against each of the PA hierarchy in the users profile.

Figure 6.6 illustrates an entity model of a users trust structure. The entity model is a hybrid between a relational database model and a class diagram. It describes the class/table name, properties/columns and foreign keys/relations between these. Entity Framework (EF) is the Microsoft Framework that creates this schema which provides the programmer with the ability to solely focus on programming with the classes and removes the concept of the actual database.

```
1  var placeStatusTrust = (double)
2    (from p in context.UserHasPlaceStatusTrusts
3    where p.Place.Name.Equals(place) && p.UserId == userId
4    select p).FirstOrDefault().TrustLevel;
```

Listing 6.6: Getting trust for a users place relation

Code example in listing 6.6 shows an example of how to retrieve the users place relation to the actual place. In this example the we wish to retrieve the trust level for a specific place related to the user. When when the TAV for all TA's for that user in the current context is retrieved, the

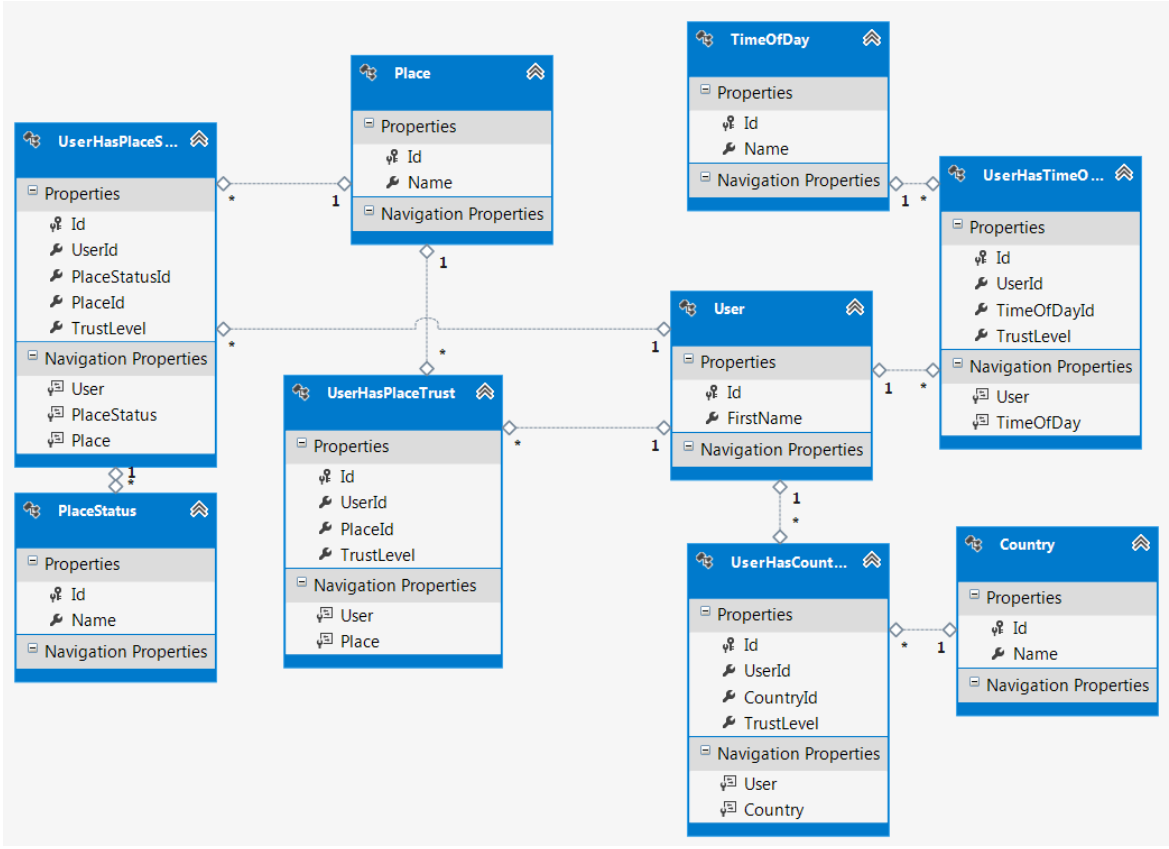

Figure 6.6: Entity model of how trust is structured in the cloud

average of these are computed and returned back to the message engine in the cloud.

6.3.3 Authorize access to the user profile

This section describes how the evaluation engine in the cloud authorizes access to the specific parts of the user profile. This authorization process will make use of the delegated TL for the requesting service, requesting the users food preferences. The part of the flow diagram that this section will cover is marked by bold borders in figure 6.7. We will start by describing how the user profile is structured, how the SA' are structured and attached to each PA, and will finish off with how authorization is given for these PA's.

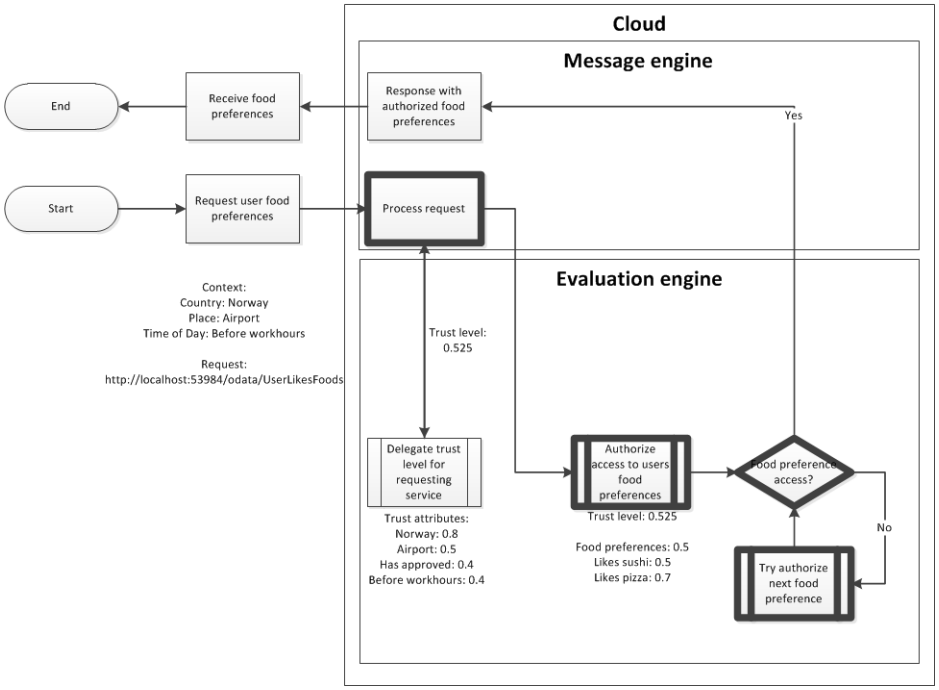

Figure 6.7: Flow diagram showing processing request and authorizing access to the user profile

Much like section 6.3.2 describes how context trust is structured, the profile is has the same basic structure. The profile is also created programatically by using EF, which creates the entity model illustrated in figure 6.8. The figure however does not imply anything about the security. The enntity model is just a way of having a common understanding between the code and database.

Security is achieved in collaboration between OData entities and XACML policies, which in terminology of this thesis are respectivly PA (OData) and SA (XACML). Each PA has a SA attached to it, where the PA is identified by a unique URI. To clearify we will describe the structure of the policies and how they are attached to the PA in order to become a SA.

The structure of the policies are done hierarchically to keep true to the hierarchical structure of the user profile. This functionality could in theory be achieved by using hierarchical resources in XACML[3], but in addition

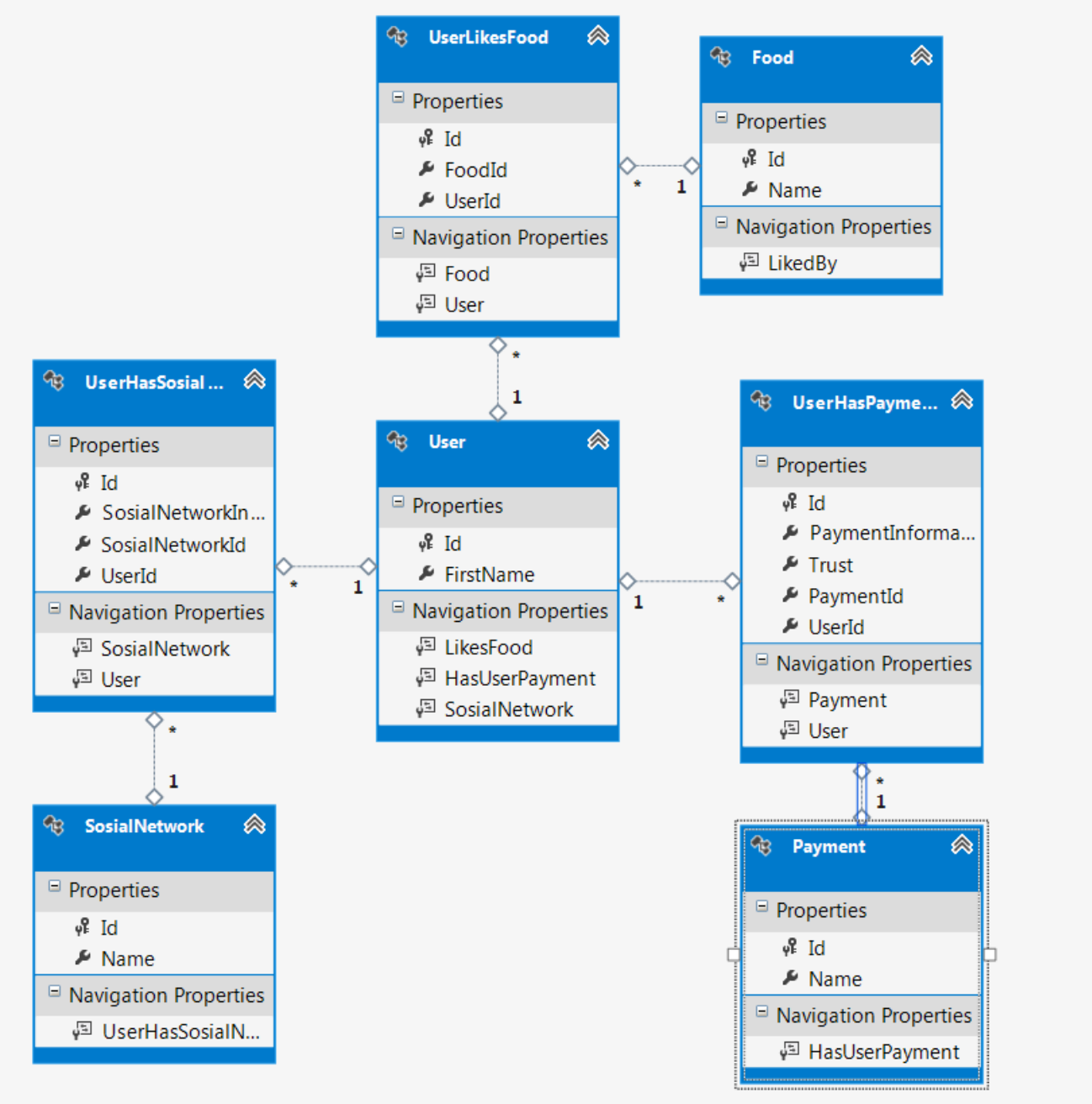

Figure 6.8: Entity model of how the profile is structured in the cloud

to that the SA can vary through the hierarchy and our selected framework for working with XACML policies, XACML.NET does not support this, we decided to use a hierarchical catalogue structure. What this structure means in practice is that the policy describing access to food preferences needs to be authorized before each of the food preferences like sushi and pizza.

Listing 6.7 shows a snipplet of Bob's policy for PA food preferences.

```
<?xml version="1.0" encoding="utf-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  PolicySetId="policySet"
  PolicyCombiningAlgId="...:rule-combining-algorithm:deny-overrides">
    <Description>description</Description>
  <Policy PolicyId="LikesFood(1)" RuleCombiningAlgId="...:deny-overrides">
    <Description>...</Description>
    <Target>
  ...
    </Target>
    <Rule RuleId="..." Effect="Permit">
      <Description>description</Description>
      <Target>
        <Subjects>
          <Subject>
            <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double">0.5</AttributeValue>
              <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="http://www.w3.org/2001/XMLSchema#double" SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" MustBePresent="true" />
            </SubjectMatch>
          </Subject>
        </Subjects>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">UserLikesFoods(1)</AttributeValue>
              <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false" />
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
              <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
    </Rule>
  </Policy>
  <Policy>
  ...
  </Policy>
</PolicySet>
```

Listing 6.7: Slipplet of Bob's policy for defining access to food preference sushi

Listing 6.7 shows a snipping of the policy set that is attached to Bob's food preferences, in this case the food preference, sushi. The below list will

describe the policy in more detail, and especially, where PA and SA fits in.

2 PolicySet: Start tag for Bob's policy set.

4 PolicyCombiningAlgId: Means that the rule needs to be fullfilled, and no other rules can override it.

6 Policy: Start of a new policy for Bob's food preference, sushi.

11 Rule: The policy rule that needs to be fullfilled, and if so this leads to permitting access.

14 Subject: This is the SA that will be evaluated against the incoming TL of the requesting service. If the TL is greater than or equal to the SA, access to the PA attached to this policy will be granted.

23 Resource: The PA which this policy is defined for, in this case UserLikesFoods(1) (sushi).

31 Action: What action is permitted if the requirements are met, in this scenario, read access.

When the process reaches the point where access to food preferences is to be authorized, the TL is available and the evaluation engine will start to try and authorize access to the PA's. This is an iterartive process where access the TL will first be evaluated against the food preferences PA, and if authorization is permitted it will move further down the hierarchy to the children. For simplicity sake we have in this implementation only created food preferences with two children.

Since this is the most complicated part of the implementation, we have provided a somewhat simplified code example in listing 6.8 followed by a description of what each of these lines do.

```
1  public override IQueryable<UserLikesFood> Get()
2  {
3          var userLikesFoods = from u in context.UserLikesFoods
4                  where u.UserId == user
5                  select u;
6
7          //PA's to output to requesting service
8          List<UserLikesFood> output = new List<UserLikesFood>();
9
10         //Evaluate food preferences PA
11         if (PolicyEvaluator.EvaluateParent(user, trustLevel, resource))
12         {
13           //Loop through every PA and evaluate policy
14           foreach (var userLikesFood in userLikesFoods.ToList())
15           {
16                   if (PolicyEvaluator.Evaluate(user, trustLevel,resource,↩
                         userLikesFood.Id))
17                   {
18                           output.Add(userLikesFood);
19                   }
20           }
21         }
22         return output.AsQueryable();
23 }
```

Listing 6.8: Getting trust for a users place relation

1 Method name user to identify the *UserLikesFoods*-part of the request URL

3 Get all food preferences for user Bob as a list

8 An empty list that contains all potential food preferences to return.

11 Evaluate the food preferences PA against TL delegated to the requesting service.

14 If access is granted to the food preferences PA, evaluate access to sushi and pizza PA.

16 If access is granted to sushi and/or pizza PA, add them to the list which should be returned to the requesting service.

22 Return the list of authorized foor preferences.

Once the evaluation engine has evaluated authorization to the users food preferences, the authorized part of the semantically structured user profile will be returned to the requesting service.

6.3.4 Respond with authorized user profile

When the cloud has authorized part of the user profile from our example request, in our case food preferences, is returned to the requesting service. This section will cover the returned response, as shown in figure 6.9, a response to the requesting service with authorized parts of the user profile. This section will also cover how to browse the returned semantic data structure.

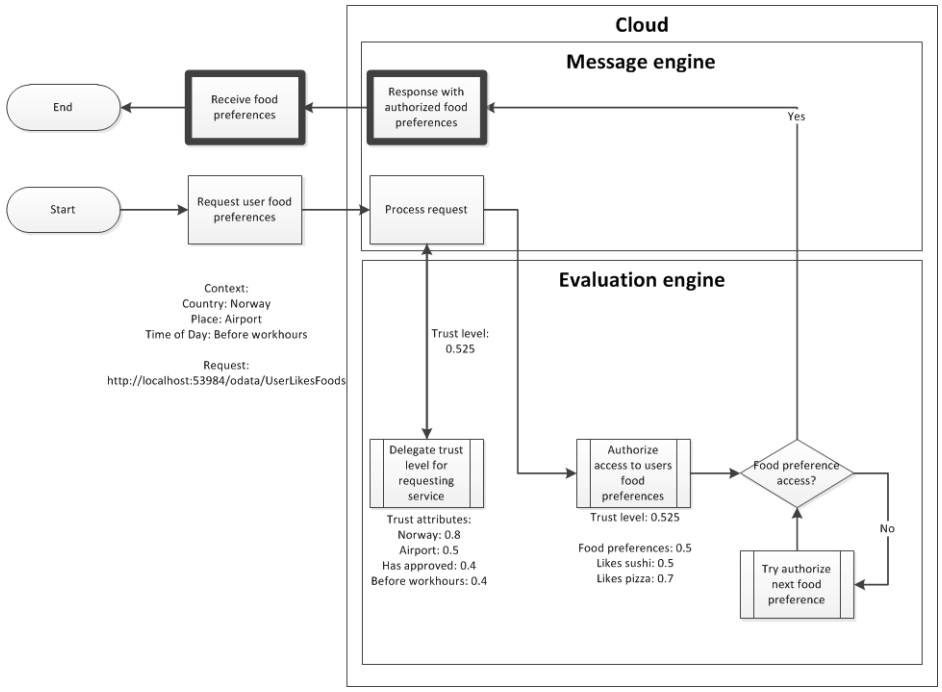

Figure 6.9: Flow diagram when a reequesting service receives a response with authorized parts of a user profile

The raw data returned to the requesting service is a list of the PA nodes within a specific level, with links to other data. This means that the semantic structure of the user profile is not all returned at once. An example is illustrated in listing 6.9, where food preferences of sushi and pizza is returned to the requesting service.

```
{
    "odata.metadata" : "http://localhost:53984/odata/$metadata#↩
        UserLikesFoods", "value":[
        {
            "Id": 5, "FoodId": 1, "UserId": 2
        },{
            "Id": 6, "FoodId": 2, "UserId": 2
        }
    ]
}
```

Listing 6.9: Example of returned food preferences for Cathrine as JSON

As we can see, the returned PA's for food preferences with Id 5 and 6 does not tell the requesting service if the food with FoodId is sushi or pizza. There are two ways this problem can be solved, either by having knowledge about these Id's in the stored in the cloud, or by providing metadata to services that communicate with the cloud about the meaning behind these data. This implementation has enabled metadata for this purpose.

```
1  <EntityType Name="UserLikesFood">
2          <Key>
3            <PropertyRef Name="Id" />
4          </Key>
5          <Property Name="Id" Type="Edm.Int32" Nullable="false" />
6          <Property Name="FoodId" Type="Edm.Int32" Nullable="false" />
7          <Property Name="UserId" Type="Edm.Int32" Nullable="false" />
8          <NavigationProperty Name="Food" Relationship="......" ToRole="↩
               Food" FromRole="FoodPartner" />
9          <NavigationProperty Name="User" Relationship="......." ToRole="↩
               User" FromRole="UserPartner" />
10 </EntityType>
```

Listing 6.10: Metadata information about food preferences PA (http://localhost:53984/odata/$metdata)

The way OData solves the relationshops between the data, like FoodId 1 is actually Sishi, is by using navigation properties. As shown in listing 6.10, the metadata contains information about what how to reach the actual food type or user for a specific food preferende. The following list will explain the meaning of the metadata further.

1 What entity, or PA this metadata applies to.

2 Food preferences are identified by an Id in the provided URI.

5-7 What information that this PA contains. These data are also strongly typed to be able to deserialize the JSON data.

8 Navigation property for food related to this food preference, and how to reach it.

9 Navigation property for user related to this food preference, and how to reach it.

Figure 6.10: The cloud for information about the food in food preference 5, which is Cathrine's food preference of sushi

Fidgure 6.10 illustrates an example of how these navigation properties can be used. As mentioned in section 6.3.1, the blue part of the URI is used to identify the cloud server and the green part is user to identify the food preferences service in the cloud. The red part of the URI however tells the cloud that we are searching for food preference 5, which is Cathrines

food preference, sushi, and that we would like to know what that food preference is. The result of this request is illustrated in listing **??** and shows that the returned result is sushi. The requesting service now know that Cathrine has a food preference of sushi, and can use this information with other open data stores to show the user suggested places to eat. This is the context aware user profile.

```
{
    "odata.metadata":"http://.../odata/$metadata#Foods/@Element", "Id":1,"Name":"Sushi"
}
```

Listing 6.11: Returned result to the requesting service when requesting more information from Cathrine's food preference

These navigation properties has been been created with the implementation and is created as shown in listing 6.12

```
public Food GetFood([FromODataUri] int key)
{
        serLikesFood u = context.UserLikesFoods.FirstOrDefault(f => f.Id == key);
        return u.Food;
}
```

Listing 6.12: Example of how to create navigation property between a food preference and the food description

6.3.5 Implementation result

in this chapter we have created a detail description of the implementation and the process from where a requesting service requests food preferences, how access to the users food preferences are authorized and the the authorized part of the user profile is sent back from the cloud. With the implementation covered in detail, we will move over to chapter 7 where we evaluate the approach of this thesis, the implementations scalability, privacy and security. We will also evaluate the selected technologies for the implementation and different choises that could have been made throughout the thesis and how this would impact the result.

Chapter 7

Evaluation

7.1 Approach

We are overall happy with our approach throughout this thesis. We chose to start with a scenario where a user walks into an airport, has time to spare before the flight leaves, and will present the user with useful information, related to their contextual information and preferences in the cloud. We chose to focus on the security part of this scenario, where users place a required security level on their on the policy attached to their individual information in the profile. We also chose to create a context-aware authorization model which services that require access to the users profile, need to gather this information from the users environment and authorize access to specific parts of the profile. Knowing that there were similar research involving this topic, and implementation, we chose to base our research on Microsoft technologies because Microsoft is such a big part of the enterprice world, and wanted to evaluate how this platform could cope with such an implementation using only technologies from this platform.

We are also quite happy with our research phase throughout the thesis. We spent most of our research effort on how contextual information information can be related to applications and security, concerning both authentication and authorization. Most of the research were able to retrieve involved how to evaluate contextual information through policies and how these policies can be linked to stand-alone resources, relational databases and semantic data structures. Before we started working on this thesis we had some experience with most of the .Net framework and Microsoft applications that has been used, but most research involving Microsoft technologies has been spent on the Open Data Protocol (OData) and policy languages available from this platform.

However, we had to change our approach during the implementation phase of the thesis, because we did not put enough effort on policy and OData research. We felt that we had a good foundational understanding of how Microsoft operates and when we therefore stumbled upon both OData and their policy language, WS-Policy, we thought this was a safe selection of technologies. Unfortunately off-the-shelf OData and WS-Policy, was

not what we expected when we started the implementation. We then had to stop our implementation phase and re-evaluate these choices of technologies. This resulted in a setback where we had to spend time going back and forth between the implementation and research phase. The change in this selection of policy language and OData implementation is covered in more detail in section 6.2. At the start of the implementation and partly because we chose the OData Web API in the implementation phase was because this enabled us to pass arguments with the request, what we realized while doing this was that this had to result in a POST message, and it broke the semantics. Therefore we also had to backtrack to our first idea and stick to using HTTP headers in with the request in order to send context information.

7.2 Implementation

Our implementation consist of the authorization process to a user profile, based on a request from a requesting service, which contains contextual information. We will evaluate our implementation considering scalability, security and privacy. We view these three as the most important parts of our implementation where scalability is the ability to expand on our implementation, security on how secure we view our implementation and privacy, where we considers the users privacy, and finally conclude on our implementation based on these evaluations. We will start by evaluating the scalability of our implementation.

Scalability

Our OData Web API implementation is what we view as the most scalable part of our implementation. The OData, which a front end for the database to display the data semantically, is implementing OData API interfaces, and most of these operations consist of few lines of code and is easy to edit and change behavior. We have tried to keep true to the OData structure as best as we can, and we feel we have managed that quite well. We also feel that the scalability of our policies is quite good. We have made extensions to the XACML.NET framework, but only just to make it work to both write policies, policy set, requests and evaluate. Therefore it remains in tact and is able to interpret all of the XACML math functions, multiple actions, subjects, resources as well as environment and override algorithms. We chose to put our own evaluation engine on top of this XACML.NET framework in order to simplify some of it's behaviors. This Evaluation engine can easily be changed to fit other needs. This means that by changing the evaluation engine and stick to using the XACML.NET framework or creating an own evaluation engine, one has all the abilities of XACML available.

We believe that the main challenge of scalability in our implementation lies in the selection of storage regarding the the user profile and trust attributes. These are stored as a relational database, where each table

has a corresponding class in the implementation. In order to extend functionality, one has to create a new table and a corresponding class. This leads to further work, where the database in worst case needs to be dropped and re-created with existing user data. There are however possibilities to create an own data store structure, which can be made compatible with Entity Framework (EF) by implementing an interface.

What we evaluate as the less scalable of the profile and trust is that, in the time of writing, the context attributes is evaluated statically, where each attribute is fetched from the request and evaluated. There is need for a more dynamic evaluation process of the trust attributes for it in order to be more scalable.

7.2.1 Security

We assume that trust and certificates is established between the requesting service and the cloud, and we do not consider security in firewalls etc. We think that security on an application level of our implementation is quite good. If the requesting service sends context information that the user has no relationship to, these will be ignored. We hare however aware of that this can lead to too much security. If a user has no relationship to a certain context attribute this can lead to a significant more lower trust level of the requesting service intended. Also, there is no data integrity check on the data provided by the requesting service. This means that in practice, the requesting service can send any information it wants to the cloud. For example, if the user is in The Democratic Republic of the Congo, which the user has defined with a trust level of 0.1, the requesting service can send context information of Country=Norway, which can lead to a much higher security level, on the other hand, if the user has no relationship with Norway, this attribute will be considered irrelevant. The requesting service cannot be evaluated as a request from Cathrine and use this trust level in order to get Bob's profile information. Both the contextual information and the requesting user is sent in the same request. And the same userId will be used for both of these processes.

To infiltrate policies which is stored on the file system, or needs administrative access to this server.

7.2.2 Privacy

We think it is hard to evaluate on the privacy, because the responsibility solely relies on the user. The user defines their trust of different contexts and set security levels on their profile information. However, we believe that there might be a need for a plan B. For example, if users chooses to set a security level of 0.1 on their payment information, we think that the system should react to such actions to "protect the user from itself" by providing help or guidelines, maybe even a threshold for certain kind of data. We will cover this future work in more detail in section 8.4.

7.2.3 Conclusion

Based on the above evaluations on Scalability, Security and Privacy, we will give a score to each of these with a conclusion on what is the best part of our implementation and what needs more work. Score for each of these will be given between 1 and 5.

Scalability	Security	Privacy
4	3	3

We have given scalability a score of 4 because it keeps true to the scalable setup of OData and the selection of storage is more difficult to expand or improve. Privacy with a score of 3 because the system does not give access across profiles with other users trust attributes, but it does not have data integrity check. Privacy has been given a score of 3, because it gives the user the ability to secure their data the way they see fit, but we believe there is a need of system action to help the user to protect itself from setting too low security threshold on information about themselves that is more secure in nature. We are however aware that this is also a matter of opinion.

7.3 Technologies

In this section we will talk about the technologies that was selected for implementing a context aware authorization model and evaluate how these were able to help us achieve our goal.

We wanted to return the authorized user profile semantically to the requesting service, which OData does a good job of doing. This is one of the core functions of OData and is pretty straight forward to implement even with the OData Web API that we have selected for this implementation, although it takes some time to get into the idea of how the protocol works.

The most challenging part of the implementation was how to help OData and XACML to collaborate. In our implementation we decided to create the profile structure hierarchically according to the structure of the user profile, where a OData resource is linked to an XACML policy. This approach in itself is a good idea, but from our knowledge, there seems to be no intuitive way of putting this logic into practice in the cloud service. By keeping to the strict guidelines of the OData interface and at the same time having to evaluate the policies, this tent to require some solutions that is viewed by us to be workarounds. An other approach could be to store the policy name along with the OData entities, but to us this seems illogical because the policies in XACML defines the resource, in our case the OData entity, and for us it was the logical step to keep this as part of the policy.

As we mentioned in our implementation in section 6.3.1, by using the OData Web API, there is no intuitive way of sending context information information with the request to an OData service. OData is a web service and is therefore restricted to the usage of the HTTP protocol and with HTTP GET only being the viable option and GET parameters reserved for other

purposes, such as query options, only headers remain for sending context information. These headers are key/value stores which limits how the data can be passed with the request. A solution for this could be to send the key/value as JSON data string and deserialize this string to objects in the code.

Chapter 8

Conclusions and future work

The goal of the thesis is to look at context-aware access control and how it is linked to semantically structured data in a business context, as supplied through the OData framework. The use of context-aware authorization mechanisms answers the need for a granular access system, where the traditional false/true authorization mechanisms are no longer sufficient. As an easy-to-catch scenario, we chose a travel scenario providing the user with context-aware and personalized information. The scenario takes place in an airport where the users has time to spare before their flight leaves and wants to get suggested places to eat and known people to interact with. This is done by having the users mobile device connect automatically to a hotspot and sends the context information to a service that will request relevant information from the user profile stored in the cloud service. We focused on the authorization part of the scenario, to implement a context-aware authorization model in enterprise systems, using as much as possible enterprise oriented solutions.

We achieved this functionality through a semantically structured user profile connected to cloud services, where every node is secured by a policy which describes in what context that data should be available. The requesting service forwards the context information gathered from the users mobile phone, followed by a request, to the cloud. The cloud will evaluate the context against relevant information from the user profile and the security set in the user profile and authorize access.

The novel aspects of this thesis are as follows:

- Service scenario for context-aware personalized services.
- Applicability by the Microsoft OData framework.
- A policy engine supporting the semantically structured data, and an implementation of the core components into a prototypical implementation.

8.1 Conclusions

To reach the goal addressed in this thesis we created a travel scenario where user get personalized information while being abroad, where information about the user profile is stored semantically in a cloud service and protected by policies. Depending on the users current context, service providers are provided with parts of that users profile.

Our functional architecture consists of a combination of the mobile phone as the identifier, public information screen as the interface for service delivery, a protected cloud infrastructure carrying a context-aware user profile and service provider entry, being able to deliver services according to the context-aware profile.

The functional description has the novelty of implementing required trust levels for the access of the user profile preferences, e.g. a trust level in the area from 0.0 to 1.0, for the access of the information. Access to the profile is only given if the context-aware trust are equal or greater than the required security attribute, for accessing the information. We have exemplified the context-aware trust through two people, Bob and Cathrine, where Bob is a restrictive person and Cathrine is an open person. The two users define their trust attributes, for example, Bob gives a security attribute of 0.8 to his food preferences, while Cathrine gives a security attribute of 0.3 to her food preferences. A trust level is composed by a set of attributes, given by the user confidence in the current environment. As an example, Norway is seen as a more trusted country with a high trust level of 0.8, while Afghanistan has a low trust level of 0.3. The trust level is enhanced by specific information of known places, like a hotel that has been visited before, that increases or reduces the trust level based of that situation. Thus our functional architecture compares the achieved trust values for a certain location with the security attribute needed to access parts of the the user profile and then gives the service provider access to those preferences, for example a food supplier at Gardermoen may ask the information on Bob and Cathrine, and as Bob is more restricted, he will only get information on seafood will be presented, where as Cathrine will get much more personalized information.

Our main focus of this thesis was on the implementation using enterprise frameworks, though semantic data, reasoning, policies are well known in the open source world. In an enterprise environment, based on Microsoft OData for implementing context-aware authorization model, the toolboxes are not present on the market, so this thesis has crated a methodology and applied that methodology to create the context-aware authorization model for personalized services.

The created proof-of-concept implementation for a context-aware authorization framework consists of the following main components:

- OData
- Entity Framework
- XACML policy language

- XACML.NET policy engine

Our experience is that, though Microsoft provides framework for semantic data representation, we have not found evidence for the successful applicability of ontologies in the framework. Entity Framework provides good support database abstraction, although it is restricted to database terminology, resulting in complexity when semantic functionality is desired. Our research shows that XACML is a flexible policy language, capable of creating complex rules that is applicable to semantic resources, but has an overhead of information for specifying simple rules. Our research and implementation experience shows that a XACML policy engine framework applicable to the .NET platform is non-existent, though the XACML.NET policy engine, which is a work in progress, shows interest for such policies in the enterprise market.

In an overall conclusion, this thesis has shown that the enterprise framework is applicable to create context-aware and personalized services, and we have demonstrated the core components being OData and XACML.

The next section is going to provide recommendations based on our findings.

8.2 Recommendations

8.2.1 Lessons learned

Our evaluation after researching the topic of creating context-aware authorization model for semantically structured data, focusing on enterprise solutions is that we are concerned about the lack of available in the enterprise world. We believe that the best approach is to use a hybrid solution consisting both of Microsoft and Java technologies. There are no fully working engine that is available on the .NET platform for evaluating these policies and no concept of reasoning on their semantically structured data. For both of these topics there are good open source tools available, which is continuously improved by big companies such as Sun and IBM.

With our experience in this thesis believe that using these hybrid solutions can be the best choice for addressing the goal of this thesis.

8.2.2 Inter working amongst semantic systems

Even if OData and RDF has different implementations, they have the same goal, to expose data freely on the web with semantically structured data[2]. Structuring data in a semantic way means to make data accessible, where a machiene can understand what the data actually is described in human-readable form. This is done by exposing the data atomically into small sets of data which links to other data and have it's own distinct URI. What this actually mean is that for example sushi has it's own identifier and will always mean the same type of object.

Both Microsoft and W3C has done extensive research on how they can make these two standard collaborate[2][24]. Both of their implementation

describes on how to convert data retrieved from a SPARQL[1] endpoint and expose these as OData, however, none of these implementations, and our research so far, explains hos to convert the other way around, namely OData to RDF.

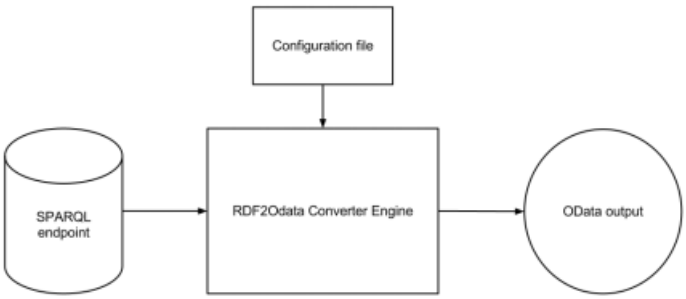

Figure 8.1: Flow diagram of how RDF can be converted into OData

As we can see from figure 8.1, the core of the research done by Microsoft and W3C is to create a converter engine between the SPARQL endpoint and OData. When a request is sent to an OData service the query will be sent to a conversion engine that will run that query on the SPARQL endpoint. To help the conversion engine on how this rewrite this query to SPARQL, it is provided by a configuration file, which describes what OData query pars have equivalents to SPARQL and vise versa. Figure 8.2 shows an example of a configuration file entry. OData is represented in Atom XML, but is strongly typed. This example shows how age is converted from RDF to OData as a 32-bit integer.

```
'map': {
        'http://example.com/age': {
                'name': 'age',
                'prop': {
                        'type': 'Edm.Int32',
                        'nullable': 'true'
                }
        }
}
```

Figure 8.2: Snipping from configuration file that describes how to convert RDF into OData

When the query reaches the SPARQL endpoint, it will run the converted query on the RDF triple store and return the data back to OData via the converter engine. Both of these researches and implementations currently supports basic RDF to OData conventions. There are several OData

[1]Query language used to query RDF stores

features that will also need to be implemented to make a complete and full-working convention to OData. Among these are relationships, schematic changes, but also features like filtering, paging and select.

8.3 Experience with OData implementations

We have throughout the thesis been working with institutions that has a lot of data available they wish to expose to the public. Among these are the Smithsonian National Museum of Natural History (NMNH), located in Washington D.C, United States. They have been working on a cross-field anthropology database for years and are currently working to publish a front-end that can be used by other researches, students or the public at large to gain access to their data in an intuitive way. While creating this front-end they realize that every researcher works differently and may be interested in other parts of the data that may not be exposed at first glance or would like the data structured in a different kind of way. They do however currently provide users with the ability to export a set of data from this database, but there may still be large amounts of data that is needed and considering these data may change on a daily basis, there is a need for a more dynamic solution where other researchers can view these data in a way they see fit.

We introduced the project leaders on this database to the OData protocol. If they expose their data using this protocol, other researchers has this ability, by either viewing the data in a browser, by writing URL's manually with their own queries or even by creating their own front end to suit their needs. OData has extensive support for multiple programming languages so that other developers can use the language they see fit, and always get the latest data. Although in the start phase of the project, NMNH is currently working on exposing their database using OData, and have already spent an amount of hours preparing the database for such an implementation.

8.4 Open issues and future work

8.4.1 Usability

We believe that the next natural step for development before our implementation can be put to use is a front end, an app, web page or others, the user can interact with to manage their profiles. We are not going to go into detail about our vision for this front end, but rather talk about usability. If there were to be created a front end directly on top of our implementation, we believe that there is a need for a simplification for delegating security level to policies attached to statements in the user profile. We do not believe that it is intuitive for users to define a security level between 0.0 and 1.0. We can still keep true to this selection of security attributes, but we think there is a need of simplifying the meaning of these numbers. Our suggestions are *Green*, *Yellow* and *Red*.

An other suggestion concerning usability, and to some extend, security as well. When users create policies that are attached to their data that is by nature more secure, we would like the system to help the users by "protecting them from themselves", meaning that if a user set security level of 0.1 to information that is more vital, recommend system to react to such actions. We suggest an initialization process when the users create their profile, to either play a game, create example policies or answer questions,to set their default security level concerning different topics, e.g. food or credit card.

8.4.2 Security and data integrity

Our approach was focused on on creating a context-aware authorization for semantically structured data. The implementation has focus on usability of the framework components and not the evaluation of security, though we are aware of the need of trust between services, certificates, and other mechanisms such as firewalls and address restrictions.

An other aspect we consider to be an important part of our model is data integrity for incoming requests to the cloud. The cloud should have the ability to check that the user the service is requesting information for is actually the requesting user, as well as checking that the contextual information is correct. Thus far we have assumed that there is integrity check on the data.

8.4.3 Reasoning

We mentioned earlier that when a user creates a profile, the user will start an initialization process for this profile and that this profile should expand based on reasoning. From our example the user will create a profile and create some of their food preferences. When the requesting service receives this information it should be able to, together with other open data stores reason on what other food preferences this use might have. OData however, does currently have, or as far as we know, will never have, support for reasoning. Considering that it is unlikely for OData to ever have reasoning capabilities, we think that the best option for achieving this functionality is by performing this operation with a semantic reasoner, like Jena[2] or Pellet[3]. We will cover interworking between OData and the Semantic Web in more detail in section 8.2.2.

[2]http://jena.apache.org/, accessed October 2013
[3]http://clarkparsia.com/pellet/, accessed October 2013

Bibliography

[1] G. D. Abowd, A. K. Dey, Brown P. J., Davies N., Smith M., and Steggles P. *Twoards a Better Understanding of Context and Context-Awareness*, volume 1707. Springer, 1999.

[2] Ahmed and Moore. Sparql / odata interop. Technical report, W3C, 2013.

[3] Anne Anderson. A comparison of two privacy policy languages: Epal and xacml. 2005.

[4] Berners-Lee. Uniform resource identifier (uri): Generic syntax. *Network Working Group*, 2005.

[5] Berners-Lee, Hendler, Lassila, and Ora. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[6] Tim Berners-Lee. Linked data. Electronic, August 2006.

[7] Bhatti, Bertino, and Ghafoor. A trust-based context-aware access control model for web-services. *Distributed and Parallel Databases - DPD*, 18:83–105, 2005.

[8] Rafae Bhatti, Elisa Bertino, and Arif Ghafoor. A trust-based context-aware access control model for Web-services. In *International Conference on Web Services*, 2004.

[9] Rafae Bhatti, Daniel Sanz, Elisa Bertino, and Arif Ghafoor. A Policy-Based Authorization Framework for Web Services: Integrating XGTRBAC and WS-Policy. In *International Conference on Web Services*, pages 447–454, 2007.

[10] Paul Biron, Ashok Malhotra, World Wide Web Consortium, et al. Xml schema part 2: Datatypes. *World Wide Web Consortium Recommendation REC-xmlschema-2-20041028*, 2004.

[11] Tim Bray, Jean Paoli, M. Sperberg-McQueen, Eve Maler, Yergeay, and John Cowan. Extensible markup language (xml), August 2006.

[12] Marijke Coetzee and Jan H. P. Eloff. *A Trust and Context Aware Access Control Model for Web Services Conversations*. 2007.

[13] Antonio Corradi, Rebecca Montanari, and Daniela Tibaldi. Context-based access control for ubiquitous service provisioning. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pages 444–451. IEEE, 2004.

[14] Luca Costabello, Serena Villata, Nicolas Delaforge, Fabien Gandon, et al. Linked data access goes mobile: Context-aware authorization for graph stores. In *LDOW-5th WWW Workshop on Linked Data on the Web-2012*, 2012.

[15] Anind K. Dey, Gregory D. Abowd, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Twoards a better understanding of context and context-awareness. *Lecture Notes in Computer Science*, 1707:304–307, 1999.

[16] Fielding, Irvine, Gettys, Mogul, Manister, Leach, and Berners-Lee. Hypertext transfer protocol – http/1.1. Electronically, June 1999.

[17] Organization for the Advancement of Structured Information Standards. Open data protocol (odata). Electronically, November 2013.

[18] Gregorio and de hOra. The atom publishing protocol. Electronically, October 2007 2007.

[19] Weili Han, Junjing Zhang, and Xiaobo Yao. Context-sensitive Access Control Model and Implementation. In *Conference on Computer and Information Technology*, pages 757–763, 2005.

[20] Heumann. Tips for writing good use cases. *Transforming software and systems delivery*, May 2008.

[21] Junzhe Hu and Alfred C. Weaver. A Dynamic Context-Aware Security Infrastructure for Distributed Healthcare Applications.

[22] Hulsenbosch, Salden, Bargh, Ebben, and Reitsma. Context sensitive access control. *Proceedings of the tenth ACM symposium on Access control models and technologies*, 1:111–119, 2005.

[23] JinYuan and Tong. Attributed based access control (abac) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, volume 1. IEEE Computer Society, 2005.

[24] Kerrin, Hausenblas, Pizzo, Viegas, and Wilson. Linking structured data. Technical report, Microsoft, 2012.

[25] Devdatta Kulkarni and Anand Tripathi. Context-aware role-based access control in pervasive computing systems. In *Symposium on Access Control Models and Technologies*, pages 113–122, 2008.

[26] Sven Lachmund, Thomas Walter, Laurent Gomez, Laurent Bussard, and Eddy Olk. Context-aware access control making access control decisions based on context information. In *Mobile and Ubiquitous*

Systems-Workshops, 2006. 3rd Annual International Conference on, pages 1–8. IEEE, 2006.

[27] Oscar García Morchon and Klaus Wehrle. Modular context-aware access control for medical sensor networks. In *Symposium on Access Control Models and Technologies*, pages 129–138, 2010.

[28] M. Nottingham and R. Sayre. The atom syndication format. Electronically, December 2005.

[29] Pung and Zhang. Web-services-based infrastructure for context-aware applications. *Pervasive Computing, IEEE*, 3:66–74, 2004.

[30] Sara Radicati and Quoc Hoang. Microsoft exchange server and outlook market analysis, 2012-2016. 2012.

[31] Arjmand Samuel, Arif Ghafoor, and Elisa Bertino. Context-Aware Adaptation of Access-Control Policies. *IEEE Internet Computing*, 12:51–54, 2008.

[32] Gerhard Skagstein. *SystemutvSystem - fra kjernen og ut, fra skallet of inn*, volume 2005. Høyskoleforlaget, 2005.

[33] Henry S Thompson. Xml schema part 1: Structures second edition, 2004.

[34] Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *The Semantic Web-ISWC 2006*, pages 473–486. Springer, 2006.

[35] Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *The Semantic Web-ISWC 2006*, pages 473–486. Springer, 2006.