Ubiquitous Semantic Applications

Der Fakultät für Mathematik und Informatik
der Universität Leipzig
eingereichte

DISSERTATION

zur Erlangung des akademischen Grades

Doktor-Ingenieur
(Dr. Ing.)

im Fachgebiet Informatik

vorgelegt

von **Dipl.-Ing. Timofey Ermilov**

geboren am 1. Januar 1987 in Murmansk, Russland

Die Annahme der Dissertation wurde empfohlen von:
1. Prof. Dr. Klaus-Peter Fähnrich, Universität Leipzig
2. Prof. Dr. Amit Sheth, Wright State University

Die Verleihung des akademischen Grades erfolgt mit Bestehen der
Verteidigung am 18.12.2014 mit dem Gesamtprädikat
magna cum laude.

Bibliographic Data

Title: Ubiquitous Semantic Applications
Author: Timofey Ermilov
Institution: Universität Leipzig, Fakultät für Mathematik und Informatik
Statistical Information: 151 pages, 49 figures, 5 tables, 119 literature references

Thesis Summary

As Semantic Web technology evolves many open areas emerge, which attract more research focus. In addition to quickly expanding Linked Open Data (LOD) cloud, various embeddable metadata formats (e.g. RDFa, microdata) are becoming more common. Corporations are already using existing Web of Data to create new technologies that were not possible before. Watson by IBM an artificial intelligence computer system capable of answering questions posed in natural language can be a great example.

On the other hand, ubiquitous devices that have a large number of sensors and integrated devices are becoming increasingly powerful and fully featured computing platforms in our pockets and homes. For many people smartphones and tablet computers have already replaced traditional computers as their window to the Internet and to the Web. Hence, the management and presentation of information that is useful to a user is a main requirement for today's smartphones. And it is becoming extremely important to provide access to the emerging Web of Data from the ubiquitous devices.

In this thesis we investigate how ubiquitous devices can interact with the Semantic Web. We discovered that there are five different approaches for bringing the Semantic Web to ubiquitous devices. We have outlined and discussed in detail existing challenges in implementing this approaches in section 1.2. We have described a conceptual framework for ubiquitous semantic applications in chapter 4. We distinguish three *client approaches* for accessing semantic data using ubiquitous devices depending on how much of the semantic data processing is performed on the device itself (thin, hybrid and fat clients). These are discussed in chapter 5 along with the solution to every related challenge. Two *provider approaches* (fat and hybrid) can be distinguished for exposing data from ubiquitous devices on the Semantic Web. These are discussed in chapter 6 along with the solution to every related challenge. We conclude our work with a discussion on each of the contributions of the thesis and propose future work for each of the discussed approach in chapter 7.

Publications

This thesis is based on the following publications and proceedings, in which I have been author, editor or contributor. *At the respective chapter and section*, I included the references to the appropriate publications.

Proceedings, (co)-edited

- Proceedings of the 8th extended semantic web conference on The semantic web: research and applications (ESWC 2011). [Tramp et al., 2011a, Ermilov et al., 2011b]
- Proceedings of the third Russian conference on Knowledge Engineering and Semantic Web (KESW 2012). [Ermilov et al., 2012]
- Proceedings of the 15th International Conference on Information Integration and Web-based Applications & Services
  (iiWAS 2013). [Ermilov and Auer, 2013]

Journal Publications, peer-reviewed

- Ubiquitous Semantic Applications: A Systematic Literature Review [Ermilov et al., 2014].

Acknowledgments

I would like to thank all the colleagues with whom we jointly edited the proceedings and journal papers: Sebastian Tramp, Norman Heino, Ali Khalili, Michael Martin, Philipp Frischmuth, Jens Lehmann, Saeedeh Shekarpour, Quan Nguyen, Natanael Arndt, Daniel Gerber and Sören Auer.

I would like to thank our colleagues from the AKSW research group for their helpful comments during the development of this thesis. This work was partially supported by a grant from the European Union's 7th Framework Programme provided for the project LOD2 (GA no. 257943). Special thanks go to Amrapali Zaveri, Michael Martin, Jörg Unbehauen and Ali Khalili as well as our amazing post-docs of AKSW – Axel, Jens and Thomas.

I would like to thank Prof. Fähnrich for his scientific experience with the efficient organization of the process of a PhD thesis. Furthermore, I would like to thank Dr. Sören Auer for his continuous help and support.

Contents

1. Introduction

1.1. Introduction and motivation

Tim Berners-Lee defines the Semantic Web as "an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [Berners-Lee et al., 2001]. As Semantic Web technology evolves many open areas emerge, which attract more research focus. In addition to quickly expanding Linked Open Data (LOD) cloud (as shown in Figure 1.1[1]), various embeddable metadata formats (e.g. RDFa[2], microdata[3]) are becoming more common. Corporations are already using existing Web of Data to create new technologies that were not possible before. Watson[4] by IBM an artificial intelligence computer system capable of answering questions posed in natural language [Ferrucci et al., 2013] can be a great example.

On the other hand, ubiquitous devices (i.e. devices that are available to user anywhere and at any time) that have a large number of sensors and integrated devices are becoming increasingly powerful and fully featured computing platforms in our pockets and homes. For many people smartphones and tablet computers have already replaced traditional computers as their window to the Internet and to the Web. According to Cisco's Global Mobile Data Traffic Forecast Update [Index, 2013] by the end of 2013, the number of mobile-connected devices will exceed the number of people on earth. The report also predicts that the average mobile connection speed will surpass 1 Mbps in 2014. In addition, monthly global mobile data traffic is expected surpass 10 exabytes in 2017. Hence, the management and presentation of information that is useful to a user is a main requirement for today's smartphones. And it is becoming extremely important to provide access to the emerging Web of Data from the ubiquitous devices. Today, ubiquitous devices account for 21% of global Web usage[5] and that number keeps growing. In parallel, consumer grade ubiquitous devices are becoming more powerful and less expensive. Currently 96% of the world is subscribed to mobile services[6].

Furthermore, as a result of the computational power increase in ubiquitous

[1] Images taken from `http://lod-cloud.net/`

[2] `http://www.w3.org/TR/rdfa-syntax/`

[3] `http://schema.org/`

[4] http://www-03.ibm.com/innovation/us/watson/

[5] According to StatCounter statistics as of December 2013 `http://gs.statcounter.com/#mobile_vs_desktop-ww-monthly-201212-201312`

[6] `http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#subscribers`

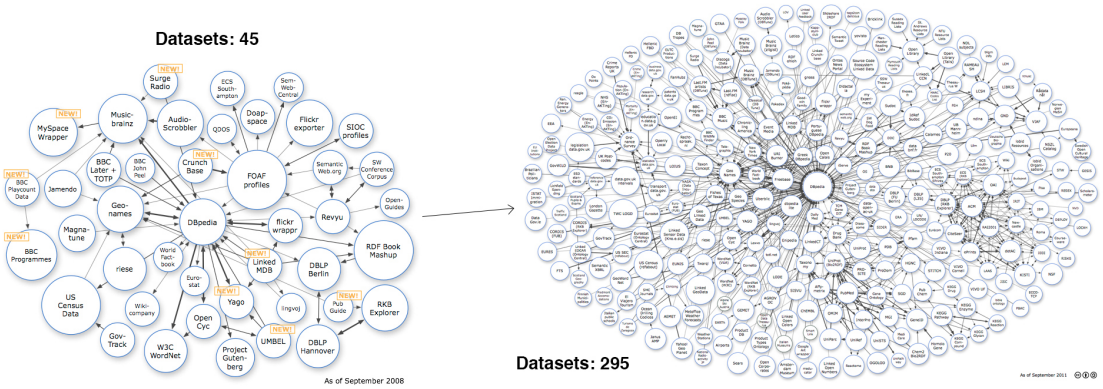


Figure 1.1.: Growth of the Linked Open Data (LOD) cloud from September 2008 (left) to September 2011 (right).

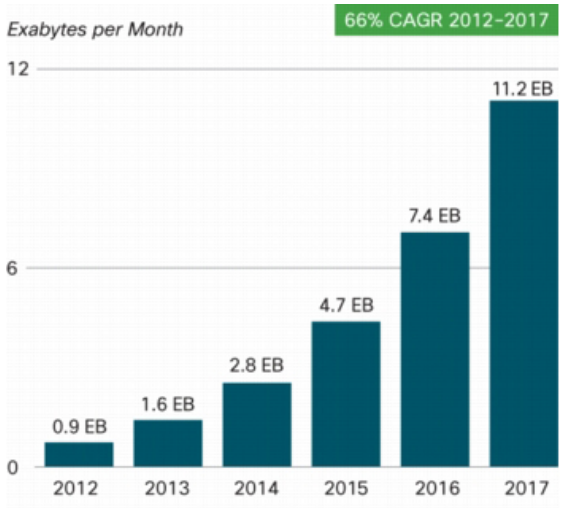


Figure 1.2.: Mobile traffic growth from 2012 to 2017. [Index, 2013]

devices, the *Internet of Things* is becoming more important. The term *Internet of Things* [Atzori et al., 2010] refers to the vision that all kinds of physical objects are uniquely identifiable and have a virtual representation on the Internet. Increasingly, some form of intelligence is either embedded into the object itself (e.g. by integrating a system on a chip into a TV set or manufacturing equipment) or the object itself is a smart device (e.g. a smartphone or tablet PC). As a result, these devices can not only identify but also describe themselves by providing comprehensive information. However, as the Web of Documents is meanwhile complemented by a Web of Semantic Data, information provided on the Internet of Things should be made available in standardized and semantically structured form as well.

In this thesis, we investigate how ubiquitous devices can interact with the Semantic Web. We discovered that there are five different approaches for bringing

the Semantic Web to ubiquitous devices. We distinguish three *client approaches* for accessing semantic data using ubiquitous devices depending on how much of the semantic data processing is performed on the device itself (thin, hybrid and fat clients). These are discussed in chapter 5. Two *provider approaches* (fat and hybrid) can be distinguished for exposing data from ubiquitous devices on the Semantic Web.[7] These are discussed in chapter 6.

1.2. Challenges

Ubiquitous devices are playing an increasingly important role for information access. But there is a set of challenges that are required to be solved before the Semantic web can be used on the ubiqtuious devices.

1.2.1. Data exchange challenges

There are two important aspects that are to be considered for ubiquitous semantic applications: (a) fetching data from the Web of Data and displaying it to the end user and (b) providing data from the ubiquitous device to other clients. In case of fetching and displaying the data, there are currently three approaches for creating clients:

- *fat*: all data storage and processing is performed on the client,
- *hybrid*: data is stored and processed partially on the client and on the Web of Data,
- *thin*: data is only accessed and visualized by the client, but no actual data is stored on the client.

In case of providing the data from the ubiquitous devices, there are currently two approaches:

- *fat*: all data is stored and provided directly from the ubiquitous device,
- *hybrid*: data is stored and provided by the ubiquitous device as well as by the online server containing the data replica.

Capabilities replication. In case of fat client approach, the ubiquitous application stores all the retrieved data on the user's device. This allows full access to the data even in cases when there is no data connection available or when the original data provider is offline. The main problem of fat client approaches, in development of ubiquitous semantic applications, is to provide the same capabilities as the original data provider (e.g. SPARQL endpoint, different data serializations etc.).

[7] Note that there can not be a thin provider approach where no data is stored on the device.

Offline functionality. In addition to the fat client approach, hybrid and thin client approaches exist. Those two approaches are pretty close to each other in terms of means of development. As (comparatively) powerful ubiquitous computing devices are becoming more common, ubiquitous web applications have started gaining popularity. Some of the existing web applications already use the Semantic Web technologies and information in the form of Resource Description Framework[8] (RDF) (e. g. TripIt[9]). An important feature of these applications is their ability to provide *offline functionality* with local updates for synchronization later with a web server. In hybrid client applications this can be achieved in an easier way than in thin client applications using the same approaches that work in fat client applications with some adjustments.

Data conversion. In case of providing data from the ubiquitous device itself, one of the challenges is converting existing structured data on on the device to the RDF. It is important to provide a way to map existing structured data to the vocabularies and ontologies, thus making it possible to expose this information as dereferencable RDF.

Power consumption. A particular specific requirement when dealing with smart and embedded devices are resource constraints. Due to progress in miniaturization, memory and processing power is meanwhile not a constraining factor anymore for most applications. Power consumption on the other hand is a key aspect, when equipping devices with additional functionality.

1.2.2. Platform fragmentation

Another challenge application developers for ubiquitous devices face currently is the plethora of development platforms as well as the incompatibilities between them. *Android* (Google), *iOS* (Apple), *Blackberry OS* (RIM), *WebOS* (HP/Palm), *Symbian* (Nokia) are popular and currently widely deployed platforms, with many more proprietary (e.g. *Sailfish OS*) and open (e.g. *Firefox OS*) ones being available or developed as well. As shown on Figure 1.3 the popularity of a platform can change very quickly. For example, Android has overtaken the iOS as a dominant system in just a bit more than two years. Because of that most of the developers who were working on applications for the dominating platform (iOS in this case) exclusively, at that point need to start learning a new programming language and environment in order to bring their applications to the new most popular platform (Android).

[8] `http://www.w3.org/RDF/`
[9] https://www.tripit.com/

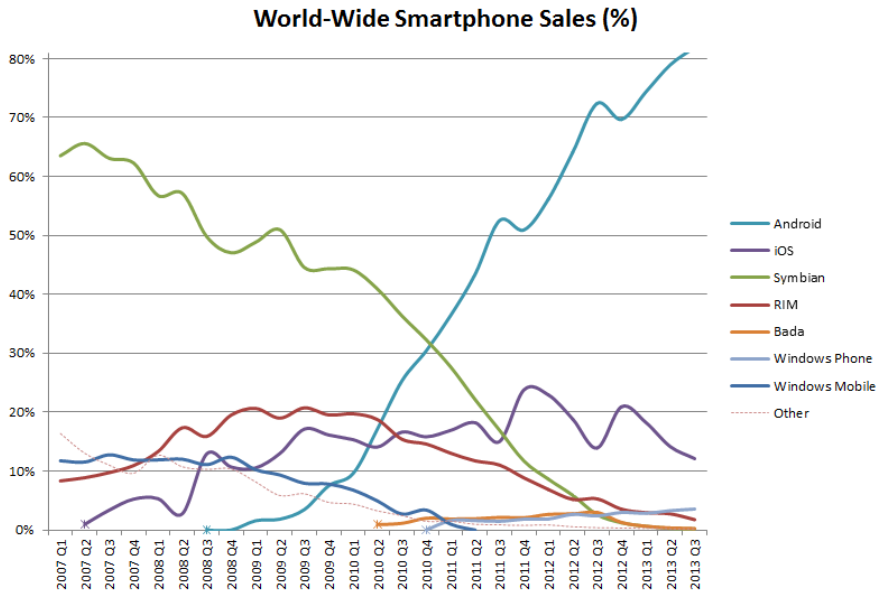


Figure 1.3.: World-Wide smartphone sales by operating system. [Gartner, 2013]

1.2.3. Reconciliation and data ownership

In addition, despite the success of applying the Semantic Web in development of applications for ubiquitous devices several problems exist:

- The key problem in ubiquitous web applications is the reconciliation, i. e. the problem of potentially *conflicting updates* from *disconnected clients*.
- As a consequence of the development platform fragmentation, realizing a special purpose application, which works with many or all of these platforms is extremely time consuming and inefficient due to the large amount of duplicate work required.

On other hand, there are provider applications and approaches for creating such applications. Online social networking became one of the most popular services on the Web. Making management and presentation of information about contacts, social relationships and associated information are one of the few main requirements and features of today's smartphones.

The problem of managing and accessing user's social information is currently solved solely for centralized proprietary platforms (such as Google mail, contacts & calendar) as well as data-silo-like social networks (e.g. Facebook). As a result of this data centralization, users' data is taken out of their hands; users are dependent of the infrastructure of a single provider, they experience a lock-in effect. Once the data is published, users also lose control about the data they own, since it is stored on a single server. Interoperability between platforms is rare and limited to proprietary APIs and third-party applications that allow it. Since there are only a few large players in social networking, the Web partly loses its distributed nature.

Many people argue that social networks should allow users to keep a control over their own data. The users should as well be able to host the data on an infrastructure, which is under their direct control (e.g. their ubiquitous device). A possibility to give the control over their data back to the users is the realization of a truly distributed social network.

Within the Semantic Web there are already a number of standards and best-practices for social, Semantic Web applications such as *FOAF*, *WebID* and *Semantic Pingback*. However, there is no comprehensive strategy, how these technologies can be combined in order to create an open and distributed social network; or how can they be used efficiently in a ubiquitous environment.

In this thesis, each of the aforementioned problems is discussed in detail, and solution to each one is proposed and discussed.

1.3. Contributions

This thesis proposes approaches to tackle each of the five aforementioned challenges.

Regarding the first challenge mentione in section 1.2.1, i.e. the challenge of making the information accessible on the device using fat client approach available while providing same set of features as the original data provider, we developed the *Mobile Semantic Social Web client* (MSSW). The main motivation behind this is to create a ubiquitous Social Semantic Web framework, which could weave a distributed social network based on semantic technologies.

Overall, we have accomplished the following contributions:

1. developed an architecture for making mobile devices endpoints for the Social Semantic Web

2. implementation of the architecture for the Android platform

Apart from the fat client approach, we also devised a hybrid client approach that allows creation of the platform independent Distributed Semantic Social Network (DSSN) client. As an example implementation we created a *Mobile DSSN Client*. The main motivation behind this is to provide access to the DSSN from as many platforms as possible while maintaining the same set of features that is provided by the fat client approach. With this approach we have tackled the challenges mentioned in section 1.2.1 and subsection 1.2.3.

Overall, we have accomplished the following contributions:

1. developed an architecture for platform agnostic mobile client that fits into the DSSN architecture

2. defined requirements in order to make the client part of the DSSN and make it compatible with the widest variety of devices possible

3. described the platform independent implementation of the client
4. discussed platform specific aspects that allow to enhance functionality of the implementation when needed

With regard to the challenges of thin client approach (i.e. section 1.2.1 and subsection 1.2.3), we developed an *OntoWiki Mobile* approach realizing a mobile semantic collaboration platform based on the OntoWiki framework [Heino et al., 2009]. The main motivation behind this is to provide a generic, application domain agnostic tool, which can be utilized in a wide range of very different usage scenarios ranging from instance acquisition to browsing of semantic data on the go.

Basically, the OntoWiki Mobile has the following features:

1. comprises specifically adopted user interfaces for browsing, faceted navigation as well as authoring of knowledge bases
2. allows users to collect instance data and refine the structured knowledge bases on-the-go
3. implemented as an *HTML5 web application*, thus being a completely platform independent ubiquitous device
4. allows offline use in cases with restricted network coverage by using the novel HTML5 local storage feature for replicating parts of the knowledge base on the ubiquitous device
5. has the advanced conflict resolution strategy for RDF stores based on a combination of the EvoPat [Rieß et al., 2010] method for data evolution and ontology refactoring along with a versioning system inspired by distributed version control systems like Git[10]

Regarding the challenge of exposing the information from the ubiquitous device (as discussed in section 1.2.1), i.e. turning the device into a fat provider, we developed an approach for equipping embedded and smart devices with a Linked Data interface *Embedded Linked Data Server*. The main motivation behind this is to enable any smart device (e.g. tablets, smart phones, TVs) to identify and describe itself by providing comprehensive information in accordance with the Linked Data principles.

Overall, we have accomplished the following contributions:

1. developed an approach for equipping embedded and smart devices with a Linked Data interface
2. implemented the approach for Android devices with a particular focus on the impact of power consumption

[10] `http://git-scm.com/`

3. developed a performance vs. power benchmarking methodology
4. evaluated the approach using this methodology and shown, that the overhead introduced by equipping a device with a Linked Data interface is neglectable given modern software and hardware environments and moderate usage

Apart from the fat provider approach, we also devised a hybrid provider approach for equipping embedded and smart devices with a Linked Data interface capable of providing data even when the device itself is offline. The main motivation behind this is to enable any smart device (e.g. smart phones, robots) to identify and describe itself by providing comprehensive information in accordance with the Linked Data principles even during poor connectivity or absence of such.

Overall, we have accomplished the following contributions:

1. developed an approach for equipping embedded and smart devices with a Linked Data interface capable of providing data even when the device itself is offline
2. developed a technical architecture for the approach
3. discussed possible implementation of the approach

1.4. Chapter Overview

As indicated in Figure 1.4, the chapters and sections of this thesis are arranged as follows.

Chapter 2 introduces the concepts of the Semantic Web and its associated technologies in context of ubiquitous devices, which constitutes the basic scientific background required for the reader to understand the thesis. The chapter starts by defining the Semantic Web followed by discussing the Resource Description Framework (RDF), its components and why is it important for ubiquitous devices. The chapter continues to explain the various RDF serialization formats (e.g. N-Triples) and the differences among them. Then, it moves to the crucial topic of Semantic Web, specifically the ontology and the various languages that can be uses to develop the ontologies. Thereafter, the SPARQL query language, triplestores and how they support the SPARQL language is described.

Chapter 3 gives an overview of the State of the Art in the area of ubiquitous semantic applications. First, the research method and the search strategy used to find relevant studies is described. Then, the methodology of the extraction and analysis of data is detailed. Furthermore, the chapter describes results of the search by defining generally used terminology, outlining possible user roles, describing common development approaches and defining quality attributes that help in evaluation of ubiquitous semantic applications. Finally, six separate ubiquitous semantic applications are discussed in-depth and concluded with an outline of research and technology challenges.

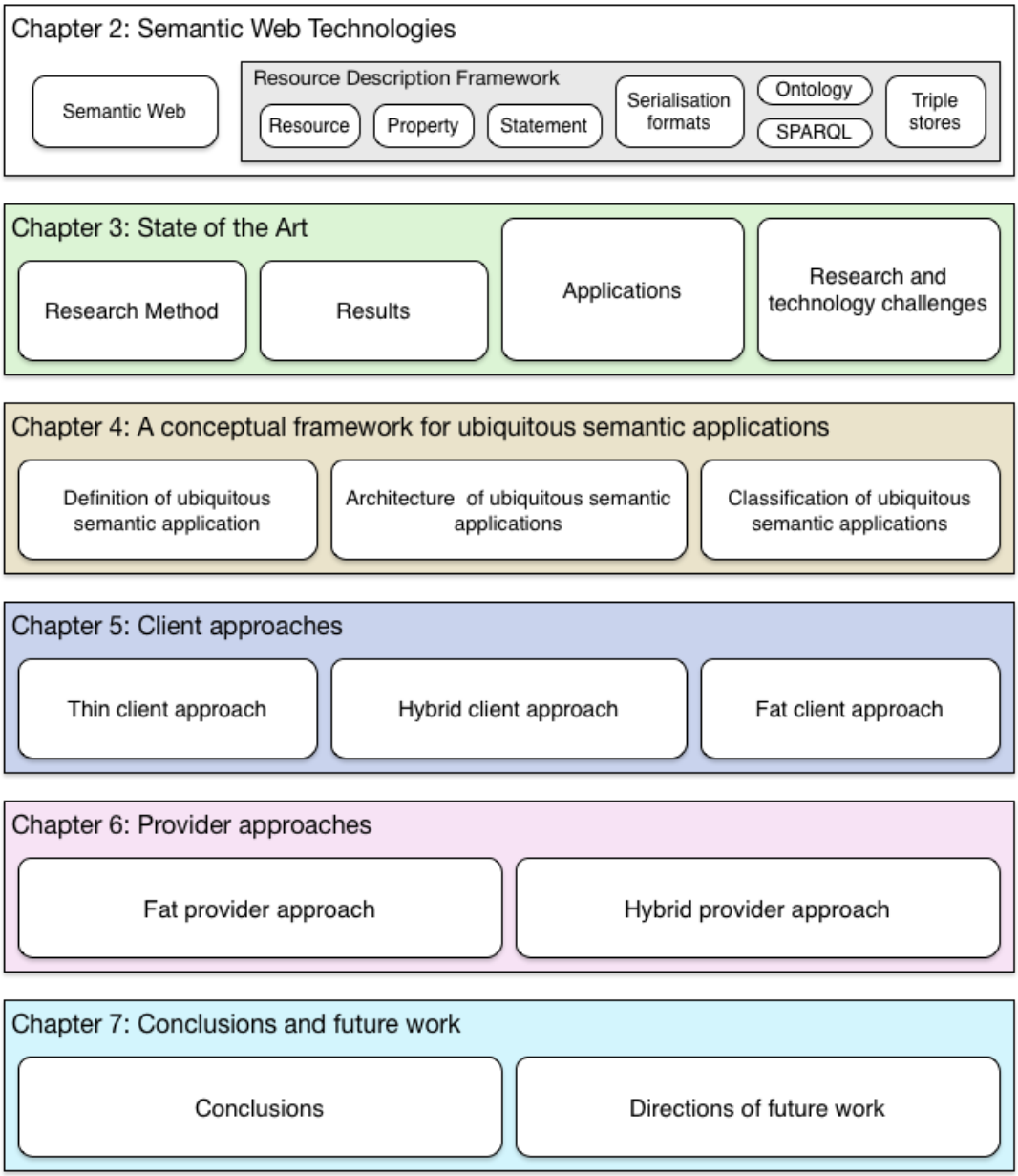


Figure 1.4.: Overview of the thesis structure.

Chapter 4 describes a conceptual framework for ubiquitous semantic applications. First, a ubiquitous semantic application is defined as an entity. Then, the generalized architecture of ubiquitous semantic applications is introduced. Finally, the classification of ubiquitous semantic applications based on various factors is described.

Chapter 5 provides an in-depth overview of existing client approaches for ubiqui-

tous semantic applications. First, thin client approaches is introduced by detailing an example application OntoWiki Mobile with real-world use cases. Then, the hybrid client approach is outlined by describing the Mobile DSSN client. Finally, the fat client approach is discussed by analysing the MSSW Android application.

Chapter 6 provides an in-depth overview of existing provider approaches for ubiquitous semantic applications. First, commonly used fat provider approaches are introduced by presenting Embedded Linked Data Server for ubiquitous devices. Thereafter, hybrid provider approach and possible implementation of such approaches is discussed.

Finally, Chapter 7 concludes with a discussion on each of the contributions of the thesis and proposes future work for each of them.

2. Semantic Web Technologies

This chapter gives a general overview of the Semantic Web. It describes basic concepts, the RDF serialization formats, the ontology and its languages in detail. This chapter is mainly based on [Yu, 2007][1].

The rest of the chapter is organized as follows: In section 2.1, we define Semantic Web. In section 2.2, we describe RDF and why it is important for ubiquitous systems specifically. In subsection 2.2.1, subsection 2.2.2 and subsection 2.2.3 we describe the basic elements of RDF in more detail. In subsection 2.2.4, we introduce the RDF serialization formats. In subsection 2.2.5, we define the term Ontology. In subsection 2.2.6, we describe the ontology languages. In subsection 2.2.7, we describe the SPARQL query language. Finally, in subsection 2.2.8, we explain triplestores.

2.1. The Definition of Semantic Web

There are many different definitions of the Semantic Web. Tim Berners-Lee, the inventor of the World Wide Web, defined it as "not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [Berners-Lee et al., 2001] In other words, Semantic Web allows the machines not only to present data but also to process it.

There is a dedicated team of people at the World Wide Web consortium (W3C) working to improve, extend and standardize the Semantic Web, and many languages, publications, tools have already been developed (e.g. [Tramp et al., 2010b, Heino et al., 2009]). W3C have defined Semantic Web as "the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration, and reuse of data across various applications." [W3C, 2009] In other words, Semantic Web is the machine-readable Web. Semantic Web can be thought of as an efficient way of representing the data on the World Wide Web, or as a globally linked database.

Semantic Web depends on several technologies including Resource Description Framework (RDF) and Uniform Resource Identifiers (URIs). In the following sections we describe each of these technologies in details.

[1] Standard components of the Semantic Web and definitions for them were taken from the book as they are following the defined standards and are widely used. Examples for each component and descriptions of how those components fit into ubiquitous systems are provided by the author.

2.2. Resource Description Framework (RDF)

RDF is an XML-based language for describing information contained in a Web resource. This Web resource can be anything, for example a Web page or a Web site. RDF is the basic building block for supporting the Semantic Web, and is same as HTML is for the conventional Web.

The properties of RDF are:

- RDF is a language recommended by W3C [W3C, 2004] and it is all about metadata,
- RDF is capable of describing any fact (resource) independent of any domain,
- RDF provides a basis for *coding*, *exchanging*, and *reusing* structured metadata,
- RDF is structured; i.e. it is machine-understandable. Machines can do useful operations with the knowledge expressed in RDF,
- RDF allows *interoperability* among applications exchanging *machine understandable* information on the Web.

RDF has several basic elements, namely *Resource*, *Property* and *Statement*, which are discussed in the following subsections.

Utilizing RDF on ubiquitous devices is especially helpful since it allows the devices to handle data for minimizing the cognitive load on the user. This can be especially helpful for situations where user is interacting with the device when the surroundings require full or most of the user attention (e.g. while driving).

2.2.1. Resource

A resource is any thing that is described by an RDF expressions. The resource can be a Web site, a person, an ubiquitous device or anything else. Resource is identified bya **U**niform **R**esource **I**dentifier (**URI**). The rationale of using URIs is that the name of a resource must be globally unique.

In fact, URLs (Uniform Resource Locators), commonly used for accessing Web sites, are simply a subset of URIs. URIs take the same format as URLs, e.g. `http://aksw.org/TimofeyErmilov`. The main reason behind this is that the domain name used in the URL is guaranteed to be unique, therefore the uniqueness of the resource is ensured. In that case the domain name is used as a namespace. Unlike URLs, URIs may or may not refer to an actual Web site or a Web page.

In the ubiquitous world, radio-frequency identification (RFID) tags are frequently used to identify objects. One could say that a RFID tag in a specific network of a ubiquitous devices is a URI.

2.2.2. Property

Property is a resource that has a name and can also be used to describe some specific aspect, characteristic, attribute or relation of the given resource. For instance, `http://xmlns.com/foaf/0.1/name`, denotes the name of some thing. In other words, this property relates a resource representing a thing to its name.

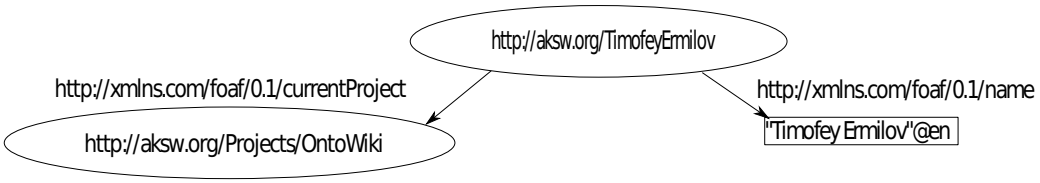


Figure 2.1.: RDF statement represented as a directed graph.

2.2.3. Statement

An RDF Statement is used to describe properties of resources. It is also called a triple and has the following format

<resource (subject)> <property (predicate)> <property value (object)>.

The property value (object) can be a string, literal or another resource referenced by the URI. For example:

<`http://aksw.org/TimofeyErmilov`>
<`http://xmlns.com/foaf/0.1/currentProject`>
<`http://aksw.org/Projects/OntoWiki`>.

This RDF statement simply states that "The subject identified by `http://aksw.org/TimofeyErmilov` has a property identified by `http://xmlns.com/foaf/0.1/currentProject`, whose value is equal to `http://aksw.org/Projects/OntoWiki`". This means that person "Timofey Ermilov" has a "currentProject" which is "OntoWiki".

Another example:

<`http://aksw.org/TimofeyErmilov`>
<`http://xmlns.com/foaf/0.1/name`>
"Timofey Ermilov"@en.

This RDF statement states that "The subject identified by `http://aksw.org/TimofeyErmilov` has the property identified by `http://xmlns.com/foaf/0.1/name`, whose value is equal to "Timofey Ermilov"". This means that person "Timofey Ermilov" has a "name" whose value is "Timofey Ermilov" and the trailing "@en" is the English language tag. In fact, RDF statements can also be expressed as directed graphs, as shown in Figure 2.1.

Subject	Predicate	Object
aksw:TimofeyErmilov	rdf:type	foaf:Person
aksw:TimofeyErmilov	foaf:age	"26"^^xsd:int
aksw:TimofeyErmilov	foaf:skypeID	"yamalight"
aksw:TimofeyErmilov	foaf:birthday	"1987-01-01"^^xsd:date
aksw:TimofeyErmilov	foaf:name	"Timofey Ermilov"@en
aksw:TimofeyErmilov	foaf:currentProject	akswProject:OntoWiki
akswProject:OntoWiki	foaf:homepage	<http://ontowiki.net>

Table 2.1.: Sample RDF statements.

It is worth pointing out here that the subject or the object or both can be an anonymous resource, called a "blank node". Blank nodes are used basically when the key purpose of a specific resource is to provide a context for some other properties to appear. In order to distinguish a blank node from the others, the RDF parser generates an *internal* unique identifier for each blank node. In other words, this identifier given to the blank node helps in identifying the node in a certain RDF document, whereas the URI given to a resource is guaranteed to be globally unique.

Since URIs can be large, there is a short format for writing them i.e. by using a prefix. For instance, if we use `http://aksw.org/` as a prefix and give it a label e.g. `aksw`, then resource `http://aksw.org/TimofeyErmilov` can be written as `aksw:TimofeyErmilov`. Similarly, if `http://xmlns.com/foaf/0.1/` is used as a prefix with label `foaf`, then the properties `http://xmlns.com/foaf/0.1/name` and `http://xmlns.com/foaf/0.1/currentProject`, can be written as `foaf:name` and `foaf:currentProject` in short form. This format is very useful in writing human-readable RDF statements.

Whenever more triples describing a specific resource are added, the machine gets more knowledge about that resource. Table 2.1 shows more RDF statements about Timofey Ermilov. This means that the resource of Timofey Ermilov is the subject of other statements, which give more details about that resource. Note that the object of a particular statement can be in turn the subject of other statement(s), e.g. Timofey Ermilov has a current project identified by URI `akswProject:OntoWiki` and the knowledge base contains more information about that project as well. Also, note that the object of the second and fifth statement (a number and a date) has a trailing datatype. This small knowledge base can also be viewed as a directed graph as shown in Figure 2.2.

Using these simple RDF statements you can pose complex queries to the machine, e.g. "What is the homepage of Timofey Ermilov's current project?".

2.2.4. RDF Serialization Formats

Serializing RDF data is a very crucial issue since different platforms and environments work better with different data formats. It is especially true for ubiquitous systems where a mobile phone could be interacting with a robot or a light switch.

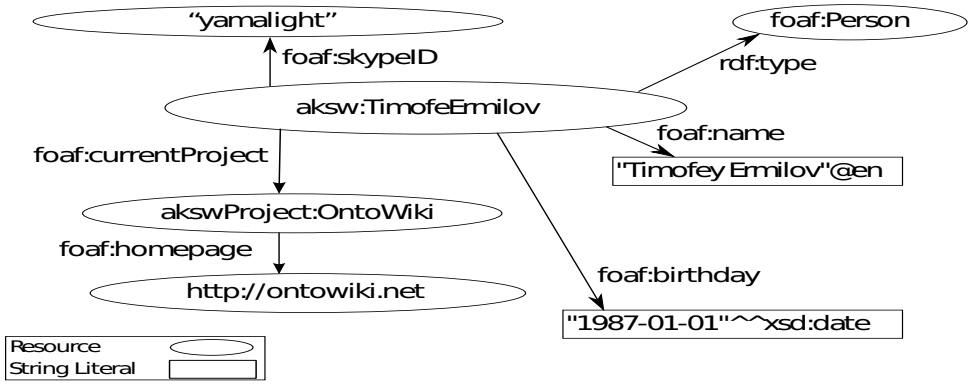


Figure 2.2.: Small knowledge base about Timofey Ermilov represented as a graph.

```
1 <http://aksw.org/TimofeyErmilov> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  > <http://xmlns.com/foaf/0.1/Person> .
2 <http://aksw.org/TimofeyErmilov> <http://xmlns.com/foaf/0.1/age> "26"^^<http://
  www.w3.org/2001/XMLSchema#int> .
3 <http://aksw.org/TimofeyErmilov> <http://xmlns.com/foaf/0.1/skypeID> "yamalight"
  .
4 <http://aksw.org/TimofeyErmilov> <http://xmlns.com/foaf/0.1/birthday>
  "1987-01-01"^^<http://www.w3.org/2001/XMLSchema#date> .
5 <http://aksw.org/TimofeyErmilov> <http://xmlns.com/foaf/0.1/currentProject> <http
  ://aksw.org/Projects/OntoWiki> .
6 <http://aksw.org/TimofeyErmilov> <http://xmlns.com/foaf/0.1/name> "Timofey
  Ermilov"@en .
7 <http://aksw.org/Projects/OntoWiki> <http://xmlns.com/foaf/0.1/homepage> <http://
  ontowiki.net> .
```

Figure 2.3.: Sample N-Triples format.

There are several formats for serializing RDF data:

N-Triples

N-Triples is a simple line-based RDF serialization format. Each RDF triple is written as a separate line and terminated by a period (.). Typically files with N-Triples have the .nt extension [Grant and Beckett, 2004]. Figure 2.3 indicates our sample triples encoded in N-Triples format.

RDF/XML

RDF/XML represents RDF triples in XML format [Beckett, 2004]. The RDF/XML format is more convenient for machines than N-Triples since the traditional XML format is commonly adopted and there are a variety of libraries available that simplify interaction with this format. Figure 2.4 shows our RDF example in RDF/XML format. Files containing RDF/XML data have .rdf as the file extension.

```
<rdf:RDF xmlns:log="http://www.w3.org/2000/10/swap/log#" xmlns:rdf="http://www.w3
.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://aksw.org/Projects/OntoWiki">
    <homepage xmlns="http://xmlns.com/foaf/0.1/" rdf:resource="http://ontowiki.
net"/>
  </rdf:Description>

  <Person xmlns="http://xmlns.com/foaf/0.1/" rdf:about="http://aksw.org/
TimofeyErmilov">
    <currentProject rdf:resource="http://aksw.org/Projects/OntoWiki"/>
    <birthday rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1987-01-01</
deathDate>
    <age rdf:datatype="http://www.w3.org/2001/XMLSchema#int">26</age>
    <skypeID xmlns="http://dbpedia.org/property/" xml:lang="en">yamalight</
skypeID>
    <name xmlns="http://dbpedia.org/property/" xml:lang="en">Timofey Ermilov</
name>
  </Person>
</rdf:RDF>
```

Figure 2.4.: Sample RDF/XML format.

```
@prefix aksw: <http://aksw.org/> .
@prefix akswProject: <http://aksw.org/Projects/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

aksw:TimofeyErmilov a foaf:Person;
    foaf:age "26"^^xsd:int;
    foaf:currentProject akswProject:OntoWiki;
    foaf:birthday "1987-01-01"^^xsd:date;
    foaf:skypeID "yamalight";
    foaf:name "Timofey Ermilov"@en .

akswProject:OntoWiki foaf:homepage <http://ontowiki.net> .
```

Figure 2.5.: Sample N3 format.

N3

N3 stands for Notation3 and is a shorthand notation for representing RDF graphs. N3 was designed to be easily read by humans and it is not an XML-compliant language [Berners-Lee and Connolly, 2011]. Figure 2.5 shows our RDF example in N3 format. Files containing RDF data in N3 format normally have a .n3 extension.

Turtle

Turtle is a subset of N3. Turtle stands for Terse RDF Triple Language. Turtle files have a .ttl extension [Dave and Berners-Lee, 2011]. This particular serialization is popular among developers of the Semantic Web.

2.2.5. Ontology

W3C defines an ontology as "the terms used to describe and represent an area of knowledge." [Heflin, 2004].

This definition has several aspects that should be discussed. First, the definition states that an ontology is used to describe and represent an area of knowledge. In other words, an ontology is domain specific; it does not represents all knowledge areas, but one specific area of knowledge. A domain is simply a specific subject area or sphere of knowledge, such as literature, medicine, education, etc.

Second, the ontology contains terms and relationships among those terms. Terms are also called classes, or concepts; these words are interchangeable. The relationships between these classes can be expressed by using a hierarchy, i.e. superclasses represent higher-level concepts and subclasses represent finer concepts. The finer concepts have all the attributes and features that the higher concepts have.

Third, in addition to the aforementioned relationships among classes, there is another level of relationship expressed by using a special group of terms called properties. These property terms describe various features and attributes of the concepts and they can also be used to associate different classes together. Thus, the relationships among classes are not only superclass or subclass relationships, but relationships expressed in terms of properties as well.

In other words, an ontology defines a set of classes (e.g. "Person", "Book", "Writer"), and their hierarchy, i.e. which class is a subclass of another one (e.g. "Writer" is a subclass of "Person"). The ontology also defines how these classes interact with each other, i.e. how different classes are connected to each other via properties (e.g. a "Book" has an author of type "Writer").

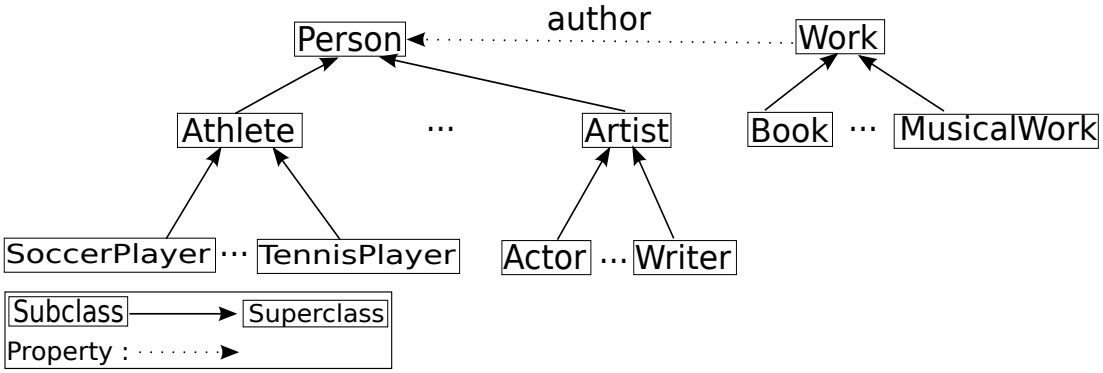


Figure 2.6.: Excerpt of the DBpedia ontology.

Figure 2.6 shows an excerpt of the ontology representing DBpedia[2]. This ontology shows that there is a class called "Writer" which is a subclass of the class "Artist",

[2]http://dbpedia.org/

which in turn a subclass of "Person". *William Shakespeare, Johann Wolfgang von Goethe*, and *Dan Brown* are candidate instances of the class "Writer". The same applies to the class "Work" and its subclasses. Note that there is a property called "author" relating an instance of class "Work" to an instance of the class "Person" i.e. it relates a work to its author. For instance, the book titled "First Folio" is an instance of classes "Work" and "Book", and related via property "author" to its author "William Shakespeare", which is an instance of the classes "Person", "Artist" and "Writer".

So, why do need ontologies? The main benefits of an ontology are:

- it provides a common and shared understanding/definition about certain key concepts in the domain,
- it provides a way for reuse of domain knowledge,
- it makes the domain assumptions explicit,
- it provides a way to encode knowledge and semantics such that machines can understand it.

2.2.6. Ontology Languages

The question now is "What are the languages used to create ontologies?". There are several languages which can be used to encode ontologies such as RDF Schema (RDFS) and Web Ontology Language (OWL).

RDFS

RDFS is an ontology language, which can be used to create a vocabulary for describing classes, subclasses and properties of RDF resources and it is a W3C recommendation [Brickley and Guha, 2004]. The RDFS language also associates the properties with the classes it defines. RDFS can add semantics to RDF predicates and resources, i.e. it defines the meaning of a given term by specifying its properties and what kinds of objects these properties can have. It is worth noting here that RDFS is written in RDF, so any RDFS document is a legal RDF document.

OWL

The Web Ontology Language (OWL) is used to create ontologies and is also a W3C recommendation [Bechhofer et al., 2004]. It is built on RDFS. We can say that **OWL = RDFS + new constructs for expressiveness**. All classes and properties provided by RDFS can be used in OWL ontologies. OWL and RDFS have the same purpose which is defining classes, properties and relations among these classes. OWL has an advantage over RDFS which is its capability to express more complex relationships.

```
1  <http://dbpedia.org/ontology/Person> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
   type> <http://www.w3.org/2002/07/owl#Class> .
2  <http://dbpedia.org/ontology/Artist> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
   type> <http://www.w3.org/2002/07/owl#Class> .
3  <http://dbpedia.org/ontology/Artist> <http://www.w3.org/2000/01/rdf-schema#
   subClassOf> <http://dbpedia.org/ontology/Person> .
4  <http://dbpedia.org/ontology/Writer> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
   type> <http://www.w3.org/2002/07/owl#Class> .
5  <http://dbpedia.org/ontology/Writer> <http://www.w3.org/2000/01/rdf-schema#
   subClassOf> <http://dbpedia.org/ontology/Artist> .
6  <http://dbpedia.org/ontology/Work> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
   type> <http://www.w3.org/2002/07/owl#Class> .
7  <http://dbpedia.org/ontology/Book> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
   type> <http://www.w3.org/2002/07/owl#Class> .
8  <http://dbpedia.org/ontology/Book> <http://www.w3.org/2000/01/rdf-schema#
   subClassOf> <http://dbpedia.org/ontology/Work> .
9  <http://dbpedia.org/ontology/author> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
   type> <http://www.w3.org/2002/07/owl#ObjectProperty> .
10 <http://dbpedia.org/ontology/author> <http://www.w3.org/2000/01/rdf-schema#domain
   > <http://dbpedia.org/ontology/Work> .
11 <http://dbpedia.org/ontology/author> <http://www.w3.org/2000/01/rdf-schema#range>
    <http://dbpedia.org/ontology/Person> .
```

Figure 2.7.: OWL representation of a part our ontology in N-Triples format.

Due to its expressiveness power, most ontology developers use OWL to develop their ontologies. For example, an ontology developer can create a new class as the union or intersection of two or more classes using the expressive power of OWL. With OWL one can also declare that two classes are representing the same thing. For instance, consider the case that there are two separate ontologies created by different developers. In the first ontology there is a class called “Poet” and in the other ontology there is a class called “PoetryWriter”. In fact, these classes are equivalent to each other and in RDFS one cannot declare that these classes are equivalent, but with OWL one can.

OWL provides some powerful features for properties as well. For example, in OWL one can declare that two properties are the inverse of each other, (e.g. author, and isAuthorOf). Figure 2.7 indicates a part of our ontology expressed in OWL.

Note that for property author we have defined two properties domain, and range. The domain property defines the class of instances which can be the subject of that property (author property), while the range property defines the class of instances which can be the object of that property.

OWL has many powerful features, interested reader can find more about those feature in [Bechhofer et al., 2004].

2.2.7. SPARQL Query Language

“The SPARQL Protocol and RDF Query Language (SPARQL) is a query language and protocol for RDF.” [Clark et al., 2008]. SPARQL is a W3C standard and it is used to ask queries against RDF graphs. SPARQL allows the user to write queries that consist of triple patterns, conjunctions (logical “and”), disjunctions

```
1    PREFIX aksw: <http://aksw.org/>
2    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3    SELECT ?homepage
4      WHERE {aksw:TimofeyErmilov foaf:currentProject ?project.
5      ?project foaf:homepage ?homepage. }
```

Figure 2.8.: SPARQL query to get the homepage of Timofey Ermilov's current project.

(logical "or") and/or a set of optional patterns [Wikipedia, 2013]. Examples of these optional patterns are: `FILTER`, `REGEX` and `LANG`.

The SPARQL query specifies the pattern(s) that the resulting data should satisfy. The results of SPARQL queries can be result sets or RDF graphs. SPARQL has four query forms, specifically `SELECT`, `CONSTRUCT`, `ASK` and `DESCRIBE` [Prud'hommeaux and Seaborne, 2008].

Let us take an example to clarify the usage of SPARQL. Assume that we want to ask the query "What is the homepage of Timofey Ermilov's current project?" to our small knowledge base. Figure 2.8 shows a SPARQL query to get information about the homepage of Timofey Ermilov's current project.

In Figure 2.8, lines 1 and 2 define prefixes in order to write URIs in their short forms. Line 3 declares the variables that should be rendered to the output of that query, which is only one variable `?homepage`. Note that SPARQL variables start either with a question mark "`?`", or with a dollar sign "`$`". Line 4 states that for the statement with subject `aksw:TimofeyErmilov` and property `foaf:currentProject`, we want the value of its object to be assigned to a variable called `?project`. Upon execution, this variable will take the value of `akswProject:OntoWiki`. In line 5, we want variable `?project` which now has the value `akswProject:OntoWiki`, to be the subject of the next statement. In other words, the statement will be `akswProject:OntoWiki foaf:homepage ?homepage`. Now, variable `?homepage` is the only unknown variable of the statement, and it will take the value `http://ontowiki.net`. Eventually, its value will be rendered to the output.

2.2.8. Triplestore

The crucial question here is "How do we store RDF data for efficient and quick access?". Basically, RDF data is stored in triplestores. A triplestore is a software program capable of storing and indexing RDF data efficiently, in order to enable querying this data easily and effectively. A triplestore for RDF data is like Relational Database Management System (DBMS) for relational databases.

Most triplestores support SPARQL query language for querying RDF data. As

there are several DBMSs in the wild, such as Oracle[3], MySQL[4] and SQL Server[5], similarly there are several triplestores. Virtuoso [Erling and Mikhailov, 2007], Sesame [Broekstra et al., 2002] and BigOWLIM [Bishop et al., 2011] are typical examples of triplestores for desktop and server computers. DBpedia, for example, uses Virtuoso as the underlying triplestore. Since ubiquitous devices usually have less powerful CPU and smaller memory size, there are special version of triplestores that are built to be used on such low-power devices. Androjena[6], RDF On The Go[7], μJena[8] and OpenSesame[9] are examples of such triplestores.

[3] http://www.oracle.com/us/products/database/overview/index.html
[4] http://www.mysql.com
[5] http://www.microsoft.com/en-us/sqlserver/default.aspx
[6] http://code.google.com/p/androjena/
[7] http://code.google.com/p/rdfonthego/
[8] http://poseidon.ws.dei.polimi.it/ca/?page_id=59
[9] http://bluebill.tidalwave.it/mobile/

3. State of the art

This chapter covers the State of the Art for the area of ubiquitous semantic applications. It provides a comprehensive overview of the existing approaches, systems and applications as well as integrates them into a common conceptual scheme. The main goal of this chapter is analyzing existing semantic applications, approaches and systems for ubiquitous devices and providing a set of quality attributes, which can serve as guidelines for designing suitable and effective semantic applications for ubiquitous devices as well as provide a simple but comprehensive model for the categorization of existing ones. This chapter is based on [Ermilov et al., 2014].

The rest of the chapter is organized as follows. In section 3.1, we introduce the reader to ubiquitous semantic applications. In section 3.2, we describe the research methodology and the review protocol used for conducting the systematic review. In section 3.3, we first define the basic terminologies and then we elaborate on the results of the review by surveying the extracted quality attributes. In section 3.4, we discuss three existing UbiSA and describe them in the light of the quality attributes. In section 3.5, we report on the gaps and open research issues revealed from the results of our systematic literature review. Finally, in section 3.6 we conclude and present some ideas for future work.

3.1. Introduction

Recently practical approaches for the development of UbiSA that allow access to the Web of Data have made quite some progress. On the backend side, a variety of triple stores were developed and their performance and maturity improved steadily. With increasing power of ubiquitous devices it has become possible to use some of the triple stores on devices to allow offline access to the semantic data. Similarly tools and algorithms for processing and presenting data on ubiquitous devices are progressing and approaches are deployed for the use on the emerging Web of Data. The quantity and quantity of semantic content being made available on the Data Web is rapidly increasing, mainly due to the use of automated knowledge extraction techniques or due to the semantic enrichment and transformation of existing structured data. Despite many interesting showcases (e.g. *Sindice*[1], *Parallax*[2] or *PowerAqua*[3]), we still lack more user friendly and scalable approaches for the exploration, browsing and search of semantic data. However, the currently

[1] `http://sindice.com/`
[2] `http://www.freebase.com/labs/parallax/`
[3] `http://technologies.kmi.open.ac.uk/poweraqua/`

least developed aspect of access to the semantic data is, from our point of view, the user-friendly *ubiquitous* applications that provide access to rich semantic content.

To define UbiSA, we must first specify what we mean by *ubiquitous* applications and *semantic* documents.

A guiding principle of ubiquitous applications is to break away from desktop computing to provide computational services to a user when and where required [Salber et al., 1998]. Ubiquitous applications are characterized by two main attributes [Weiser, 1991]:

- *ubiquity*: interaction with the system is available wherever the user needs it;
- *transparency*: the system is non-intrusive and is integrated into the everyday environment.

Semantic documents are documents that consist of semantic data and describe specific entities or collections of entities. Semantic data on the other hand is the data that is defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications. Semantic data should provide a basis for coding, exchanging and reusing structured metadata among applications exchanging *machine understandable* information on the Web.

Taking all of the above into account, we define *ubiquitous semantic application* as the computer software implemented specifically for ubiquitous devices and designed to help the user to perform specific tasks that satisfy the following requirements:

- the application is designed and developed specifically for (or with respect to) ubiquitous devices,
- the application utilizes semantic data during the work process in any way (e.g. executing SPARQL queries, reading or writing RDF triples).

A ubiquitous semantic application provides a human accessible interface with capabilities for reading, writing or modifying semantic documents. Semantic documents facilitate a number of important aspects of information management:

- For *search and retrieval*, enriching documents with semantic representations helps to create more efficient and effective search interfaces, such as faceted search (e.g. in [Ermilov et al., 2011a]) or question answering. Ultimately, users are empowered to fight the increasing information overload and gain better access to relevant documents and answers related to their information needs.
- For *information presentation*, semantically enriched documents can be used to create more sophisticated ways of flexibly visualizing information, such as geospatial maps as described in [Viana et al., 2007, Braun et al., 2010, Wilson et al., 2005b].

- For *information integration*, semantically enriched documents can be used to provide unified views on heterogeneous data stored in different applications by creating composite applications such as semantic mashups, like ones presented in [Wilson et al., 2005b, Ermilov et al., 2011a].
- To realize *personalization*, semantic documents provide customized and context-specific information which better fits user needs and will result in delivering customized applications such as personalized semantic portals (e.g. [Ruta et al., 2010a, WeiBenberg et al., 2006]).
- For *reusability* and *interoperability*, enriching documents with semantic representations (e.g. using the SKOS[4] and Dublin Core[5] vocabularies) facilitates exchanging content between disparate systems.

There are already many approaches, frameworks and tools available for ubiquitous devices which address different aspects of this tasks. Due to the wealth of different approaches emerging, it is crucial to obtain an overview on the advancement in this emerging field. Furthermore, having a holistic view on approaches and tools provides us with an exhaustive set of quality attributes, which are important for conceiving guidelines for developing more effective and intuitive UbiSA. Since most of the current approaches and applications are developed for smartphones (e.g. [Tramp et al., 2011a, Viana et al., 2007, Wilson et al., 2005b, Bellini et al., 2012, Ostuni et al., 2013]), these quality attributes will be especially important for new and upcoming ubiquitous devices such as application-enabled smart TVs, gaming consoles, wearable computers, etc. UbiSA development for tablet computers that are also gaining popularity and predicted to outnumber PCs in the next few years can also benefit from defined quality attributes.

In this chapter, we summarize the findings of a systematic literature review on UbiSA. We extract different types and properties of applications proposed for ubiquitous use. The results reveal a set of quality attributes which can be used for classification of UbiSA. Furthermore, we report on the suggested application types and features proposed in the literature to realize these quality attributes.

3.2. Research Method

We followed a formal systematic literature review process for this study based on the guidelines proposed in [Dyba et al., 2007, Kitchenham, 2004]. A systematic literature review is an evidence-based approach to thoroughly search studies relevant to some pre-defined research questions and critically select, appraise and synthesize findings for answering the research questions at hand. Systematic reviews maximize the chance to retrieve complete data sets and minimize the chance of bias. As

[4] `http://www.w3.org/2004/02/skos/`
[5] `http://www.cs.umd.edu/projects/plus/SHOE/onts/dublin.html`

part of the review process, we developed a protocol (described in the sequel) that provides a plan for the review in terms of the method to be followed, including the research questions and the data to be extracted.

3.2.1. Research Questions

The goal of our survey is analyzing existing semantic applications for ubiquitous devices and thereby providing a set of quality attributes, which can serve as guidelines for designing suitable and effective semantic applications for ubiquitous devices. To achieve this goal we aim to answer the following general research question:

What are the existing approaches for development of ubiquitous semantic applications?

We can divide this general research question into the following more concrete sub-questions:

- RQ1. *How to classify existing approaches for development of ubiquitous semantic applications?*
- RQ2. *What type of applications are developed in each approach?*
- RQ3. *What are the features supported by the proposed application?*
- RQ4. *How is the application evaluated?*

After doing some pilot searches and consulting experts in the field, we obtained a list of pilot studies which served as a basis for the systematic review.

3.2.2. Search Strategy

To cover all the relevant publications, we used the following electronic libraries:

- ACM Digital Library
- IEEE Xplore Digital Library
- ScienceDirect
- SpringerLink
- ISI Web of Sciences

Based on the research questions and pilot studies, we found the following basic terms to be most appropriate for the systematic review:

1. *ubiquitous* OR *mobile*

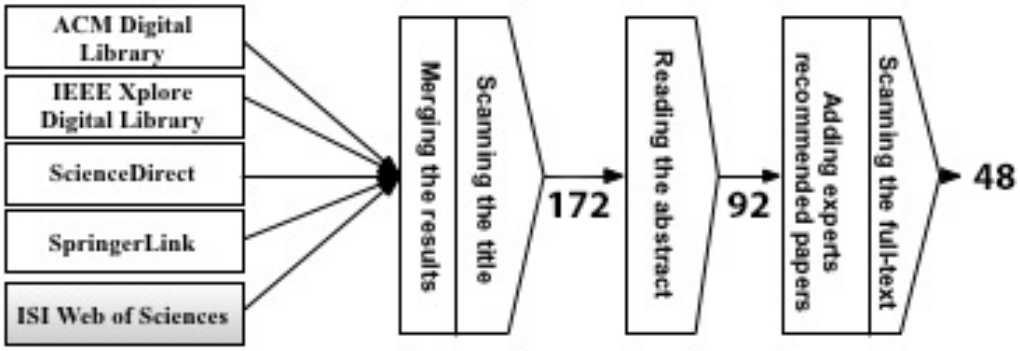


Figure 3.1.: Steps followed to scope the search results.

2. *semantic* OR *linked data* OR *web of data* OR *data web*
3. *application* OR *software* OR *system*

To construct the search string, all these search terms were combined using Boolean "AND" as follows:

1 AND 2 AND 3

The next decision was to find the suitable field (i.e. title, abstract and full-text) to apply the search string on. In our experience, searching in the "title" alone does not always provide us with all relevant publications. Thus, "abstract" or "full-text" of publications should potentially be included. On the other hand, since the search on the full-text of studies results in many irrelevant publications, we chose to apply the search query additionally on the "abstract" of the studies. This means a study is selected as a candidate study if its title or abstract contains the keywords defined in the search string. In addition, we limited our search to the publications that are written in English and are published after 2002 (when the first ISWC conference was held).

3.2.3. Study Selection

Some of the studies might contain the keywords used in the search string but might still be irrelevant for our research questions. Therefore, a study selection has to be performed to include only studies that contain useful information for answering the research question.

Peer-reviewed articles that satisfy all the following inclusion criteria are selected as primary studies:

- I1. A study that focuses on ubiquitous semantic applications

- I2. A study that either proposes an approach or a set of specific features for the purpose of accessing semantic content on ubiquitous devices.

Studies that met any of the following criteria were excluded from the review:

- E1. A study that does not focus on ubiquitous semantic applications but only mentions the term e.g. as an example or use case.
- E2. A study that does not propose any approach or specific features used in development of UbiSA.
- E3. A study that is not about semantic data (e.g. studies about semantics as the study of meaning).

The conduction of our search commenced in early September 2013. As a consequence, our review included studies that were published and/or indexed before that date. As shown in Figure 3.1, we first applied the search query on each data source separately. Subsequently, to remove duplicate studies, we merged the results obtained from the different data sources. To remove irrelevant studies, we scanned the articles by title and thereby reduced the number of studies to 172. Then, we read the abstract of each publication carefully and further decreased the number of studies to 92. Finally, we added a list of additional papers recommended by experts and then scanned the full-text of the publications. Experts recommended adding 3 papers that did not appear in the results during the search phase. We checked the full-text of studies to see if they fit with our predefined selection criteria. The result comprised 48 publications that represented our final set of *primary studies*.

3.2.4. Data Extraction and Analysis

The bibliographic metadata about each primary study were recorded using the bibliography management platform *JabRef*[6]. In addition, we extracted the following information from each paper:

- used approach for UbiSA development
- type of application
- features supported by the application
- domain and type of user

[6]http://jabref.sourceforge.net/

- evaluation method used in the paper

To analyze the information appropriately, we required a suitable qualitative data analysis method applicable to our dataset. We used *coding* as our qualitative analysis method. A common method that is used for this purpose is the *grounded theory method* because the theories (the UbiSA approaches and application features) are "grounded" in the data [Glaser and Strauss, 1967].

Constant comparison method, one of the grounded theory techniques, has been often used in analyzing data and generating categories of data. Although constant comparison method can be used on any set of data, it is particularly suitable for the data that are context sensitive [Seaman, 1999] (i.e. data can be interpreted differently in different contexts). To interpret UbiSA approaches and application features correctly, one often needs to understand in which context the approach and feature is proposed and how it is addressed. For instance, consider one study that mentions "interoperability" as a feature for application. Without understanding the context of this feature, we cannot conclude whether this feature is about designing interoperable UIs or about supporting annotation/ontology interoperability.

Miles and Huberman [Miles and M., 1994] described *coding* as a procedure for the constant comparison method. Codes are tags or labels for assigning units of meaning to the descriptive or inferential information compiled during a study. Codes are efficient data-labelling and data-retrieval devices [Miles and M., 1994]. One method of creating codes (recommended in [Miles and M., 1994]) is that of creating a provisional "start-list" of codes prior to fieldwork. We created this list from our research questions and the pilot studies. To carry out the analysis systematically, we used the following coding procedures proposed by Lincoln and Guba [Miles and M., 1994]:

- *Filling-in*: we read each study carefully and added the codes for related fragments and items. As new insights or new ways of looking at the data emerged, we reconstructed our coherent coding schema.
- *Extension*: if needed, we returned to materials coded earlier and interrogated them in a new way, with a new theme, construct, or relationship.
- *Bridging*: if new or previously not understood relationships within units of a given category were found, we recorded that relationship.
- *Surfacing*: we identified new categories which contained the previously created codes.

We used the *Weft QDA* software[7] to record the codes. More detailed information on coding and usage of the *Weft QDA* software can be found in [Fenton, 2006].

[7] `http://www.pressure.to/qda/`

3.2.5. Overview of Included Studies

For quantitative analysis purposes, we performed some queries on the collected database of primary studies. The distribution of studies per year as shown in Figure 3.2 indicates an increasing intensity of research in the area of ubiquitous semantic applications. The remarkable rise after 2009 can be explained with the emergence and increasing adoption of new touch-enabled smartphones lead by Apple's iPhone. The low number of publications in 2013 can be attributed by commencing our survey in September, when most conference proceedings were not yet published or properly indexed.

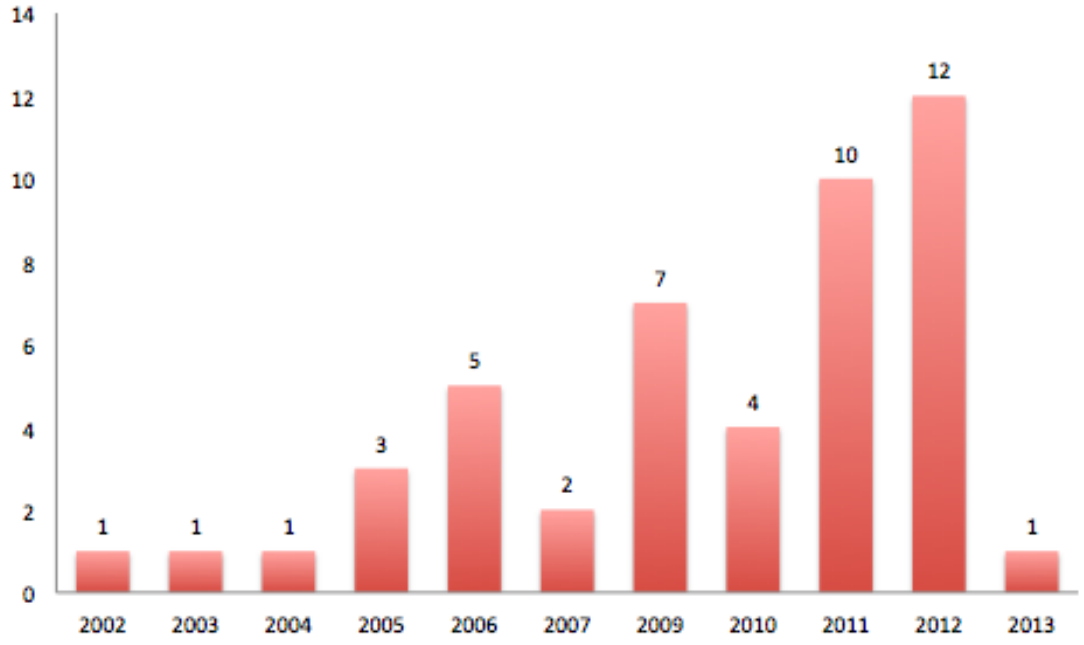


Figure 3.2.: Publications per year.

The primary studies included 29 conference papers, 11 journal articles, 4 workshop papers and 2 book sections. Among them, the following studies are survey papers [Sakkopoulos, 2009, Bellavista et al., 2012] which provide an overview of the approaches that unify semantic Web technologies with a number of different mobile operations.

3.3. Results

In this section we first define the basic concepts used in the chapter and then elaborate on the results of our qualitative data analysis.

3.3.1. Terminology

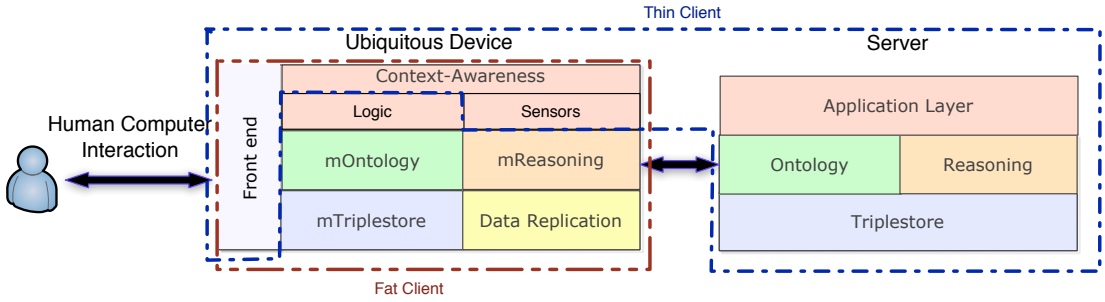


Figure 3.3.: Ubiquitous semantic applications architecture.

A generalized architecture of ubiquitous semantic applications is depicted in Figure 3.3. The presented generalized architecture was conceived on the basis of reviewed approaches, systems and applications to reflect all possible variations of UbiSA. In general, any existing UbiSA can be described using this generalized architecture. The opposite is true as well – it is possible to create any UbiSA by instantiating the presented generalized architecture. In the sequel we describe the research context terminology as well as individual components and concepts in more detail.

Human Computer Interaction (HCI) is crucial for the development of ubiquitous semantic applications and represents a research field that aims at improving the interactions between users and computers by making computers more usable and receptive to the user's needs. This field is particularly relevant for ubiquitous semantic applications, since ubiquitous devices such as smartphones, tablet PCs and smart TVs offer a wide range of novel interaction paradigms.

Frontend is an abstraction, which simplifies the usage of underlying components by providing a user-friendly interface for HCI. This area is especially important for UbiSA development since ubiquitous devices might have several different ways to interact with the user (e.g. touch screen input, voice control, gestures control, camera input, etc). There are currently several standard approaches used to develop front-ends such as:

- *Native.* Natively developed frontends provide users with familiar experience since usually all of the control elements are generated by operating system thus resulting in a consistent look-and-feel. Native frontend development is performed using tools provided by the platform manufacturer and is thus limited to a specific platform. Hence, portability and consequently development efficiency are a main disadvantage of native frontend development [Charland and Leroux, 2011].

- *Web.* Web frontends are gaining popularity with the increase of data connection speeds and improvements of web browsers for ubiquitous devices. Web frontends are built using web technologies such as HTML, JavaScript and CSS. This results in exceptional portability. However, there are still many nuances (e.g. different screen sizes, connectivity issues, hardware limitations) that need to be considered while developing a web frontend (cf. [Roto, 2006] for an in-depth discussion of such issues). One of their main issues is the look-and-feel of the frontend, since it is not possible to provide a native user experience for all platforms from a single web frontend.

- *Abstract.* Abstract frontend development approaches, that are inspired by the model-driven development paradigm gained recently popularity. These approaches define an abstract frontend description and to transform the abstract model into a (usually limited) number of platform-specific frontends which provide a native look-and-feel for each supported platform. The Unified Interface Markup Language (UIML) as described in [Farooq Ali et al., 2005] is an example. While providing native UI support for several platforms, abstract development approaches are usually very limited with regard to the customization of the frontend elements.

Context-Awareness is a term coined to describe applications that can passively or actively determine their context by utilizing on-board or peripheral device sensors. Especially in mobile and ubiquitous usage scenarios context-awareness is a crucial aspect, since the device is used in very specific and distinct situations (e.g. while walking or sitting in a restaurant). Also, mobile and ubiquitous devices have a variety of sensors not available in desktop computers, such as GPS, accelerometer, gyroscope, compass, etc. (a detailed overview is provided in [Lane et al., 2010]). Hence, for applications supporting ubiquitous usage scenarios, it is of paramount importance to employ the additional sensing techniques for optimally supporting the users' experience. Context-awareness includes two dependent parts: sensors and logic. Logic is processing data from sensors and facilitates decision making about the current user's context. [Hu et al., 2009, Van Woensel et al., 2011a, Costabello et al., 2012, Yu et al., 2012] describe various approaches in that regard. A survey on context data distribution for mobile ubiquitous devices is also described in [Bellavista et al., 2012]. While sensors can only be located on the ubiquitous device, the logic layer can as well be moved to the server application layer (e.g. in thin client approach section 3.3.3).

Ontology is a formal, explicit specification of a shared conceptualisation that represents knowledge as a set of concepts within a domain as well as relationships between those concepts. Ontologies can be used in various scenarios in the context of UbiSA:

- *Ontology authoring* the UbiSA is used for the creation of ontologies on the

ubiquitous device. For example, OntoWiki mobile (as described in subsection 3.4.1) is a comprehensive ontology and knowledge base authoring interface.

- *Ontology use* in UbiSA can happen at the server (e.g. [Villalonga et al., 2009]) or directly at the ubiquitous device (labeled as mOntology in Figure 3.3). Examples for the use of ontologies directly at the device are presented in [Korpipää and Mäntyjärvi, 2003], [Liao et al., 2005], [Cano et al., 2012], [Zargayouna and Amara-Hachmi, 2006] and [Hu and Moore, 2007].

Reasoning is the act or process of deriving logical conclusions from premises known or assumed to be true. Depending on the UbiSA development approach, reasoning can be performed on the ubiquitous device itself (mReasoning on Figure 3.3, e.g. [Chen et al., 2010, Steller et al., 2009, Ruta et al., 2010b, Ruta et al., 2012, Motik et al., 2012]) or on the server (e.g. [Dietze et al., 2009]). Reasoning approaches being deployed directly on the device have to cope with resource restrictions (e.g. available memory capacity and processing power). Hence, most mobile reasoning engines currently provide only simple rule processing through forward/backward chaining [Ruta et al., 2010b]. There is also an increasing number of studies that aim to develop scalable semantic reasoning techniques that are useful for both ubiquitous and standard service selection algorithms [Steller et al., 2009].

Data Replication is the process of exchanging information so as to ensure consistency between different UbiSAs or a UbiSA's client and server application layers. Replication is crucial for any UbiSA since the data connection might be limited, unstable or not available at all. Maintaining data consistency while replicating data to the client and synchronizing changes to the server is especially important in the area of Social Semantic Web. This is because social user interactions usually involves much collaboration aiming at creating and changing data. Examples of approaches include:

- Frameworks for selective replication of data sets on mobile devices. The goal of such frameworks is to provide access to data sets in situations without network connectivity when communication with remote data sources is impossible. Most frequently used technique is adding intermediate components that handle queries transparently, either by forwarding them to the actual data store if connectivity is up, or by answering them from a locally cached partial replica of the data set on the mobile device, if there is no connectivity [Schandl and Zander, 2009].

- Conflict resolution approach based on a combination of distributed revision control strategies as well as the data evolution and ontology refactoring [Ermilov et al., 2011a].

- Domain-specific approaches that tries to combine two or more techniques. For example, combination of multi-resolution spatial data structure and semantic caching, aimed towards efficient spatial query processing in mobile environments as described in [Sun et al., 2005].

Triplestore is a specific database for the storage and retrieval of information adhering to the RDF data model, i.e. triples composed of subject-predicate-object (e.g. "Bob is 35" or "Bob knows Fred"). Triplestores can also be implemented either on the ubiquitous device itself (mTriplestore on Figure 3.3, e.g. [Tramp et al., 2011a]) or on the server.
[Braun et al., 2010, Wilson et al., 2005b] and [Van Woensel et al., 2011b] describe UbiSAs accessing triples stores on the server side. With the emerging of Android OS several triplestores (mostly developed in Java) were ported to the Android system. List of the most popular triplestores for ubiquitous devices is provided in Table 3.1.

Triplestore	Description
Androjena http://code.google.com/p/androjena/	Androjena is a porting of Hewlett-Packard's Jena semantic web framework to the Google Android platform.
RDF On The Go http://code.google.com/p/rdfonthego/	RDF On The Go is the project to build a persistent RDF store and query processor on Android phone.
TriplePlace https://github.com/white-gecko/TriplePlace	TriplePlace a light weight and flexible Triple Store for Android. It uses a indexing structure similar to the one in Hexastore. TriplePlace uses TokyoCabinet as persistent storage system.
OpenSesame http://bluebill.tidalwave.it/mobile/	OpenSesame Core was ported to Android as part of blueBill Mobile project.
μJena http://poseidon.ws.dei.polimi.it/ca/?page_id=59	μJena is a reduced, lightweight porting of the Jena API for Android.

Table 3.1.: List of triplestores for ubiquitous platforms.

Server Application Layer (SAL) is an abstraction, describing the underlying logic of the UbiSA server. SAL can be used in both thin and fat client approaches (e.g. [Soriano et al., 2006, Gümüs et al., 2006]). However, SAL is more commonly used in thin client approaches to execute most of the required operations on

the server. A fat client approach might as well use SAL to outsource expensive operations (e.g. reasoning) in order to decrease the usage of ubiquitous device resources.

Social Semantic Web is a very general technology field triggered by the advent of Web 2.0. It aims at bringing a social novelty, rather than a technical one by providing user-friendly tools to facilitate broad user participation in the process of creating semantic content. Examples can be found in [Viana et al., 2007, Ermilov et al., 2011a]. The Social Semantic Web vision comprises many of the aforementioned domains and techniques. The Social Semantic Web is a crucial application domain for UbiSA.

3.3.2. Possible User Roles

One more important aspect that needs to be considered in process of development of the UbiSA is possible user roles. User roles describe main focus in user interaction with the UbiSA.

Professional user is a user that has extensive knowledge of how ubiquitous devices and semantic web applications work. Professional users usually have deep understanding of the UbiSA they use. Main purpose of using UbiSA is usually data gathering for research purposes (e.g. for the knowledge management project *Caucasian Spiders* [Ermilov et al., 2011a]).

User seeking entertainment or social interactions is a user that has little or no knowledge on semantic web applications and has only basic skills in using ubiquitous devices. Gamification, research field that is currently gaining popularity, can be used to engage such users into useful process that is also entertaining for them.

User aiming to learn or access certain information is a user that has little or no knowledge on semantic web applications and has basic or advanced skills in using ubiquitous devices. Usual task of such user is to find information related to his current environment (e.g. points of interest [Braun et al., 2010], semantic calendar [Sheshagir et al., 2004]).

Engaged user is a user that has advanced knowledge in semantic web applications and advanced skills in using ubiquitous devices. Usual task of such user is to contribute to crowdsourcing projects (e.g. create new points of interest [Braun et al., 2010]).

3.3.3. Ubiquitous Semantic Applications Development Approaches

There are already a number of different approaches proposed for ubiquitous applications development but for non-semantic content (see [Lee et al., 2004] and [Niemelä and Latvakoski, 2004]) and a smaller number of approaches specifically for semantic content (e.g. [Veijalainen et al., 2006]). These approaches aim at user gratification in the form of useful visualizations and interesting data aggregation but do not focus on using shared vocabularies and formal ontologies which ultimately facilitate customizability, portability and reuse. With regard to ubiquitous applications development recent approaches can be roughly classified into two categories: *Fat Client* and *Thin Client*. As demonstrated in Figure 3.3, the classification is based on the provided functionality independent of the server-side.

Overview of the comparison between thin and fat client advantages with regard to provided benefits is provided in Table 3.2.

Thin Client	**Fat Client**
Fewer device requirements	Fewer server requirements
Always latest data	Offline working
Future proofing	Better multimedia performance
More portability	More flexibility
Single point of failure	Using existing infrastructure
Lower device's resource consumption	Higher server capacity

Table 3.2.: Comparison of thin and fat client approaches for UbiSA development.

Fat Client Approaches

Fat client approaches, which are also called "rich client approaches", aim to allow access to existing or create new semantically enriched data while using as many required features as possible on the ubiquitous device itself. The basic parts of a UbiSA are triple store, ontologies, reasoner, context-awareness service and front end that provides the user access to underlying components. In case of fat client approach, UbiSA can have all of the described components on the device itself. A fat client still requires periodic connection to a network, but is often characterised by the ability to perform many functions without that connection. Fat client approach has the following advantages:

- *Fewer server requirements.*
  A fat client server does not require high level of performance since the fat client itself does the most of the application processing.

- *Offline working.*
  Fat clients have advantages over thin clients in that a constant connection to the server is often not required.

- *Better multimedia performance.*
  Fat clients have advantages in multimedia-rich applications that would be bandwidth intensive, if fully served.

- *More flexibility.*
  On some operating systems software products are designed for ubiquitous devices that have their own local resources.

- *Using existing infrastructure.*
  As many people now have very fast ubiquitous devices, they already have the infrastructure to run fat clients at no extra cost.

- *Higher server capacity.*
  The more work that is carried out by the client, the less the server needs to do, increasing the number of users each server can support.

The downsides of doing all the work on the ubiquitous device are high resource consumption (and as result decrease in device's battery life) and depending on task demand for better hardware.

Thin Client Approaches

Thin client approaches aim to allow access to existing or create new semantically enriched data on the remote servers while using ubiquitous device as an input-output terminal for user interaction and sensors. In case of thin clients, application on ubiquitous device can have only frontend for the user, context-awareness sensors and means to exchange data with the server, thus being more independent from device's resources and hardware but more sensitive to data connection quality. The exact roles assumed by the server may vary, from providing data persistence (for example, for diskless nodes) to actual information processing on the client's behalf. Thin client approach has the following advantages:

- *Fewer device requirements.*
  A thin client does not require high level of performance from the device since the server does the most of the data processing.

- *Future proofing.*
  Thin clients are likely to remain useful for longer times as the burden of processing is completely on the server.

- *Single point of failure.*
  The server forms a single point of failure for its clients. The security threat model for the software becomes entirely confined to the servers: the clients simply do not run the software.

The downsides of doing all the work on the servers are high demand for server hardware that grows with number of clients and possible issue with single point of failure: any problems with the server will harm many clients.

3.3.4. Quality Attributes

In order to evaluate the strengths and weaknesses of different UbiSAs, we assess the applications according to predefined criteria which we call *Quality Attributes* in this chapter. Quality attributes are non-functional requirements used to evaluate the performance of an application. They are widely used in architecture development and assessment as high level characteristics which applications enclose. In the context of this chapter, quality attributes represent the areas of concern regarding the development of UbiSA from the viewpoint of its consumers.

Based on the qualitative analysis of our primary studies, we obtained 9 quality attributes. For each quality attribute we extracted one or more feature(s). Features describe a specific type or property that can be used to realize an intended quality attribute. The realization features are directly (e.g. faceted browsing) or indirectly (e.g. versioning and change tracking) addressing the required functionalities for UbiSA. Table 3.3 surveys the quality attributes and various approaches for their implementation. In the sequel we describe each of the 9 quality attributes in more detail.

Mobility

Mobility is the ability of application to work on different ubiquitous platforms. Instead of being developed for one platform, ubiquitous semantic application should provide support for as many ubiquitous operating systems as possible.

The following features are proposed for improving the ubiquity of the applications:

- *Cross-device Compatibility.*
  Application should be able to work on the device most relevant to the user, dynamically adjusting to the device's specific features.

- *Device-dependent UIs.*
  Device-dependent UIs allow the generation of different views on the same data and aggregations of the knowledge base based on the ubiquitous device parameters, personal preferences and local policies of the intended users. Such views can be either generic or domain specific. Generic views provide visual representations of data according to certain property values (e.g. map view or calendar view). Domain specific views address the requirements of a particular domain user (e.g. chemists need specific views for visualizing the atomic structure of chemical compounds).

Usability

Usability is a measure of the quality of a user's experience in interacting with an application. In [Lauesen, 2005], usability is defined as consisting of the six factors:

(a) *Fit for use* (or *functionality*). The application can support the tasks that the user has in real life.

(b) *Ease of learning.* How easy is the application to learn for various groups of users?

(c) *Task efficiency.* How efficient is it for the frequent user?

(d) *Ease of remembering.* How easy is it to remember for the occasional user?

(e) *Subjective satisfaction.* How satisfied is the user with the application?

(f) *Understandability.* How easy is it to understand what the application does? This factor is particularly important in unusual situations, for instance error situations or system failures. Only an understanding of what the application does can help the user out.

Ease of use (or user friendliness) is defined as the combination of factors (b) to (f).

Simplicity is the main prerequisite of usability. An UbiSA should, as a rule, hide technical concepts and ontologies from the end users as well as provide (if possible) native for the user's device way of interaction with data. It is crucial to provide end-users with easy to use interfaces that simplify the interaction process and place it in the context of their everyday work. More attention needs to be paid to decrease or blur the gap between the normal interaction process and the interaction process with the semantic content. Ubiquitous semantic applications should focus on users main task. Usually, a user wants to perform the task of browsing or writing data with regard to current context.

The following features are proposed for improving the usability of UbiSAs:

- *Single Point of Entry Interface.*
  It means the environment in which users interact with the semantic data should be integrated (or adjusted to be as close as possible to) with the one in which they usually interact with any other data. So, there is no added user effort involved in interacting with a semantic content versus a conventional approach.

- *Faceted Browsing.*
  Faceted browsing is a technique for accessing a collection of information represented using a faceted classification, allowing users to explore by filtering the available information. In the UI which implements this technique, all property values (i.e. facets) of a set of selected instances are analyzed. If for a certain property the instances have only a limited set of values, those

values are offered to further restrict the instance selection. Hence, this way of navigation through data will never lead to empty results [Auer et al., 2006a]. This feature is useful when searching for available resources or vocabularies especially for ubiquitous devices with limited screen sizes.

- *Inline Resource Editing.*
  Inline editing allows editing items by clicking on them. Such behaviour allows to minimize amount of information on screen which is extremely important on ubiquitous devices.

Customizability

Customizability is the ability of an application to be configured according to users' needs, preferences or context. Instead of being a static form strictly dependent on a given schema, UbiSA should provide a mechanism to tailor its functionalities based on the user's needs or context.

The following features are proposed for improving the customizability of UbiSA:

- *Living UIs.*
  A Living UI is a user interface that configures itself to automatically display the information most relevant to the user, dynamically adjusts to changing data and still allows single users to customize according to their preferences and context [Sakkopoulos, 2009]. End-user development techniques like *personalized UIs* allow inferring user intents as well as present context in real interactions and according to that providing customized outputs.

- *Providing Device-dependent UIs.*
  Device-dependent UIs allow the generation of different views on the same data and aggregations of the knowledge base based on the ubiquitous device parameters, personal preferences and local policies of the intended users. Such views can be either generic or domain specific. Generic views provide visual representations of data according to certain property values (e.g. map view or calendar view). Domain specific views address the requirements of a particular domain user (e.g. chemists need specific views for visualizing the atomic structure of chemical compounds).

Heterogeneity

Heterogeneity is the ability of an application to adapt to different situations or use cases. UbiSA may support a wide range of metadata schemata in a flexible way. In fact, the more flexible and adaptable an application is, the more valuable it is for different contexts and users. A generic UbiSA reduces the costs of supporting new schemata considerably, by following the evolution of existing standards and integrating heterogeneous resources. *Adaptivity* is an important capability of a heterogeneous application. UbiSA could be adaptable to different data sets and use

cases with different kinds of contents to be processed. Two possible ways to adapt to ontology changes exist in UbiSA: runtime adoption or compile time adoption. Runtime adoption assumes on the fly changes of underlying ontology, meaning user or application can change ontology at any time during UbiSA execution. Compile time adoption assumes changes of underlying ontology during the build process, meaning only developer or distributor of the UbiSA can change the ontology prior to distribution. In most of the cases *Heterogeneity* is in opposition to *Usability* of an application. For instance, adding more and more editing possibilities counteracts ease of use for UbiSA [Auer et al., 2006a].

The following features are proposed for improving the heterogeneity of UbiSA:

- *Supporting Multiple Ontologies.*
  A domain is usually described by several ontologies. For example, in a medical context there may be one ontology for general metadata about a patient and other technical ontologies that deal with diagnosis and treatment. UbiSAs need to be able to support multiple ontologies. In a heterogeneous UbiSA, the user interface must be completely decoupled from the ontological models. It should be possible to add models at runtime and become immediately accessible to the users.

- *Supporting Ontology Modification.*
  A heterogeneous UbiSA should provide users with user-friendly interfaces to navigate or modify the structure (classes and properties) of ontologies. In this case, the application also needs to deal with consistency issues which might arise between ontologies and annotations with respect to ontology changes (a.k.a. *Ontology Maintenance*).

Collaboration

Collaboration refers to the ability of a application to support cooperation between different users of the system. UbiSA can support collaborative semantic authoring, where the authoring process can be shared among different authors at different locations. This is a key requirement of knowledge sharing between users from different fields who are contributing to and reusing intelligent documents. Web 2.0 applications and related technologies provide incentives to their users for collaboration and lead to rapidly growing amounts of content. Triggered by the success of the Web 2.0 phenomenon, the *Social Semantic Web* idea has gained momentum yielding tools that allow collaboration and participation incorporating semantics by lay users (e.g. [Aranda-Corral et al., 2009]). As a result, many collaborative and community-driven approaches to semantic content creation have been proposed. Examples are *Semantic Wikis* and *Semantic Tagging Systems* (e.g. OntoWiki Mobile[8]) which exploit Web 2.0 principles and technologies to facilitate broad user participation and collaboration in the process of creating semantically enriched or annotated content.

[8] `http://m.ontowiki.net/`

Access control and *supporting standard formats* are two additional independent prerequisites of collaboration in a ubiquitous semantic applications. The ubiquitous semantic application should allow to distinguish between writeable and non-writeable content based on the users permission level. It also needs to support standard formats which promote the collaboration and make it possible to share and re-use the generated content.

To realize collaboration, UbiSA should provide appropriate UI elements for *meta-level interactions* around different types of semantically created content such as rating, tagging and discussing. Supporting social networking features such as following other authors, watching the evolution of content as well as reusing and re-purposing of content are also important to increase the collaboration in UbiSA.

Accessibility

Accessibility describes the degree to which an application is available to as many people as possible. It can be viewed as the ability to access and benefit from some application. Accessibility is often used to focus on people with disabilities or special needs and their right of access the application. As mentioned in [Hachey, 2011], papers discussing accessibility are clearly lacking in the context of Semantic Web UIs. Accessibility is especially important in area of ubiquitous devices since they can provide *Multimodal User Interactions* [Ringland and Scahill, 2003] (e.g. voice recognition, text to speech, gestures) that might be suitable for different contexts or people with disabilities.

Evolvability

Evolvability is defined as the capacity of a system for adaptive evolution. UbiSA should support evolution of the used data. To achieve this goal, it should take into account the following consistency constraints:

- *Resource Consistency.*
  In cases where several users may edit the same resources, replication issues may occur. It is especially important if the UbiSA allows offline work without synchronization in between client-server data exchange sessions. To address this, UbiSA should use data replication techniques (e.g. [Schandl and Zander, 2009, Ermilov et al., 2011a]) to safely merge data from different users. Otherwise, data from some of the clients might be lost in the process of synchronization.

- *Document and Annotation Consistency.*
  One of the important issues for the design of a semantic authoring environment is to determine how changes should be reflected in the knowledge base of annotated documents and whether changes of ontologies create conflicts with existing annotations [Ermilov et al., 2011a]. Ontologies change sometimes

but some documents change many times. So, it is crucial for a ubiquitous semantic application to track data evolution.

UbiSA should provide appropriate UIs and frameworks for versioning and change tracking to deal with data evolution.

Interoperability

Interoperability is the ability of an application to work and interact with other systems. A ubiquitous semantic application could provide mechanisms to interoperate together with other systems which generate or consume the semantic content created. The following features are proposed for improving the interoperability of ubiquitous semantic applications:

- *Support of Standard Formats.*
  To minimize the problems of interoperability the ubiquitous semantic application should be built on standards. There are already many standards for semantic content serialization (e.g. typical RDF serializations and particular JSON-RDF), representation (e.g. RDF/RDF-S/OWL/RIF and established vocabularies such as SIOC, SKOS, FOAF, rNews, etc.) and exchange (e.g. Linked Data, Web Services, REST). Supporting standard formats and avoiding proprietary formats are essential for compatibility of data with other systems [Ermilov et al., 2011a, Tramp et al., 2011a].

- *Semantic Syndication.*
  Semantic syndication supports the distribution of information and their integration into other applications by providing mechanisms such as *Semantic Atom* [Patel and Khuba, 2009] and *Semantic Pingback*[9] [Auer et al., 2006a].

Scalability

Scalability refers to the capability of an application to maintain performance under an increased work load. UbiSA should support scalability as, for example, the number of users, data or annotations increase. Support of *caching* and implementing a suitable *storage strategy* play an important role in achieving a scalable UbiSA [Auer et al., 2006a, Ermilov et al., 2011a, Tramp et al., 2011a]. Most of the current UbiSA adopt a variety of replication frameworks. In this case, replication framework handles replication of required data to the ubiquitous device for later (e.g. offline) use. A replication framework sometimes poses a redundancy but allows information from heterogeneous resources to be queried centrally and, if it is supported by the framework, even in offline mode.

[9] `http://aksw.org/Projects/SemanticPingBack`

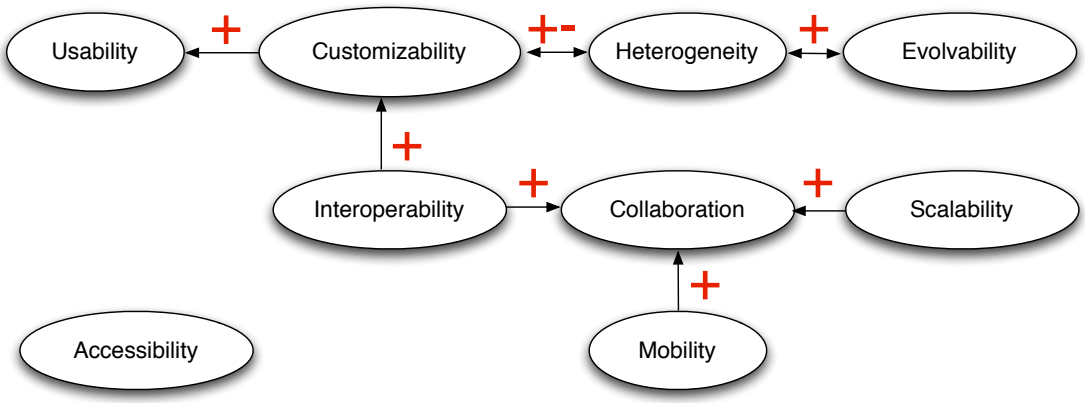


Figure 3.4.: Quality attributes dependencies ('+': positive effect, '+-': reciprocal effect).

3.3.5. Quality Attributes Dependencies

The aforementioned quality attributes are not completely isolated and independent from each other but have overlaps and relations with each other. Figure 3.4 shows an overview of these quality attributes with their inter-relations. Customizability will improve the usability of UbiSA. Customizable applications are configured based on the user needs thereby increasing the overall usability of the application.

Scalability will enhance the level of collaboration since scalable application will support more users and data thereby more collaboration in the system. Interoperability will also enhance the collaboration support of an application, since an interoperable application supports users of different devices. It can also support importing user's data from other devices or systems which will play a positive role in enhancing the customizability. Mobility will also enhance the collaboration support of an application, since an ubiquitous application supports users of different devices.

Evolvability and heterogeneity are directly related. The more evolvable to change an application is, the more heterogeneous it will be and vice versa. Customizability and heterogeneity share a reciprocal relation. A heterogeneous application will decrease its customization and a customizable application needs to focus on specific user needs and thus lacks heterogeneity.

3.3.6. Applications Evaluation

In this section we briefly outline various methods for applications evaluation and report about their usage in the surveyed papers. Table 3.4 lists existing methods for application evaluation adopted from [Chen and Babar, 2011].

Among the primary studies, the majority of studies (24) were using an *Example*

Application as their evaluation method. *Discussion* method was used by variety of papers (13) mostly related to the ontology creation topic. Other papers (11) were focused on algorithms development and thus used *Experiment with Software Subjects* method. The following UbiSA were described in the primary studies:

- *mSpace Mobile* [Wilson et al., 2005b],
- *BuddyAlert* [Grimm et al., 2002],
- *MSSW* [Tramp et al., 2011a],
- *OntoWiki Mobile* [Ermilov et al., 2011a],
- *csxPOI* [Braun et al., 2010],
- *PhotoMap* [Viana et al., 2007],
- *myCampus* [Sheshagir et al., 2004].

These studies were selected to cover as many quality attributes and parts of the generalized architecture as possible.

3.4. Applications

In this section we look at six available UbiSA and compare them according to the quality attributes defined in subsection 3.3.4. Among the applications five (i.e. Ontowiki Mobile, csxPOI, mSpace Mobile, myCampus and Bottari) follow the thin client approach (cf. section 3.3.3) and one (i.e. MSSW) follows the fat client approach (cf. section 3.3.3) for UbiSA. Figure 3.5 summarizes the assessment of the applications according to the defined quality attributes.

3.4.1. OntoWiki Mobile

OntoWiki Mobile[10] section 5.1 is an application that provides support for agile, distributed knowledge engineering scenarios in ubiquitous environments. Ontowiki Mobile facilitates the visual presentation of a knowledge base as an information map, with different views on instance data. It is made for professional users that want to have extensive control over the data.

OntoWiki Mobile architecture is shown in Figure 3.6. The application is implemented as a HTML5 thin client aimed to run in web browser of a ubiquitous device. The backend is developed in PHP using the Zend framework[11]. The backend powered by the Erfurt[12] framework provides support for MySQL database and

[10] `http://m.ontowiki.net`
[11] `http://framework.zend.com/`
[12] `http://aksw.org/Projects/Erfurt`

	OntoWiki Mobile	csxPOI	mSpace Mobile	myCampus	MSSW
Mobility	• Compatible with all web-enabled devices • One UI for all devices	• Compatible only with Android device • Native Android UI	• Compatible only with Win Mobile devices • Native Win Mobile UI	• Compatible with all web-enabled devices • One UI for all devices	• Compatible only with Android devices • Native Android UI
Usability	• Single point of entry UI • Faceted browsing • Inline resource editing	• Single point of entry UI • Inline editing	• Single point of entry UI • Faceted browsing	• Single point of entry UI	• Single point of entry UI
Customizability	• Semantic views: domain specific & generic (e.g. map, calendar)	-	• Semantic views: domain specific & generic (e.g. map)	-	- Supports different WebID providers
Heterogeneity	• Multiple ontology support • Ontology modification support	-	-	-	-
Collaboration	• Access control • Standard formats: RDF, RDFa • Social collaboration UIs: rating and commenting UIs	• Access control	• Social collaboration UIs: rating UIs	• Access control	• Access control • Standard formats: RDF
Accessibility	-	-	-	-	-
Evolvability	• Resource consistency • Versioning & change tracking	• Resource consistency • Versioning & change tracking	• Resource consistency	• Resource consistency	-
Interoperability	• Standard formats: RDF, RDFa • Semantic syndication: semantic pingback	-	-	-	• Standard formats: RDF
Scalability	• Caching support • Storage strategy: backend independent (Mysql, Virtuoso)	• Storage strategy: using a server-side triple store	• Storage strategy: using a server-side triple store	• Storage strategy: using a server-side triple store	• Storage strategy: using a client-side triple store
Main user role	• Professional user	• Engaged user	• Information seeking user	• Information seeking user	• Information seeking user

Figure 3.5.: Comparison of OntoWiki Mobile, csxPOI, mSpace Mobile, myCampus and MSSW according to the quality attributes and user role.

the Virtuoso triple store[13] as storage backends. The user interface is built using jQuery Mobile[14] framework.

Figure 3.7 shows screenshots of OntoWiki Mobile. As a use case OntoWiki Mobile describes on-site data gathering for the bio-diversity knowledge management project *Caucasian Spiders*[15]

The Ontowiki Mobile, as an application with a single point of entry UI, adopts the thin client approach for UbiSA. It provides a semantic search feature with support for faceted browsing. It also supports two complementary knowledge base authoring strategies: a) *Inline editing*, which enables users to edit small information chunks (i.e. statements). b) *View editing*, which enables users to edit common combinations of information (such as an instance of a distinct class) in one single step. In order to do so, OntoWiki Mobile uses the version of RDFAuthor [Tramp et al., 2010b] specially adapted for ubiquitous devices to make generated RDFa views editable. Regarding the customizability, OntoWiki

[13] `http://virtuoso.openlinksw.com/`
[14] `http://jquerymobile.com/`
[15] `http://caucasus-spiders.info/`

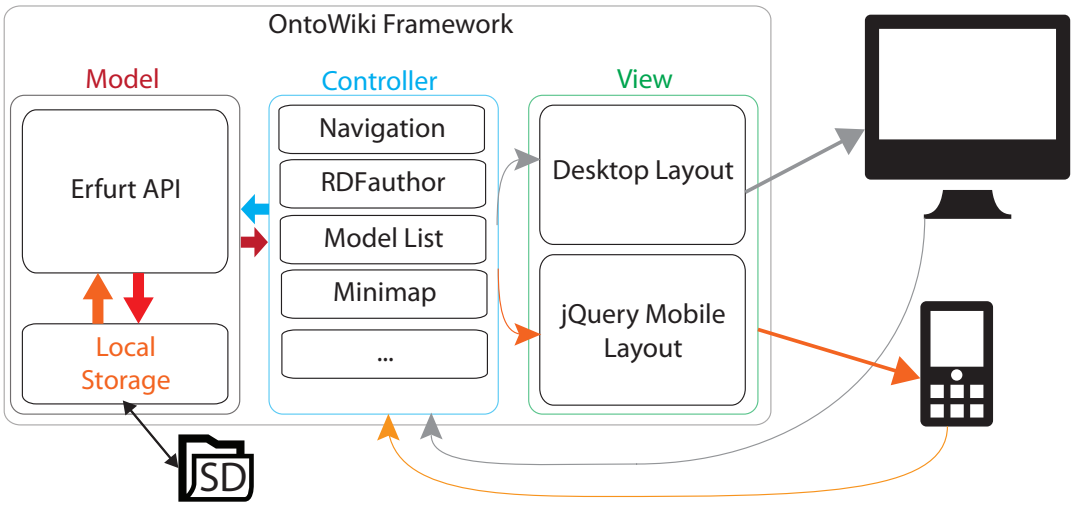


Figure 3.6.: OntoWiki Mobile Architecture (from [Ermilov et al., 2011a]).

Mobile supports different semantic views of the knowledge base which can be generic or domain-specific. It also supports editing multiple ontologies including both the instances and structures of the ontologies. As a Web-based application, it provides cross-browser compatibility but does not provide adoption of UI for different ubiquitous operating systems.

OntoWiki Mobile also provides versioning and evolution features to track, review and selectively roll-back changes which is really important for collaborative ubiquitous applications that might have problems while merging resource from different users. It also supports semantic syndication (employing Semantic Pingback and Linked Data interfaces) to interoperate with other systems. OntoWiki Mobile is backend independent to some extent and supports two different types of storage engines. It also provides a caching component to optimize the performance of the system.

As a drawback, OntoWiki Mobile does not provide any UI elements to facilitate accessibility and automation. It supports only the editing of structured content thus lacking UIs for the annotation of unstructured or semi-structured content.

3.4.2. csxPOI

csxPOI[16] [Braun et al., 2010] (short for: collaborative, semantic and context-aware points-of-interest) is an application that allows its users to collaboratively create, share and modify semantic points of interest (POIs) in ubiquitous environments. It is made for engaged users who want to collaboratively create and share location-based data.

[16] `http://isweb.uni-koblenz.de/Research/systeme/csxPOI`

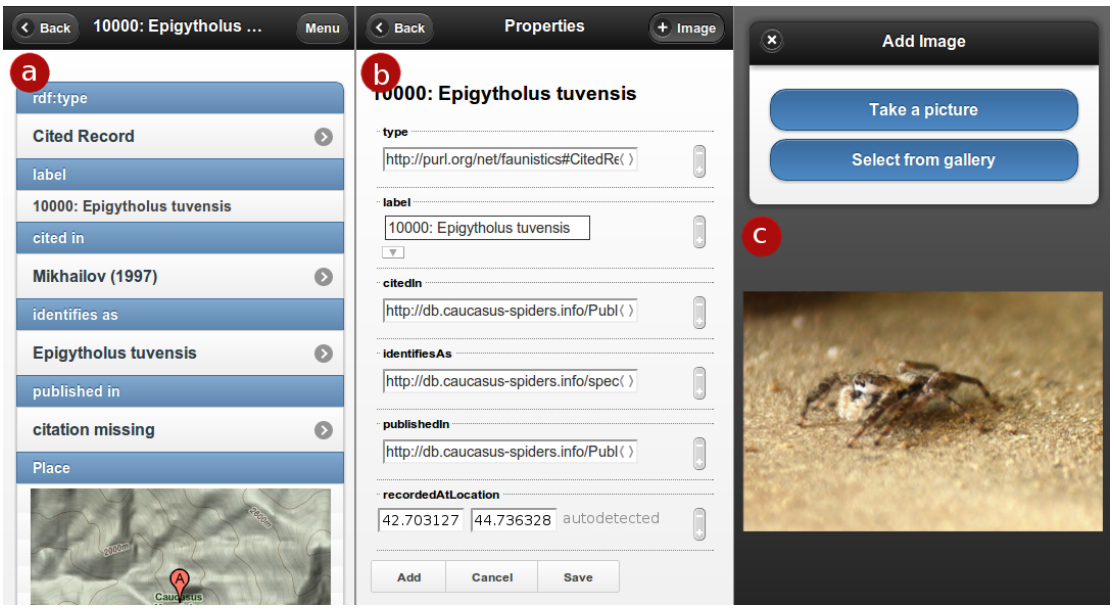


Figure 3.7.: Screenshot of the OntoWiki Mobile. (a) instance view, (b) inline editing, (c) device camera access (from [Ermilov et al., 2011a]).

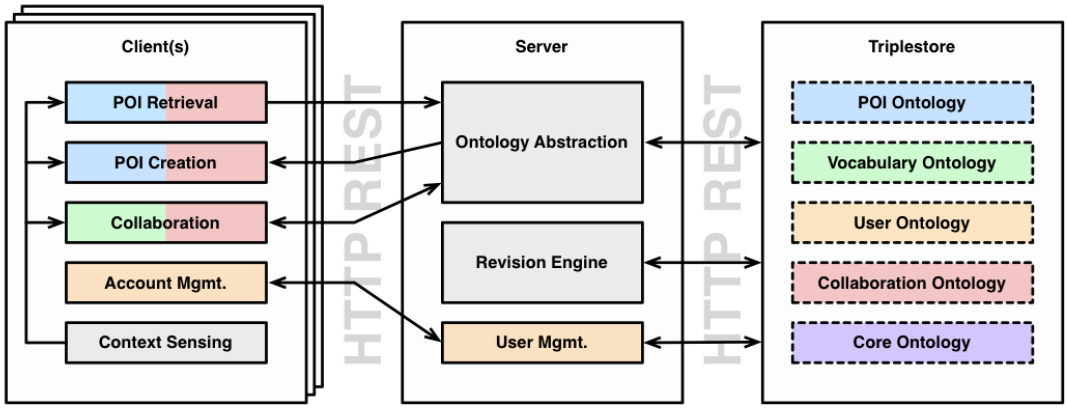


Figure 3.8.: Architecture of csxPOI (from [Braun et al., 2010]).

The architecture of csxPOI is shown in Figure 3.8. The application is implemented as a native Android thin client aimed to deliver access to database of collaboratively created POIs from the ubiquitous device. The backed of csxPOI is organized in a two-tier server architecture. The server consists of an abstraction layer for the collaborative ontology of POIs and provides user management. It also provides a POI revision engine to improve the quality of collaboratively created POIs. The server is implemented as an Apache Tomcat servlet handling the com-

munication with the mobile clients and the triplestore over HTTP. The triplestore is realized as a Sesame web application on top of the same Apache Tomcat web server. Figure 3.9 shows a screenshots of csxPOI.

The csxPOI, as an application with a single point of entry UI, adopts the thin client approach for UbiSA. It provides a way to collaboratively create, share and modify semantic POIs. While working with the POIs, the users implicitly and collaboratively modify and improve an ontology of POI categories underlying the application.

Since collaboratively created semantic POIs inevitably introduce a significant amount of inaccuracy, inconsistency and redundancy, the csxPOI application provides a revision engine that clusters POIs with combinations of spatial, linguistic and semantic similarity measures in order to identify and clean duplicate POIs.

As a drawback, csxPOI does not provide any compatibility with other devices except Android-based ones and there is no support for customizability. csxPOI does not provide any UI elements to facilitate accessibility and automation. It also works only in one specific domain.

3.4.3. mSpace Mobile

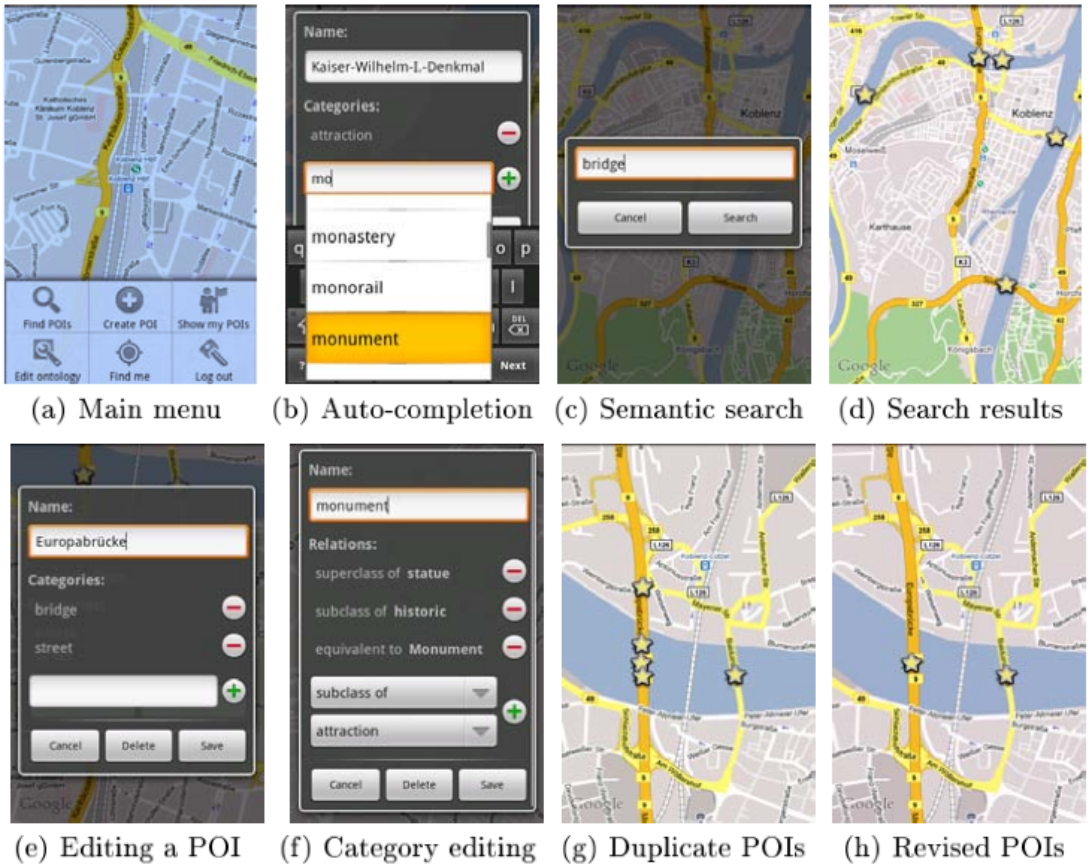
mSpace Mobile [Wilson et al., 2005b] is a Semantic Web application that lets people explore the world around them by leveraging contexts that are meaningful to them in time, space and subject. Especially applicable to those unfamiliar with their surroundings, the application provides information about topics of chosen interest, based upon the location, as determined by an optional GPS receiver.

Architecture of mSpace Mobile is shown in Figure 3.10. The application is implemented as native Windows Mobile thin client aimed to deliver access to database of topics of interest from ubiquitous device. mSpace Mobile is organized in a three-layer architecture: the mSpace Mobile application (MA) queries the mSpace Query Service (MQ), which is connected to RDF triplestore knowledge interfaces (MK). The architecture is designed to query multiple triplestores, as well as support incorporation of resources, which may not yet be referenced in triplestores. It also abstracts the internal concepts of query generation and triplestore querying, taking load away from ubiquitous devices. Figure 3.11 shows a screenshots of mSpace Mobile.

The mSpace Mobile, as an application with a single point of entry UI, adopts the thin client approach for UbiSA. It provides a way to access the location-based information while on the move. The mSpace Mobile interface is designed to let users of ubiquitous devices run complex queries through direct manipulation without typing.

The five key features of the mSpace Mobile interaction model are:

- *A spatial browser*, maintaining persistent display of domain dimensions, for browsing information within a domain.



(a) Main menu (b) Auto-completion (c) Semantic search (d) Search results

(e) Editing a POI (f) Category editing (g) Duplicate POIs (h) Revised POIs

Figure 3.9.: Screenshots of csxPOI application showing its different features (from [Braun et al., 2010]).

- *User-determined organization* of dimensions presented: they can be added, removed and swapped.
- *Information area*, providing contextual information about selected items in the column browser.
- *Preview cues*, which provide typical examples of information within a domain.
- *Triage area*, allowing the user to save items from the domain that are of particular interest for further exploration in the future.

As a drawback, mSpace Mobile does not provide any compatibility with other devices than Windows Mobile-powered ones as well as there is no customizability support provided in any way. mSpace Mobile also works only in one specific domain and does not allow any editing of the data (except for rating the results).

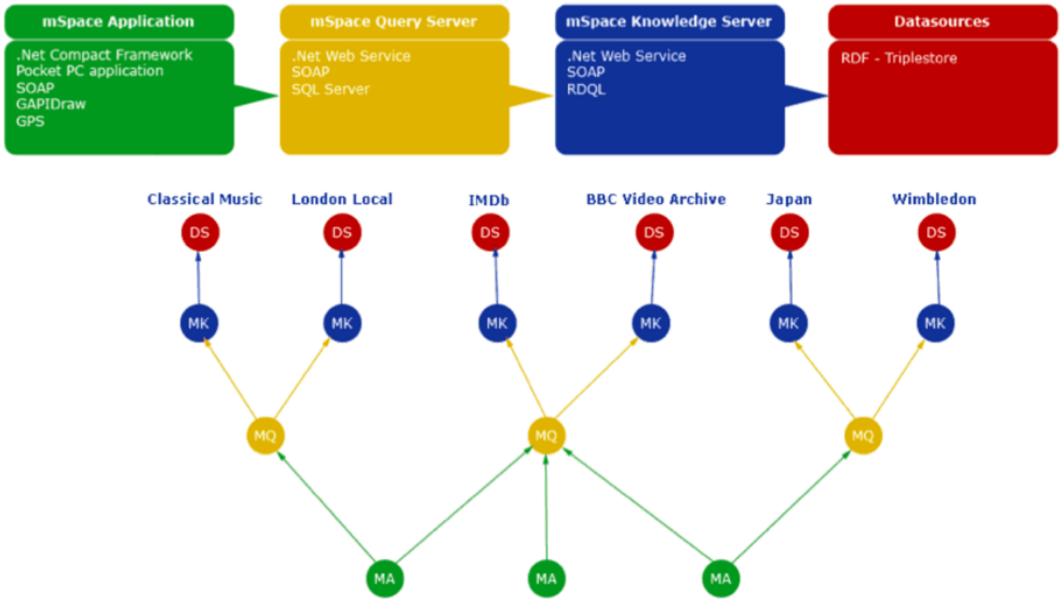


Figure 3.10.: Architecture of mSpace Mobile (from [Wilson et al., 2005b]).

3.4.4. myCampus

myCampus [Sheshagir et al., 2004] is a context-aware semantic web environment aimed at enhancing everyday campus life. In myCampus, users can, over time acquire or subscribe to a variety of different sets of task-specific agents that assist them in context of different tasks (e.g. scheduling meetings, sharing documents, organizing evenings out, filtering and routing incoming messages, etc.). It is made for users aiming to learn or access information depending on the context.

The architecture of myCampus is shown in Figure 3.12. In myCampus, sources of contextual information are represented as Semantic Web Services. This means that each source of contextual information is described by a profile that describes its functional properties in relation to one or more ontologies. Service descriptions also include details about how to invoke a service (e.g. input, output and preconditions).

myCampus, as an application with a single point of entry UI, adopts the thin client approach for UbiSA. It provides an environment, where relevant sources of contextual information about a user can automatically be discovered and accessed in support of different queries. This approach makes it possible to accommodate users that rely on different sets of contextual resources (e.g. different calendar systems, different sources of location information, etc.) and to adapt to situations where sources of contextual information for a user may change over time (e.g. different location tracking services depending on where the user is). myCampus agents can range from simple agents that rely on one or more sources of contextual information about their users to more complex agents that are capable of dynamically building

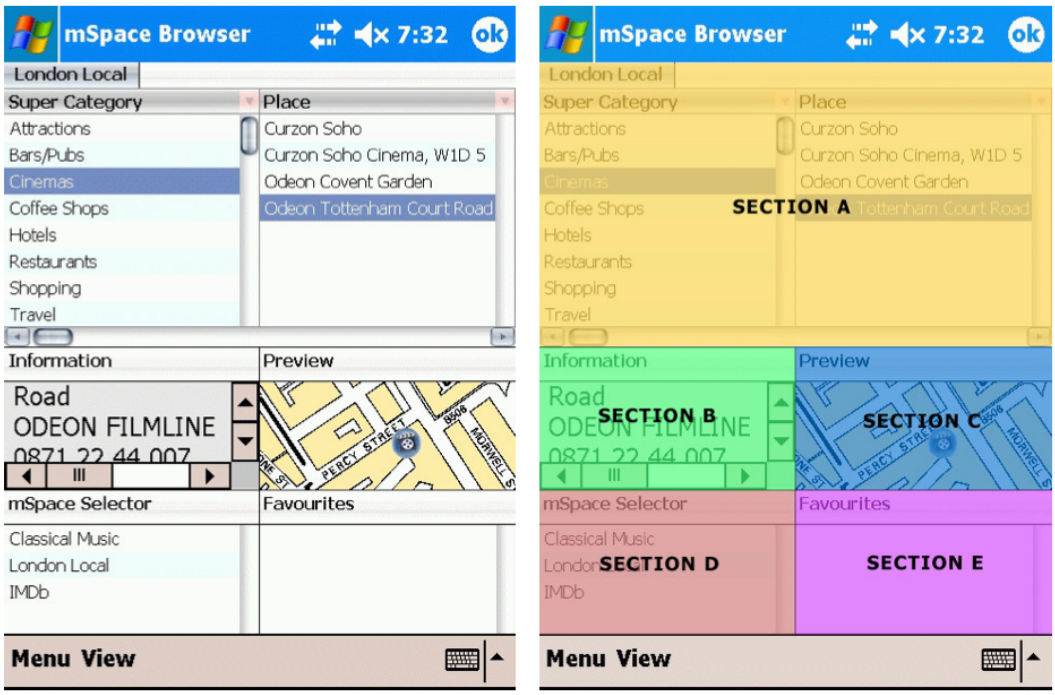


Figure 3.11.: Screenshots of mSpace Mobile application. There are five features within the user interface: A – the columnar mSpace browser; B – the information box; C – a preview cup map; D – an mSpace selector and E – a favourites list (from [Wilson et al., 2005b]).

plans in response to requests from their users.

As a drawback, myCampus does not provide any UI elements to facilitate accessibility and automation as well as there is no customizability support provided in any way. myCampus also works only in one specific domain and does not allows any editing of the data.

3.4.5. MSSW

MSSW[17] section 5.3 is an Android-based Social Semantic Web client as well as a contacts provider, which integrates the distributed FOAF/WebID social network[18] into a ubiquitous device. It is made for users aiming to access information from their social graph.

The application is implemented as a native Android fat client. The overall MSSW architecture is depicted in Figure 3.14. It integrates a number of vocabularies, protocols and technologies. The semantic representation of personal information

[17] http://aksw.org/Projects/MobileSocialSemanticWeb
[18] http://www.w3.org/wiki/WebID

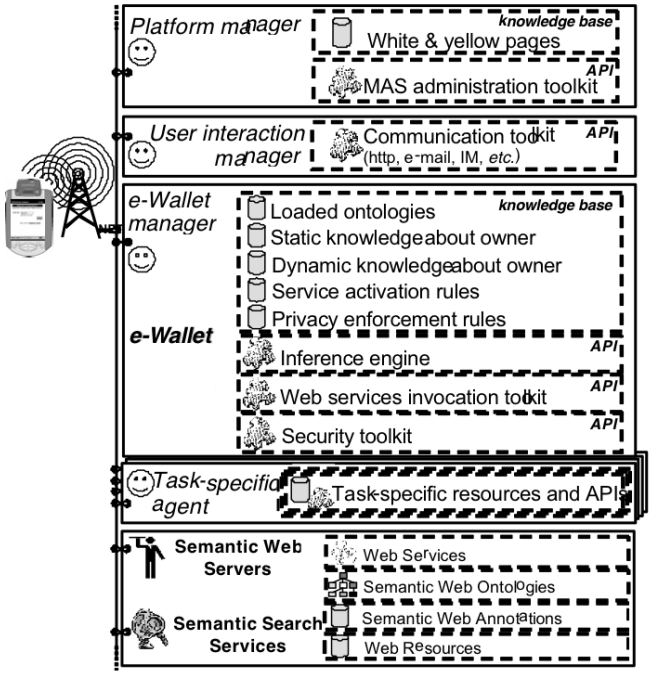


Figure 3.12.: myCampus architecture: a user's perspective [Sheshagir et al., 2004]

is facilitated by the WebID protocol and vocabulary. *FOAF+SSL* allows the use of a WebID for authentication and access control purposes. *Semantic Pingback* facilitates the first contact between users of the social network. The subscription services based on *PubSubHubbub* allow obtaining specific information from people in ones social network as near-instant notifications. The MSSW application itself consists of two application frameworks, which are built on top of the Android runtime and a number of libraries. In particular, *androjena* is one of those libraries, which itself is a partial port of the popular Jena framework to the Android platform (cf. Table 3.1). MSSW uses the included Jena rules[19] engine to transform the retrieved WebID statements into Android-specific data structures which are well suited for a straightforward import into the Android contacts provider.

Figure 3.15 shows screenshots of MSSW. MSSW provides a way to traverse a user's FOAF network, synchronize the user's contact with contact book on the device, add/remove relations or search for profiles using the Sindice [20] search engine. As a drawback, MSSW is not compatible with non-Android devices. Also, there is no customizability support provided in any way. MSSW works only in one specific application domain but it is possible to adjust the data with custom Jena rules.

[19] `http://jena.sourceforge.net/inference/#RULEsyntax`
[20] `http://sindice.com/`

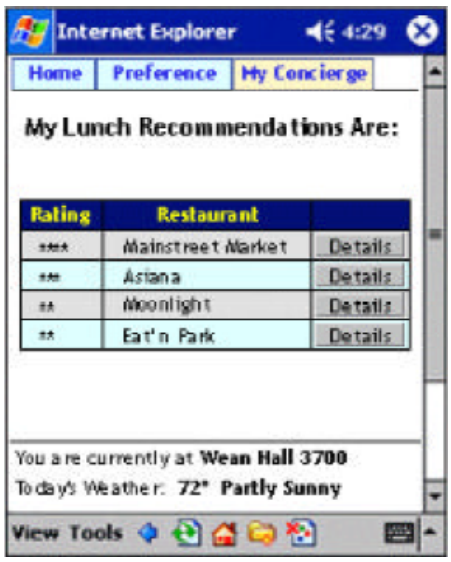


Figure 3.13.: Screenshot of myCampus (from [Sheshagir et al., 2004]).

3.4.6. Bottari

Bottari[21] [Celino et al., 2011] is an Android-based application that exploits social media and context to provide point of interest (POI) recommendations to user in a specific geographic location. It is made for users aiming to discover nearby points of interest on the basis of user's tastes and influencing people's opinion.

The application is implemented as a native Android thin client. The overall Bottari architecture is depicted in Figure 3.16. It integrates a number of vocabularies, protocols and technologies. Semantic information retrieval is used to get interesting POIs close to user's location. Sentiment analysis of posts from social media is used to get nearby POIs that are popular among other people. Stream reasoning is used to get emerging POIs that are getting a lot of traction. Finally, inductive reasoning on social media is used to compute personalized recommendations. The Bottari application itself consists of an application framework that is built on top of the Android runtime and a number of libraries. In particular, *androjena* is one of those libraries, which itself is a partial port of the popular Jena framework to the Android platform (cf. Table 3.1).

Figure 3.17 shows screenshots of Bottari.

Bottari provides a way to get personalized POI recommendations on Android tablets, to help users find their way when they are in a specific location. As a drawback, Bottari is not compatible with non-Android devices. Also, there is no customizability support provided in any way. As well Bottari works only in one specific application domain.

[21] `http://larkc.cefriel.it/lbsma/bottari/`

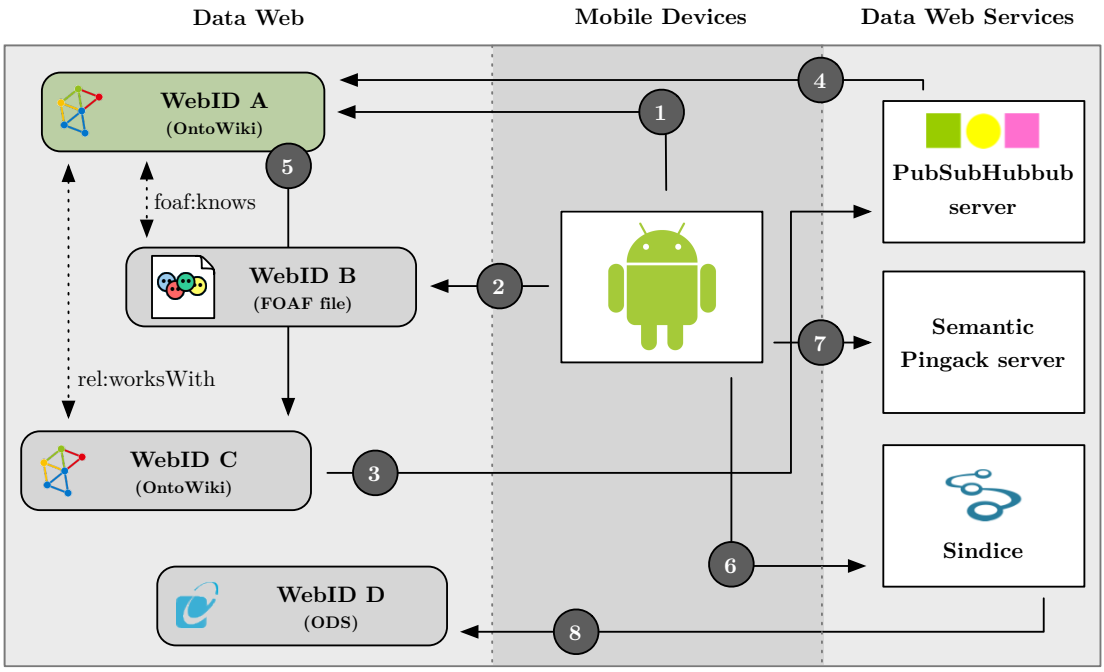


Figure 3.14.: Architecture of a distributed, semantic social network: (1) A mobile user may retrieve updates from his social network via his WebID provider, e.g. from OntoWiki. (2) He may also fetch updates directly from the sources of the connected WebIDs. (3) A WebID provider can notify a subscription service, e.g. a PubSubHubbub server, about changes. (4) The subscription service notifies all subscribers. (5) As a result of a subscription notification, another node can update its data. (6) A mobile user can search for a new WebID by using a semantic search engine, e.g. Sindice. (7) To connect to a new WebID he sends a Pingback request which (8) notifies of the resource owner (from [Tramp et al., 2011a]).

3.5. Research and Technology Challenges

The results of our systematic review revealed several research and technology gaps and corresponding challenges with regard to the development of UbiSA. To the best of our knowledge, none of the challenges outlined in this section was so far addressed in existing research in any way.

Accessibility. Addressing accessibility issues during the design of UbiSA UIs and providing special UIs that utilize different input methods available on ubiquitous devices are very important aspects of UbiSA development. Addressing these issues can be beneficial not only for people with disabilities and special needs, but also for people that access UbiSA in special contexts (e.g. using voice control while

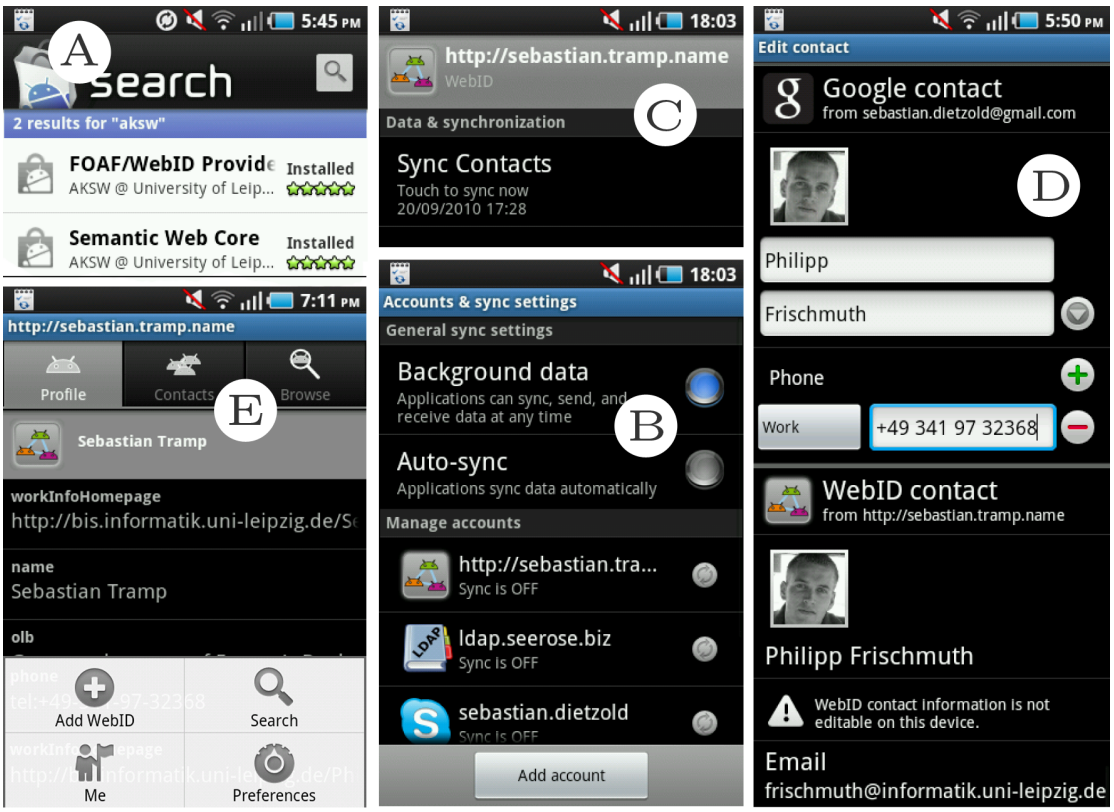


Figure 3.15.: Screenshots of the Mobile Social Semantic Web Client, the FOAF Browser and the Android components which integrate the WebID account: (A) The client as well as the triple store can be found in the official Google application market. (B) After installation, users can add a WebID account the same way they add an LDAP or Exchange account. (C) The account can be synchronized on request or automatically. (D) A contacts profile page merges the data from all given accounts. (E) By using the FOAF browser, people can add contacts or browse the contacts of their friends (from [Tramp et al., 2011a]).

driving).

Semantic disambiguation. Usage of Linked Data as background knowledge for disambiguation can simplify user interactions with ubiquitous devices in several ways. For example, search can be significantly improved and simplified by mapping keywords (in text or voice input) to URIs and using them for creating formal queries (e.g. in SPARQL).

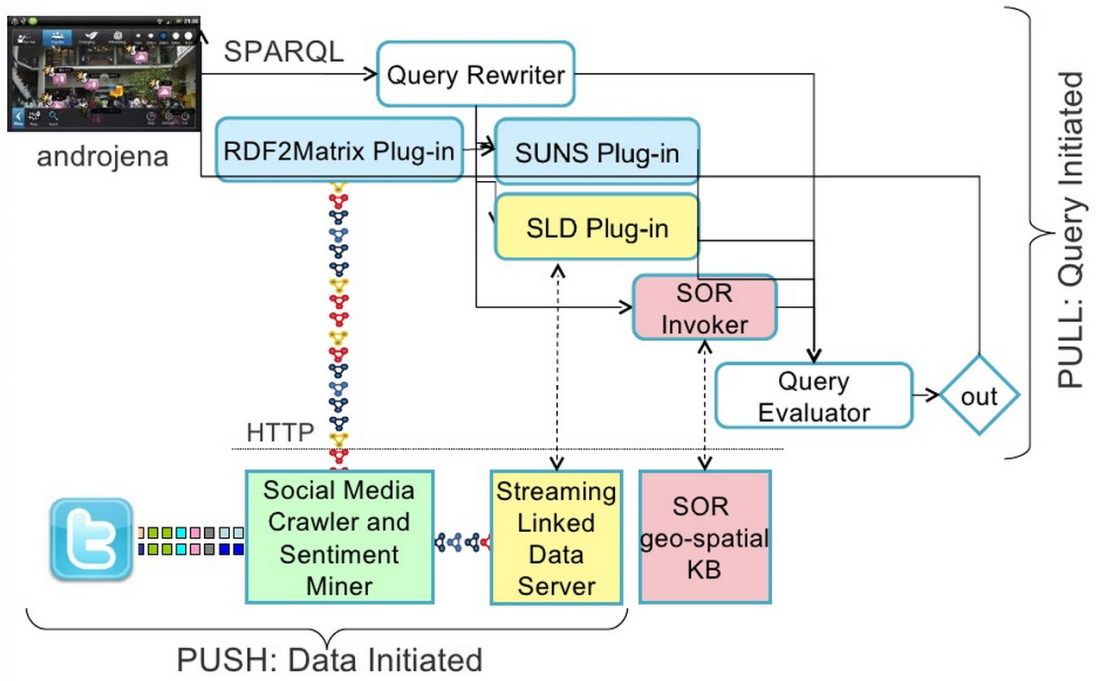


Figure 3.16.: Bottari architecture.

Facilitating entertainment. Serious games on ubiquitous devices can bring entertainment to people seeking it while solving real problems. The challenge is to define game templates, which are tailored to mobile devices, employ or producing semantics and can be played in a casual way (e.g. while waiting for a bus). For example, a paper chase game could use background knowledge about points-of-interest such as LinkedGeoData, for generating verification tasks (e.g. location, opening hours etc.).

Making complex algorithms accessible. Bridging between complex algorithms and easily accessible functionality is one more important research field. The amount of complex algorithms developed by various research communities to solve different problems is increasing. But, there is only a small amount of research on how to make those complex algorithms easy for users to interact with. One possible approach for addressing this issue, could be a market place for algorithms and UIs. The market place could provide clearly defined interfaces for interacting with information in certain application scenarios. Different information object recognition algorithms (e.g. specialized on faces, buildings etc.), for example, could be dynamically plugged-into a UbiSA UI providing image annotation to a user.

Augmented reality. Augmented reality is one more gap in the research. With advancements of ubiquitous device's camera sensors and image recognition technolo-

Figure 3.17.: Screenshots of BOTTARI: (a) augmented reality display of recommended POIs, (b) POI selection and (c) visualization of the selected POI details, (d) trends in user sentiment about the POI.

gies it is becoming possible to detect the context and display information about the surroundings in better way. Projects like *Layar*[22] or *Google Goggles*[23] can benefit from semantic datasets like LinkedGeoData [Auer et al., 2009] or Europeana[24] for augmenting reality on ubiquitous devices with semantic background information.

3.6. Conclusions

In this chapter, we reported the results of a systematic literature review on ubiquitous semantic applications comprising initially of 172 papers. The review aimed to answer the four research questions defined in Section subsection 3.2.1 by thorough analysis of the 48 most relevant papers. Before addressing the defined research questions, we drew a terminology which defines the basic concepts used in the literature as well our survey. To answer the RQ1, we classified existing approaches for ubiquitous semantic applications into two categories namely *Fat Client* and *Thin Client* approaches discussed in Section subsection 3.3.3. Furthermore, our data analysis revealed a set of 9 quality attributes for ubiquitous semantic applications together with the corresponding features suggested for their

[22] http://www.layar.com/
[23] http://www.google.com/mobile/goggles/
[24] http://europeana.eu/

realization. These quality attributes plus the features were used to answer the RQ2 and RQ3. In order to answer RQ4, we extracted the main user roles as well as evaluation methods discussed in the primary studies and reflected the results in subsection 3.3.6. Additionally, to show the applicability of the results, we performed an in-depth comparison of six existing ubiquitous semantic applications according to the defined quality attributes and described their strengths as well as their weaknesses. Essenthe foundational quality attributes for a ubiquitous semantic applications are: mobility, usability, heterogeneity, customizability and evolvability. A basic ubiquitous semantic application should fulfill a reasonable level of user-friendliness and adopt to different situations or use case while providing mechanisms to tailor its functionality based on specific user needs. It should also take into account issues such as resource consistency over different clients as well as cross-device compatibility. Support of collaboration, interoperability and scalability are quality attributes required when UbiSA is employed in a community-driven environment with large amount of users, systems and interactions. A UbiSA should support standard formats and provide appropriate UI elements for social networking including both human-to-human as well as system-to-system interactions. Additionally, it should maintain performance under an increased work load by supplying appropriate storage and caching mechanisms. Also, it should take into account possible issues with data connection and hardware related ubiquitous device limits. Accessibility is, as our survey indicates, not well addressed by the literature so far. Furthermore, providing accessible UIs for people with disabilities or special needs is another requirement which should be taken into account when designing ubiquitous semantic applications.

While there are many benefits of systematic reviews, they also bear some limitations and validity threats originating from human errors. The main threats to validity of our systematic review are twofold: correct and thorough selection of the studies to be included as well as accurate and exhaustive selection of quality attributes together with their corresponding features. With the increasing number of works in the area of ubiquitous semantic applications, we cannot guarantee to have captured all the material in this area. The scope of our review is restricted to the scientific domain. Therefore, some tools or approaches employed in the industry might have not been included in our primary studies. Furthermore, since the review process was mainly performed by one researcher, a bias is possible. In order to mitigate a potential subjective bias, the review protocol and results were checked and validated by a senior researcher and other colleagues experienced in the context of Semantic Web. We see this effort and in particular the identification of a comprehensive set of quality attributes as a crucial step towards developing more effective and user-friendly ubiquitous application for accessing the Social Semantic Web. New approaches and applications can be evaluated in the light of these quality attributes, thus revealing additional aspects to be taken into consideration. As a result, more user-friendly applications will enable more people to interact with the Semantic Web thereby facilitating the realization of the intelligent Web vision.

Quality Attribute	Realization
Mobility	Cross-device Compatibility [Ermilov et al., 2011a, Sheshagir et al., 2004, Soylu et al., 2012], Device-dependent UIs [Tramp et al., 2011a, Braun et al., 2010, Wilson et al., 2005b, Viana et al., 2007, Celino et al., 2011]
Usability	Single Point of Entry Interface [Ermilov et al., 2011a, Sheshagir et al., 2004, Tramp et al., 2011a, Wilson et al., 2005b, Viana et al., 2007], Faceted Browsing [Ermilov et al., 2011a, Braun et al., 2010], Inline Resource Editing [Ermilov et al., 2011a, Braun et al., 2010]
Customizability	Living UIs [Soriano et al., 2006], Providing Device-dependent UIs [Tramp et al., 2011a, Braun et al., 2010, Wilson et al., 2005b, Viana et al., 2007, Bellini et al., 2012, Ostuni et al., 2013], Supporting Multiple Data Sources [Ermilov et al., 2011a, Van Woensel et al., 2011b, Van Woensel et al., 2011a]
Heterogeneity	Supporting Multiple Ontologies [Ermilov et al., 2011a, Sheshagir et al., 2004, Wilson et al., 2005b], Supporting Ontology Modification [Ermilov et al., 2011a, Braun et al., 2010]
Collaboration	Access Control [Tramp et al., 2011a, Braun et al., 2010, Ermilov et al., 2011a, Sheshagir et al., 2004], Support of Standard Formats [Wilson et al., 2005b, Braun et al., 2010, Ermilov et al., 2011a, Tramp et al., 2011a], UIs for Social Collaboration [Braun et al., 2010, Ermilov et al., 2011a]
Accessibility	Accessible UIs, Multimodal UIs [Ringland and Scahill, 2003]
Evolvability	Resource Consistency [Tramp et al., 2011a, Ermilov et al., 2011a, Braun et al., 2010], Versioning and Change Tracking [Ermilov et al., 2011a, Tramp et al., 2011a, Schandl and Zander, 2009, Sun et al., 2005, Braun et al., 2010]
Interoperability	Support of Standard Formats [Wilson et al., 2005b, Braun et al., 2010, Ermilov et al., 2011a, Tramp et al., 2011a], Semantic Syndication [Tramp et al., 2011a]
Scalability	Replication Support [Tramp et al., 2011a, Sun et al., 2005, Schandl and Zander, 2009], Suitable Storage Strategies [Ermilov et al., 2011a, Tramp et al., 2011a, Schandl and Zander, 2009, Sun et al., 2005]

Table 3.3.: List of quality attributes together with their corresponding features suggested for UbiSA.

Evaluation Method	Definition
Rigorous Analysis	Rigorous derivation and proof, suited for formal model.
Case Study	An empirical inquiry that investigates a contemporary phenomenon within its real-life context; when the boundaries between phenomenon and context are not clearly evident; and in which multiple sources of evidence are used.
Discussion	Provided some qualitative, textual, opinion-oriented evaluation. e.g., compare and contrast, oral discussion of advantages and disadvantages.
Example Application	Authors Describe an application and provide an example to assist in the description, but the example is "used to validate" or "evaluate" as far as the authors suggest.
Experience	The result has been used on real examples, but not in the form of case studies or controlled experiments, the evidence of its use is collected informally or formally.
Field Experiment	Controlled experiment performed in industry settings.
Experiment with Human Subjects	Identification of precise relationships between variables in a designed controlled environment using human sub jects and quantitative techniques.
Experiment with Software Subjects	A laboratory experiment to compare the performance of newly proposed system with other existing systems.
Simulation	Execution of a system within artificial data, using a model of the real word.

Table 3.4.: Application evaluation methods.

4. A conceptual framework for ubiquitous semantic applications

This chapter describes the conceptual framework for ubiquitous semantic applications. This chapter also includes the meaning (definition) of a *Ubiquitous semantic application*. Additionally, a generalized architecture of ubiquitous semantic application is described along with how different approaches fit in it. Finally, a number of factors that allow classification of UbiSA are discussed.

4.1. Definition of the ubiquitous semantic applications

To define *Ubiquitous Semantic Application* as a term, we first have to define each separate piece of the term. *Ubiquitous Semantic Application* term can be split into three independent components (or sub-terms):

- *Ubiquitous* - referring to ubiquitous devices that are used by the end-user
- *Semantic* web - referring to semantically enriched data generated or received by the end-user from the web
- *Application* - referring to the software application an end-user interacts with

In the following paragraphs we define each term in more detail.

Ubiquitous computing (or ubicomp) is a paradigm for interaction between people and computers. According to [Salber et al., 1998], a guiding principle of ubicomp is to break away from desktop computing to provide computational services to a user when and where required. A ubiquitous computing system usually consists of:

- a (possibly heterogeneous) set of computing devices
- a set of supported tasks
- some optional infrastructure (e.g., network, GPS location service) the device may rely on to carry out the supported tasks

Two main inherent features of a system that make it a ubiquitous computing system, according to [Weiser, 1991] are:

- *ubiquity* - interaction with the system is available wherever the user needs it
- *transparency* - the system is non-intrusive and is integrated into the everyday environment

The best examples of ubiquitous devices in modern world are smartphones or tablet computers.

Semantic web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation (as was discussed in section 2.1). In other words, Semantic Web allows the machines not only to present data but also to process it.

Semantic Web depends on several technologies including Resource Description Framework (RDF) and Uniform Resource Identifiers (URIs). In-depth description for each of these technologies was given in chapter 2.

Application is a collection of computer programs and related data. A computer program is a sequence of instructions, written to perform a specified task with a computer. [Stair and Reynolds, 2011] The program has an executable form that the computer can use directly to execute the instructions. The same program in its human-readable source code form, from which executable programs are derived (e.g., compiled), enables a programmer to study and develop its algorithms.

4.1.1. Definition

Taking into account all the individual piece descriptions given above, we define *Ubiquitous Semantic Application* as the computer software implemented specifically for ubiquitous devices and designed to help the user to perform specific tasks that satisfy the following requirements:

- application is designed and developed specifically for (or with respect to) ubiquitous devices
- application utilizes semantic data during the work process in any way

A ubiquitous semantic application provides a human accessible interface with capabilities for reading, writing or modifying semantic documents. Semantic documents facilitate a number of important aspects of information management:

- For *search and retrieval*, enriching documents with semantic representations helps to create more efficient and effective search interfaces, such as faceted search (e.g. in [Ermilov et al., 2011a]) or question answering (e.g. [Shekarpour et al., 2013]). Ultimately, users are empowered to fight the increasing information overload and gain better access to relevant documents and answers related to their information needs.

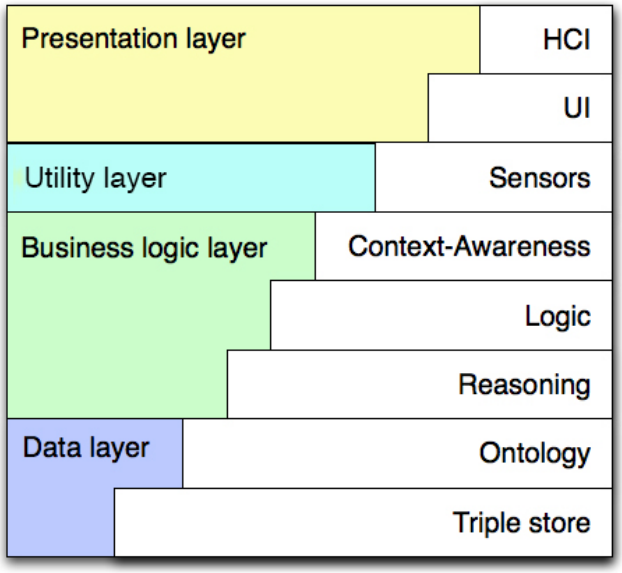


Figure 4.1.: Conceptual framework of a generic Ubiquitous Semantic Application.

- In *information presentation*, semantically enriched documents can be used to create more sophisticated ways of flexibly visualizing information, such as geospatial maps [Viana et al., 2007, Braun et al., 2010, Wilson et al., 2005b].
- For *information integration*, semantically enriched documents can be used to provide unified views on heterogeneous data stored in different applications by creating composite applications such as semantic mashups as discussed in [Wilson et al., 2005b, Ermilov et al., 2011a].
- To realize *personalization*, semantic documents provide customized and context-specific information which better fits the user needs and will result in delivering customized applications such as personalized semantic portals (e.g. [Ruta et al., 2010a, WeiBenberg et al., 2006]).
- For *reusability* and *interoperability*, enriching documents with semantic representations (e.g. using SKOS[1] or Dublin Core[2] vocabularies) facilitates exchanging content between disparate systems.

4.2. Architecture

A conceptual framework for ubiquitous semantic applications is depicted in Figure 4.1. In the sequel, we describe the conceptual framework terminology as well

[1] `http://www.w3.org/2004/02/skos/`
[2] `http://www.cs.umd.edu/projects/plus/SHOE/onts/dublin.html`

as individual layers, components and concepts in more detail.

4.2.1. Presentation layer

Presentation layer contains components that directly interact with the end-user of the UbiSA. The layer comprises of two tightly interacting components - Human Computer Interaction (HCI) and User Interface (UI).

HCI (Human Computer Interaction) is crucial for the development of UbiSA and represents a research field that aims at improving the interactions between users and computers by making computers more usable and receptive to the user's needs. HCI involves the study, planning and design of the interaction between end-users and the computers. This field is often regarded as the intersection of computer science, behavioural sciences, design and several other fields of study.

UI (User Interface) is an abstraction, which simplifies the usage of underlying components by providing a user-friendly interface for HCI. The user interface includes hardware and software components. User interface provides a means of input, allowing the users to manipulate a system and output, allowing the system to indicate the effects of the users' manipulation. Generally, the goal of HCI engineering is to produce a UI which gives a self exploratory, efficient and user friendly way to interact with a machine such that it produces the desired result.

4.2.2. Utility layer

Utility layer contains miscellaneous components that helps the UbiSA in carry out its tasks. Those components can be Sensors, security measures, configurations, etc.

Sensors are another important part of UbiSA. They allow converting measures of a physical quantity into a signal which can be read by an UbiSA. Sensors are already widely used in everyday ubiquitous devices such as touch sensors, accelerometers and GPS.

4.2.3. Business logic layer

Business logic layer contains components that encode the rules that determine how the data can be created, displayed, stored and changed.

Context-Awareness is a term coined to describe applications that can passively or actively determine their context by utilizing on-board or peripheral device sensors. Especially in mobile and ubiquitous usage scenarios, context-awareness is a crucial aspect, since the device is used in very specific and distinct situations (e.g.

while walking or sitting in a restaurant). Also, mobile and ubiquitous devices have a variety of sensors not available in desktop computers, such as GPS, accelerometer, gyroscope, compass, etc. (a detailed overview is provided in [Lane et al., 2010]).

Logic is the application specific coordination of domain and infrastructure components according to the requirements of the particular UbiSA. It handles most of the interactions within the application itself as well as interaction with the external components, be it software or hardware.

Reasoning is a component whose function is to generate conclusions from the available knowledge using logical techniques of deduction, induction or other forms of reasoning. Reasoning systems are a subset of a broader category of intelligent systems. A reasoning system manipulates previously acquired knowledge in order to generate new knowledge. Reasoning systems automate the process of inferring or otherwise deriving new knowledge via the application of logic.

4.2.4. Data layer

Data layer contains components which provide simplified access to the data stored in persistent storage.

Ontology is a formal, explicit specification of a shared conceptualisation that represents knowledge as a set of concepts within a domain as well as relationships between those concepts. Ontology is domain specific; it does not represents all knowledge areas, but one specific area of knowledge. An ontology defines a set of classes (e.g. “Person”, “Book”, “Writer”) and their hierarchy, i.e. which class is a subclass of another one (e.g. “Writer” is a subclass of ”Person”). Additionally, the relationship between the classes is defined, i.e. how different classes are connected to each other via properties (e.g. a “Book” has an author of type “Writer”).

Triplestore is a specific database for the storage and retrieval of information adhering to the RDF data model, i.e. triples composed of subject-predicate-object (e.g. “Bob is 35” or “Bob knows Fred”). Most triplestores support SPARQL query language for querying RDF data.

4.3. Classification of ubiquitous semantic applications

In this section, we discuss a number of factors that allow us to classify UbiSAs. The classification is mainly based on [Martin and Auer, 2010]. Below we give an overview of the factors with a brief explanation for each of them.

4.3.1. Device type

The very basic classification of UbiSA can be done by the type of device that the specific UbiSA was implemented for. Firstly, they can be simple *Radio-frequency identification* (RFID) tags, or more complex *Sensors* or *Sensors Systems*. Secondly, there are more complex *Embedded* systems and *Mobile* or *Smart* devices, that are most commonly used for UbiSA development. And finally, they can be *Desktop* or *Server* devices.

4.3.2. Client-server workload balancing

As mentioned earlier, UbiSA can be developed using three different approaches - fat, thin or hybrid. Utilizing the conceptual framework that was described before in section 4.2, we can strictly define each of these approaches. Considering that UbiSA has a typical client-server architecture, we define each of these approaches below.

Fat client approach means that all or most of the conceptual framework layers are located and handled by the client application on the ubiquitous device.

Thin client approach means that all or most of the conceptual framework layers are located and handled by the server, while the client application on the ubiquitous device only deals with UI and HCI.

Hybrid client approach means that some of the conceptual framework layers are located and handled by both - the client application on the ubiquitous device and the server.

4.3.3. Semantic technology depth

This categorisation dimension aims to capture to which degree the architecture of an UbiSA makes use of semantic technologies. Generally, UbiSA use semantic technologies in two different ways – externally and/or internally:

Extrinsic UbiSA make use of semantic knowledge representation formalisms on the surface of the application in order to facilitate the interaction and integration with other UbiSA and technologies. Implementation-wise, extrinsic UbiSA are easy to realise, since conventional mobile application development technologies and design patterns can be used.

Intrinsic UbiSA make direct internal use of semantic representations for their original application architecture. Here the situation is more complicated than with solely extrinsic UbiSA, since conventional technologies have to be complemented

or replaced by their Semantic Web equivalents. On the persistence layer, relational databases have to be replaced by triple stores. On the API layer, ObjectRelational-Mapping (ORM) techniques have to be replaced by corresponding APIs, which provide higher-level functions for handling RDF, RDF-Schema and OWL.

4.3.4. Information flow direction

The class of extrinsic UbiSA can be further refined into UbiSA, which produce, consume or both produce and consume semantic representations.

Producing UbiSA: Based on either an intrinsic semantic information representation or on the mapping of other data models to RDF (as discussed in the previous section 4.2), four different types of Semantic Web interfaces can be distinguished:

- ETL-style dumping of information in RDF,
- provisioning of Linked Data, RDFa or GRDDL interfaces,
- declarative querying (e.g. by means of SPARQL endpoints),
- Semantic Web Services or REST-style APIs, which return structured information adhering to the RDF data model.

The provisioning of semantically represented information in one of these forms helps distribute and syndicate structured content. In particular, the re-usability and repurposability of information is facilitated.

Consuming UbiSA: Information published as RDF is reusable by UbiSA. If an UbiSA accesses information from the Data Web to enrich their own information space, it is classified as a Consuming UbiSA. A Consuming UbiSA can obtain information from either one or multiple of the methods used for publishing structured information used by producing UbiSA or other Semantic Applications. In most cases, it will be sufficient for a consuming UbiSA to retrieve information via the HTTP protocol and parse one or multiple of the resultant RDF serializations formats, RDFa or SPARQL result formats.

4.3.5. Semantic richness

UbiSA can be further classified according to their use of rich knowledge representation (KR) formalisms:

- *Shallow KR UbiSA* comprise UbiSAs which e.g., primarily use taxonomies, simple hierarchies and relatively simple knowledge representation formalisms such as RDF and RDF-Schema.

- *Strong KR UbiSA* comprise UbiSAs which use higher level knowledge representation formalism such as different OWL variants, rules etc.

Already the declarative querying of knowledge bases, by means of SPARQL, currently adds a substantial performance overhead to UbiSA compared to relational database backed ubiquitous applications. That is without even considering implicit information, which must be revealed by reasoning. This is why we do not expect comprehensive description logic reasoning to be part of a standard UbiSA in the short to medium term. Instead, there might be some light inferencing, which can be performed (on demand or in certain intervals) by executing inference rules directly within triple stores (e.g., for resolving co-references, inverse relationships and computing transitive closures).

4.3.6. Semantic integration

This categorisation dimension measures how well an UbiSA is integrated within the Semantic Web. The integration can be measured on the schema and instance level. On the schema level, for example, the number of overall schema elements (i.e. RDF/OWL classes and properties) can be put in relation to the number of reused schema elements, which are either defined elsewhere or for which a `owl:sameAs` relation with an external element is defined. Similarly, the semantic integration can be measured on the instance level. Depending on the level of semantic integration, we call representatives integrated (respectively isolated):

- *Isolated UbiSA* are categorized by a limited reuse of shared identifiers, vocabularies and ontologies.
- *Integrated UbiSA* are categorized by a strong reuse of shared identifiers, vocabularies and ontologies.

4.3.7. User involvement

Another important characteristic of UbiSA is the degree of end-user involvement. End-users can be roughly classified into spontaneous contributors, advanced users and knowledge engineering experts. Subsequently, an UbiSA can be categorized according to the sizes and ratios in which these different end-user groups are participating in the creation of semantic knowledge representations within an UbiSA. Also, it can be distinguished which of these groups are restricted to contributions on the instance level and which participate in refining the knowledge schema. Other facets of the user involvement, which are not specific to UbiSA, are for example: the degree of closed user group, free for all, edit functionality for all information or just parts of the content.

5. Client Approaches

This chapter provides a general overview of existing client approaches for ubiquitous semantic applications. As defined by subsection 3.3.3, there are currently two existing client approaches: *Fat Client* and *Thin Client*. Additionally, there is the third approach that uses methodologies from both of the defined approaches *Hybrid Client*.

5.1. Thin client approach

In this section, we describe *OntoWiki Mobile*, which was developed using the thin client approach. This section is based on [Ermilov et al., 2011b].

The section is structured as follows: We introduce OntoWiki Mobile in subsection 5.1.1. We outline the general architecture of OntoWiki Mobile in subsection 5.1.2. We describe our replication and reconciliation strategy in subsection 5.1.3. The OntoWiki Mobile user interface and the implementation of browsing and authoring in restricted mobile environments is presented in subsection 5.1.4. A description of a use case for OntoWiki mobile in the domain of field expeditions in bio-diversity research is presented in subsection 5.1.5. Finally, we conclude in subsection 5.1.6.

5.1.1. Introduction

One of most important features of ubiquitous applications is their ability to provide *offline functionality* with local updates for later synchronization with a web server. The key problem here is the reconciliation, i. e. the problem of potentially *conflicting updates* from *disconnected clients*.

Another problem current ubiquitous application developers face is the plethora of ubiquitous application development platforms as well as the incompatibilities between them. *Android* (Google), *iOS* (Apple), *Blackberry OS* (RIM), *WebOS* (HP/Palm), *Symbian* (Nokia), *Firefox OS* (Mozilla) are popular and currently widely deployed platforms, with many more proprietary ones being available as well. As a consequence of this fragmentation, realizing a special purpose application, which works with many or all of these platforms is extremely time consuming and inefficient due to the large amount of duplicate work required.

The W3C addressed this problem, by enriching HTML in its 5th revision with access interfaces to local storage (beyond simple cookies) as well as a number of devices and sensors commonly found on ubiquitous devices (e. g. GPS, camera,

compass etc.). We argue, that in combination with semantic technologies these features can be used to realize a *general purpose* ubiquitous collaboration platform, which can support the long tail of ubiquitous special interest applications, for which the development of individual tools would not be (economically) feasible.

In this section, we present the *OntoWiki Mobile*[1] approach realizing a mobile semantic collaboration platform based on the OntoWiki framework [Heino et al., 2009]. It comprises specifically adopted user interfaces for browsing, faceted navigation as well as authoring of knowledge bases. It allows users to collect instance data and refine the structured knowledge bases on-the-go. OntoWiki Mobile is implemented as an *HTML5 web application*, thus being completely ubiquitous device platform independent. In order to allow offline use in cases with restricted network coverage (or in order to avoid roaming charges), it uses the novel HTML5 local storage feature for replicating parts of the knowledge base on the ubiquitous device. Hence, a crucial part of OntoWiki Mobile is the advanced conflict resolution for RDF stores. The approach is based on a combination of the EvoPat [Rieß et al., 2010] method for data evolution and ontology refactoring along with a versioning system inspired by distributed version control systems like Git.

There are already a number of ubiquitous semantic applications ranging from semantic backend services [Sonntag et al., 2007] for ubiquitous devices to applications covering very specific use cases (e. g. *DBpedia Mobile* [Becker and Bizer, 2009] or *mSpace Mobile* [Wilson et al., 2005a]). OntoWiki Mobile, however, is a generic application domain agnostic tool, which can be utilized in a wide range of usage scenarios ranging from instance acquisition to browsing of semantic data on the go. Typical OntoWiki Mobile usage scenarios are settings where users need to author and access semantically structured information on the go or in settings where users are away from regular power supply and restricted to light-weight equipment (e. g. scientific expeditions).

5.1.2. Architecture

OntoWiki was developed to address the need for a Web application for rapid and simple knowledge acquisition in a collaborative way. OntoWiki can be used for presenting, authoring and managing knowledge bases adhering to the RDF data model. In order to render its functionality, OntoWiki relies on several APIs that are also available to third-party developers. Usage of these programming interfaces enables the users to extend, customize and tailor OntoWiki in several ways. OntoWiki's architecture consists of three separate layers: persistence layer, application layer and user interface layer. These layers represent the standard MVC[2] architecture. The persistence layer consists of the Erfurt API which provides an interface to different RDF stores (e. g. Virtuoso, MySQL). Content in OntoWiki is rendered through the templates (user interface layer). The controller action

[1] A live demo is available at `http://m.ontowiki.net/`

[2] Model-View-Controller

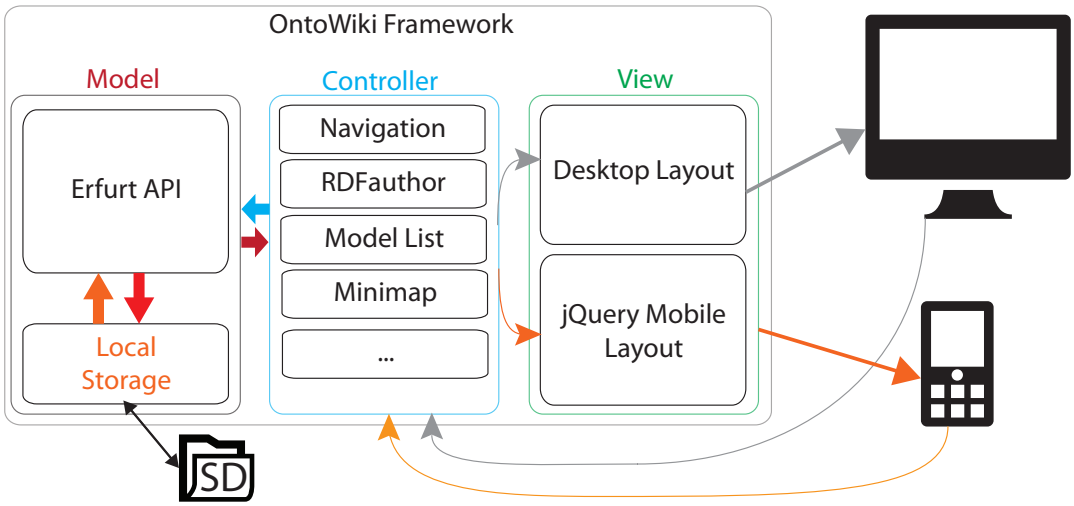


Figure 5.1.: OntoWiki Mobile architecture.

serving the request renders its output in a template. OntoWiki as a Web application is based on the Zend Framework[3] which lays out the basic architecture and is primarily responsible for request handling. Such architecture allows to easily extend the functionality of OntoWiki and change the layout based on context parameters of the user request. OntoWiki Mobile is based on the OntoWiki Framework. It utilizes all of the described OntoWiki Framework architecture and tailors it to better fit mobile usage scenarios by e. g. replacing with mobile-specific layout (see Figure 5.1).

The mobile user interface was built using HTML5 and the jQuery Mobile[4] framework which includes the core jQuery library in an improved version to ensure compatibility across all of the major mobile platforms. Built on a jQuery and jQuery UI foundation, it allowed us to create a unified user interface regardless of the actual platform the user's device runs on. The resulting source code presents a thin JavaScript layer, built with Progressive Enhancement principles so as to allow for a minimal footprint.

To access the device's hardware (e. g. camera, GPS sensor), OntoWiki Mobile uses the extended HTML5 API[5]. Geolocation API is used from JavaScript to get the user's current latitude and longitude. Local storage is a part of the HTML5 application caches and is a persistent data storage of key-value pair data in Web clients. It is used to store replicated parts of knowledge bases on the client-side. OntoWiki Mobile stores RDF data as JSON-encoded strings for offline usage and to increase page loading speed when online (e. g. in cases where the resource has

[3] http://framework.zend.com/
[4] http://jquerymobile.com/
[5] http://w3.org/TR/html5/offline.html#offline

not been changed and does not require reloading). Attached photograph are stored to local cache using HTML5 Canvas base64 encoding. Usage of local storage allows to export and import user-gathered data, for example to do backups (snapshots) of data to external SD card or to share data with other mobile devices via bluetooth.

Resource editing in OntoWiki is done using RDFauthor [Tramp et al., 2010c]. The system makes use of RDFa-annotations in web views in order to make RDF model data available on the client. Embedded statements are used to reconstruct the graph containing statements about the resource being edited. A set of editing widgets, tailored to specific editing tasks and equipped with end-user supporting functionalities (e. g. resource autocompletion from OntoWiki and Sindice) are selected based on the statements contained in the graph. In OntoWiki Mobile, RDFauthor has been adapted to better cope with mobile environments by adapting the user interface and introducing lazy script loading.

Data replication and conflict resolution is the most complex part of the OntoWiki Mobile. The process consists of three steps, handled by separate components (explained in more detail in Section 5.1.3):

- a client-side replication component utilizing HTML5 local storage,
- a server-side replication component and
- a server-side conflict resolver.

The conflict resolver uses additional mechanisms to simplify merging concurrent edits of the same resource. The first one is policy-based semi-automatic merging tool that utilizes the EvoPat engine – an OntoWiki extension for dealing with evolution of knowledge bases using patterns [Rieß et al., 2010]. Evolution patterns in EvoPat consists of variables, a SPARQL query template and a SPARQL/Update query with functional extensions. Results of the SPARQL query are bound to variables which in turn are used in SPARQL/Update queries to perform knowledge base transformations. OntoWiki Mobile uses specifically created patterns that can be applied by the user. The second mechanism provides a user interface for manual conflict resolution. This mechanism allows the user to select the specific statements from different versions to include in a merged version of a resource.

5.1.3. Replication

One critical requirement for OntoWiki Mobile was the ability to work without an internet connection. In cases where several users edit the same resources without synchronization in between, replication issues may occur. At least one of the users is likely to be working with an outdated version of a resource. When the user attempts to synchronize data with the main OntoWiki server, several steps are taken to minimize the need for human intervention. However, fully automatic conflict resolving is not possible in all cases.

Concepts The unit of editing and display in OntoWiki is a *resource*. Since OntoWiki Mobile needs to identify the same resource at different points in time, we define a resource r_t as a set of triples contained in some graph that share the same subject at a certain point in time, i. e. a description of r at timestamp t. When an editing operation is carried out, all the changed triples are saved/deleted at once for a given resource. Thus, a *diff* $d_{s,t}$, $s < t$ is the change applied to a resource description from timestamp s to timestamp t. It is defined as a quadruple

$$d_{s,t} \coloneqq (s, t, \mathsf{Add}, \mathsf{Del}) = (s, t, r_t \setminus r_s, r_s \setminus r_t).$$

That is, it contains a set of added and a set of removed statements that led from r_s to r_t. Two diffs $d_{s_1,t_1} = (s_1, t_1, \mathsf{Add}_1, \mathsf{Del}_1)$ and $d_{s_2,t_2} = (s_2, t_2, \mathsf{Add}_2, \mathsf{Del}_2)$ are said to be *in conflict* if both of the following conditions are met:

$$2 \cdot |\mathsf{Add}_1 \cup \mathsf{Add}_2| > |\mathsf{Add}_1| + |\mathsf{Add}_2| \tag{5.1}$$

$$\mathsf{Del}_1, \mathsf{Del}_2 \neq \emptyset \Rightarrow \mathsf{Del}_1 \cap \mathsf{Del}_2 \neq \emptyset \tag{5.2}$$

In other words, both diffs remove at least one identical and add at least one different triple. The empty diff $d_{s,t} = (s, t, \emptyset, \emptyset)$ does not conflict with any other diff. This definition gives necessary, but not sufficient, conditions for conflicting changesets, i. e. there are non-conflicting changesets that meet both conditions.

Let $d_{s_1,t_1} = (s_1, t_1, \mathsf{Add}_1, \mathsf{Del}_1)$ and $d_{s_2,t_2} = (s_2, t_2, \mathsf{Add}_2, \mathsf{Del}_2)$ be two diffs at timestamps t_1 and t_2 with $s_i < t_i$, $t_1 < s_2$. The *concatenation* operation $\circ : \mathsf{D} \times \mathsf{D} \longrightarrow \mathsf{D}$ (D denoting the set of all diffs) yields a new diff with the combined additions and deletions from d_{s_1,t_1} and d_{s_2,t_2}:

$$d_{s_1,t_1} \circ d_{s_2,t_2} = (s_1, t_2, \mathsf{Add}_1 \cup \mathsf{Add}_2, \mathsf{Del}_1 \cup \mathsf{Del}_2).$$

Consecutive diffs to the same resource r_t are combined to *changeset*s, that are exchanged between mobile devices (OntoWiki Mobile) and OntoWiki on synchonization.

Synchronization We are now in the position to specify what happens when users synchronize data with OntoWiki.

Given a re-established data connection and the user's consent, OntoWiki Mobile sends all changesets back to the server's synchronization component. Let c be a changeset on resource r with diffs $(d_{s_1,t_1}, d_{s_2,t_2}, \ldots, d_{s_k,t_k})$. OntoWiki Mobile concatenates the diffs contained in c into a single diff d_{s_1,t_k}. A server diff is then calculated as $d_{s,t}$ where s and t are the largest timestamps for changes on r smaller than s_1, t_k respectively. Using conditions (5.1) and (5.2) conflicting diffs are determined. In case of a conflict, all diffs $d_{s,t}$ in c are applied sequentially (w. r. t. t) until the conflict occurs. At this point, two branches of r are created having as last common version $r_{t_{k-1}}$ where t_k is the conflicting patch.

For merging branches, two different ways exist: manually (using the OntoWiki's merging UI) or semi-automatically (using EvoPat). EvoPat allows the application of policy-based merging patterns on conflicting branches. Patterns for the following merging policies are provided with OntoWiki Mobile:

- *User privilege-based* – changesets from users with higher priority have prevalence or
- *time privilege-based*, which can also be called "first-come, first-served" – the latest changes are considered least prioritized.

Additional policies (in the form of EvoPat evolution patterns) can be created by the user, if needed.

There are some situations that cannot be completely resolved without user intervention or creation of additional rules for EvoPat. For example, in cases where two users create a resource describing the same real world object by using different identifiers.

Example Consider two users *Alice* and *Bob* who work on the same resource r_{t_0} which has two statements, s_1 and s_2, as of timestamp t_0. The described scenario is depicted in Figure 5.2.

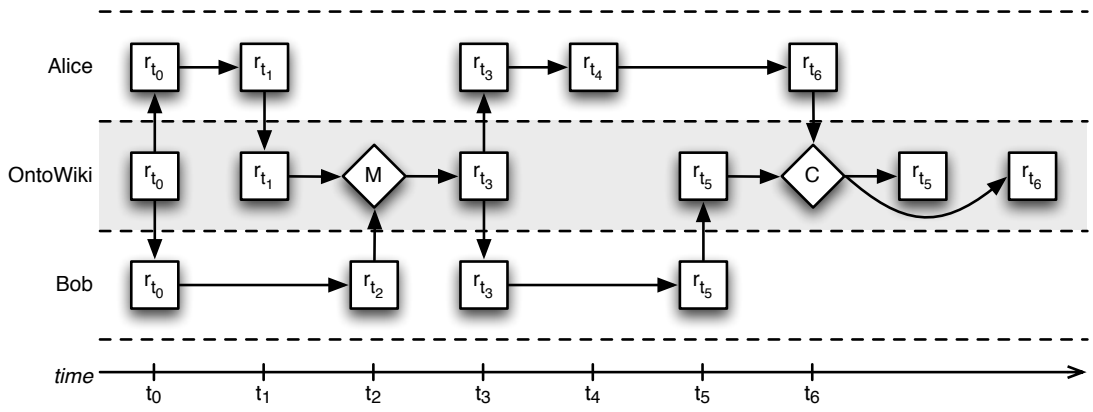


Figure 5.2.: Data replication example with merge (M) and conflict detection (C).

At t_1, Alice changes statement s_1 to s_3; this is actually reflected as deleting s_1 and adding s_3. In the same way, Bob removes s_2 and adds s_4 at t_2. When both synchronize with OntoWiki, their respective changesets contain only one patch each

$$c_{t_0,t_1,\text{Alice}} = (t_0, t_1, \{s_3\}, \{s_1\}) \quad \text{and} \quad c_{t_0,t_2,\text{Bob}} = (t_0, t_2, \{s_4\}, \{s_2\}).$$

Since $\{s_1\} \cap \{s_2\} = \emptyset$, condition (5.2) does not hold and we have non-conflicting changesets that can be merged into r_{t_3} at t_3. Alice then adds another statement s_5 at t_4 and later discovers that she entered duplicate information and decides to remove s_3 at t_5. Meanwhile, Bob also notices the error on s_3, removes it and adds s_6 at t_4. This time, when both synchronize their data with OntoWiki, we have patches

$$c_{t_3,t_6,\text{Alice}} = (t_3, t_6, \{s_5\}, \{s_3\}) \quad \text{and} \quad c_{t_3,t_5,\text{Bob}} = (t_3, t_5, \{s_6\}, \{s_3\}).$$

As can be easily verified, both conditions now hold and we deal with a conflicting changeset. OntoWiki Mobile thus creates two versions, r_{t_5} and r_{t_6}, resulting from applying $c_{t_3,t_5,\text{Bob}}$ and $c_{t_3,t_6,\text{Alice}}$ to r_{t_3}, respectively.

5.1.4. User Interface

The OntoWiki Mobile user interface supports currently three different usage patterns: standard browsing along the taxonomic structures (e. g. class hierarchies) found in the knowledge base, faceted browsing for filtering instances based on property values as well as authoring of new information on-the-go.

Standard Browsing. Figure 5.3 shows the OntoWiki Mobile standard navigation user interface in different browsing states.

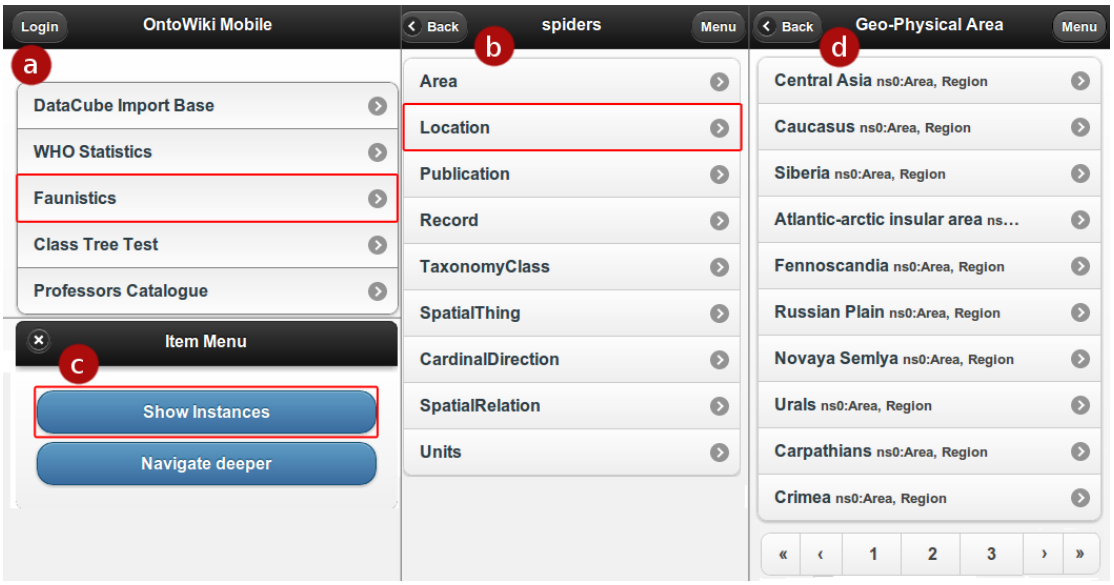


Figure 5.3.: OntoWiki Mobile standard browsing interface.

In accordance with popular touch-oriented mobile software platforms, the user interface was based on lists so as to simplify navigating through interlinked resources. The first screenshot (Figure 5.3a) shows the list of all knowledge bases. The login button in the top-left corner allows to log in as a registered user (e. g. to write to protected knowledge bases). After the user selects the knowledge base by tapping on it (tapped areas show as red squares), the top level class structure is displayed, as shown in second screenshot (Figure 5.3b). Once a particular class is chosen the user has to select (Figure 5.3c) whether he wants to see the list of instances for this class or navigate deeper in a class tree. Navigating through the class tree simply changes the classes list entries and refreshes the view. If the user chooses to view instances, a new list of instances from this class is presented (Figure 5.3d).

After selecting a particular instance all properties are grouped by predicates and rendered in a list.

Faceted browsing. Faceted browsing is a special way to navigate through instances in a specific knowledge base. It allows for simple and efficient filtering of the displayed instances list by applying available instance properties as filters. Faceted browsing can be used with any instance list in OntoWiki Mobile. As show in Figure 5.3d instance list view has a menu button in the upper-right corner.

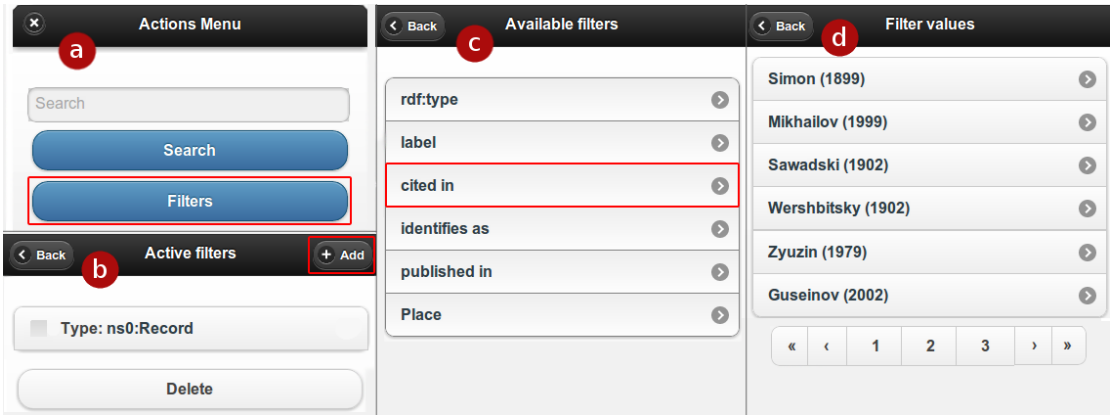


Figure 5.4.: OntoWiki Mobile faceted browsing interface.

Using this button the user can access the instance list menu (Figure 5.4a), where he can execute a simple string search in current knowledge base or use the "Filters" button to access the faceted browsing feature. As shown in Figure 5.4b, the active filters list view displays all currently applied filters. By checking filters and pressing the "Delete" button at the bottom of the screen, the user can remove filters he does not like to apply to the list. To add a new filter, the "Add" button in the upper-right corner of the screen can be used, which will display the list of all available filters (Figure 5.4c). Selecting one of the displayed filters will open the list of values for it (Figure 5.4d). Selecting values from the list shown will apply the new filter value restriction on the previously displayed instances list.

Authoring. Data authoring in OntoWiki Mobile is done using the RDFauthor [Tramp et al., 2010c] – a JavaScript-based system for RDF content authoring. As mentioned earlier, instance properties are grouped by predicates and rendered as lists (Figure 5.5a). The rendering of properties and their values is based on the data type and ontology structure (see the display of a map for attached geo-coordinates). Figure 5.5b shows how an instance can be created or edited using forms, which are automatically created by RDFauthor based on the underlying ontology structure in the knowledge base (note auto-detected geographical coordinates for current location). Figure 5.5c shows an example of the interaction of OntoWiki mobile

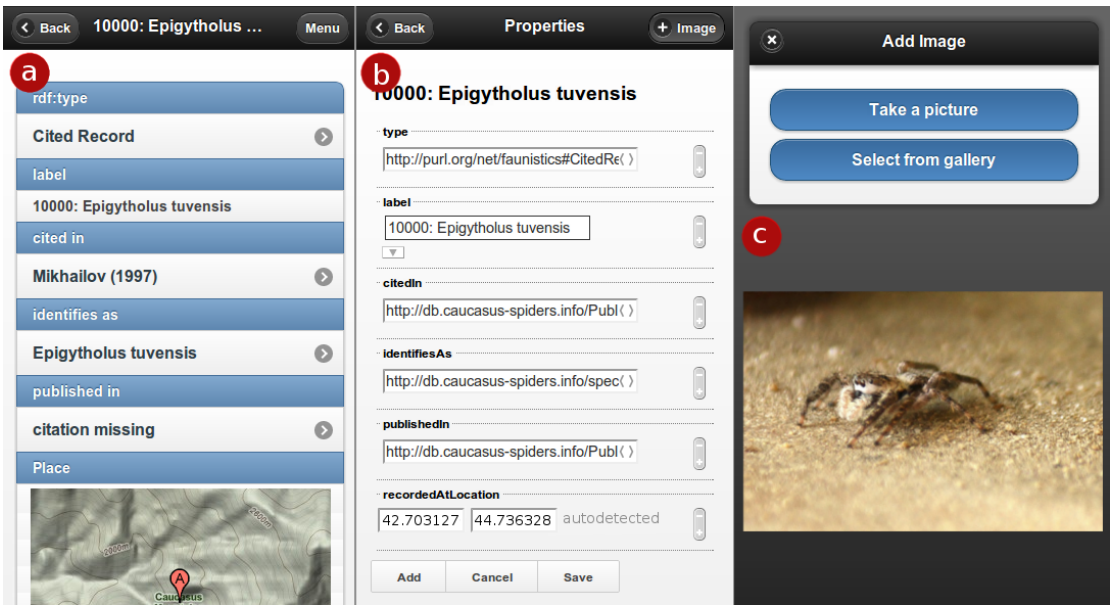


Figure 5.5.: OntoWiki Mobile authoring interface.

with the sensors of the mobile device in terms of accessing the integrated camera for adding a picture to an ontology instance.

5.1.5. Use Case and Evaluation

The development of OntoWiki Mobile was triggered by users aiming to gather data in field conditions. To simplify the data collection, we created a mobile interface that allows users to enter data instances on mobile devices, such as mobile phones and tablets. In particular, there is a community of scientists who collect data about spiders in the Caucasus region [Otto and Dietzold, 2007] in a web portal[6]. The project consists of two major software parts:

- The portal backend, which is based on the semantic data wiki OntoWiki. Each arachnologist can login to this backend and use it for data entry, management and queries. The backend itself is a standard OntoWiki installation with some custom and some common vocabularies imported.
- The portal front-end, which itself is an extension of OntoWiki, generates a more visitor-friendly representation of the databases resources. The main focus for these visitors are species checklists, which give an overview of verified species of a given region.

To calculate these species checklists for a specific area, data from original finding spots (e. g. "I've found the species Pholcus phalangioides near Khashuri in Georgia

[6] `http://db.caucasus-spiders.info`

in a cave.") as well as data from the literature (e. g. "Mkheidze (1964) published he found this species in Lentekhi as well.") is collected by the users. While the literature research is done at home using the desktop browser-based version of OntoWiki, the field data is gathered according to the following workflow:

1. A research team travels to the area and sets up traps at specific locations or specifically catches interesting individuals.
2. The finding spots are documented and the individual animals are associated with these finding spots (e. g. by signing a conservation container with the location ID).
3. The individuals are carried to the laboratory where they have to be identified. This is a challenging task and often individuals are sent to specialists for a specific genus or family of spiders.
4. Finally, the individual is identified as a certain species. This event either increases the finding spot counter for this species in a certain area or adds another species entry to the species checklist for this area. In the latter case, this (re-)discovery of the species in a certain area can be published.

In this workflow, only the second step is relevant for the evaluation of our work since step 3 and 4 are done with the standard wiki and the portal front-end. The goal of step 2 is to describe the finding spot where a specific animal was found in order to proof assumption about the habitat and living of a specific species. These finding spots are classified according to a nature-phenomenological system (e. g. a cave, field, ...) and are allocated to a nearby populated place. Populated places are ordered and associated to a political and administrative system (e. g. counties, administrative regions, country). The database currently consists of 1060 populated places and locations as well as 191 areas. A complete finding spot documentation is then entered in the following steps (see N3 example in Listing 5.1):

- Instantiate a specific type of location (e. g. a cave, example line 8).
- Add geo-coordinates to this resource (taken automatically from the GPS subsystem, example line 13).
- Associate pictures with this resource (taken from the camera subsystem, example line 12).
- Associate a nearby populated place or an area by searching the local store for an existing one or by creating a new resource (example line 14).
- Add any other information either specific for the location type (e. g. height for glaciers), specific for the researcher (e. g. comments and tags) or specific for the research journey (e. g. internal location ID for the conservation containers, refer example line 9–11).

Listing 5.1: Example finding spot documentation in N3.

```
@prefix db: <http://db.caucasus-spiders.info/> .
@prefix faun: <http://purl.org/net/faunistics#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .

<http://db.caucasus-spiders.info/Place/555> a faun:Cave ;
    rdfs:label "Lower Mzymta Cave (Sochi)";
    rdfs:comment "container 5";
    dc:creator "Stefan Otto"; dc:date "2010-07-21";
    foaf:depiction db:FotoXXX;
    geo:long "39.99933"; geo:lat "43.57695" ;
    faun:nearby db:Place19; faun:within db:Area439.
```

The result of a finding spot location documentation is shown in Listing 5.1.

The data in this listing correspond to the screenshots shown in Figure 5.6.

After using the prototype for a few weeks on an Android developer phone, the following pro and con statements were obtained from the OntoWiki Mobile evaluation participants during interviews:

- Even without doing extensive data entry, the feature of associating images to existing resources was liked very much.
- There was a constant fear for data loss by breaking, misusing or loosing the mobile device. The added feature to export and import file backups from and to the application eased this. If there are more than one mobile device in the field, these backups can be additionally used to approve and inspect the data with a second pair of eyes.
- Obtaining GPS data from the mobile phone is nice but slightly inaccurate since the internal GPS systems of mobile phones are not as good as dedicated devices e. g. for hiking. Auto-completion of these values is a nice feature but users need to be able to correct them or, even better, receive them from another device and overwrite the existing values.
- The user experience strongly depends on the given CPU of the mobile device. Users running a mobile device with 500Mhz (HTC Hero) complained about the slowly responding user interface. In comparison to that, users with a 1000Mhz device (Samsung Galaxy S) reported a fast and reliable interface. In addition to the CPU, the version of the hosting Android operating system and esp. the used browser version strongly affects the user experience since

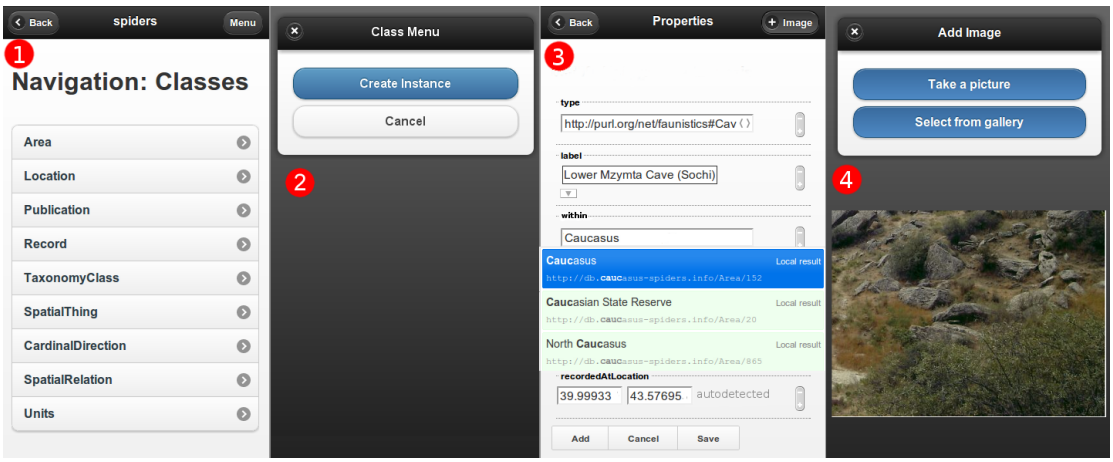


Figure 5.6.: Screenshots illustrating the workflow for creating a new finding spot according to the listing in Figure 5.1. From left to right: (1.) Searching or browsing for the class which needs to be instantiated. (2.) Initialization of a new resource from this class; all properties which are offered, are used in other instances of this class; GPS data is automatically requested and pre-filled by the phone. (3.) Entering literal data as well as linking to other resources. (4.) Assignment of existing images from the phone's image library.

newer Android versions also ship a new browser with a faster JavaScript execution engine.

5.1.6. Conclusions

As the penetration of mobile devices that are able to access and interact with the Web can be expected to dramatically increase within the next years, the Semantic Web can ultimately only be successful if the use of semantic technologies on mobile devices is fully supported. With OntoWiki Mobile, we tackled one particular but crucial aspect – the provisioning of a comprehensive knowledge management tool for mobile use. It employs the new HTML5 application cache functionality to support offline work and has advanced conflict resolution features built-in. OntoWiki Mobile demonstrates that a comprehensive semantic collaboration platform is possible to implement for mobile devices with minimal requirements based on recent Web standards (in particular HTML5). Although OntoWiki Mobile was already used in a specific use-case by a number of non-IT, domain expert users, more effort is required to evaluate the tool in a wider range of application scenarios. Due to its general purpose architecture, OntoWiki Mobile is particularly suited to support the long tail of domain-specific mobile applications, for which the development of individual tools would not be (economically) feasible.

5.2. Hybrid client approach

In this section, we describe *Mobile DSSN Client* that was developed using the hybrid client approach. This section is based on [Ermilov et al., 2012].

The section is structured as follows: We introduce the idea of accessing the Social Semantic Web using hybrid client in subsection 5.2.1. We outline the general architecture of Mobile DSSN Client in subsection 5.2.2. We describe our implementation of Mobile DSSN Client in subsection 5.2.3. The evaluation of our implementation is explained in subsection 5.2.4. Finally, we conclude in subsection 5.2.5.

5.2.1. Introduction

For many people smartphones already replace the computer as their window to the Internet, to the Web as well as to social networks. Online social networking meanwhile became one of the most popular service on the Web. Hence, the management and presentation of information about contacts, social relationships and associated information is one of the main requirements and features of today's smartphones.

The problem is currently solved solely for *centralized* proprietary platforms (such as Google Mail, Contacts & Calendar) as well as data-silo-like social networks (e.g. Facebook). As a result of this data centralization, users' data is taken out of their hands, they have to accept the predetermined privacy and data security regulations; users are dependent of the infrastructure of a single provider, they experience a lock-in effect, since long-term collected profile and relationship information cannot be easily transferred.

Especially Facebook with it's 600M+ million users creates a web inside the Web. Drawing the metaphor of islands, Facebook is becoming more like a continent. Users are locked into a certain platform and hardly have a chance to get out again if they want to keep their connections. Once published, users also lose control about the data they own, since it is stored on a single companies servers. Migrating to another platform is not possible or at least very difficult. Interoperability between platforms is rare and limited to proprietary APIs. In order to keep data up-to-date on multiple platforms users have to update their information on each of their used SNSs and thus information might diverge. Since there are only a few large players the Web partly loses it's distributed nature.

Increasingly, many people argue that social networks should be evolving. That means, social networks should allow users to control what to enter and to keep a control over their own data. Also, the users should be able to host the data on an infrastructure, which is under their direct control, the same way as they host their own website [Berners-Lee, 2010]. A possibility to give the control over their data back to the users is the realization of a truly *distributed* social network. Initial approaches for realizing a distributed social network appeared with *GNU social* and more recently *Diaspora*. However, we argue that a distributed social network

should be also based on semantic resource descriptions and de-referenceability so as to ensure versatility, reusability and openness in order to accommodate unforeseen usage scenarios.

Within the Semantic Web initiative already a number of standards and best-practices for social, Semantic Web applications such as *FOAF*, *WebID* and *Semantic Pingback* emerged. However, there is no comprehensive strategy, how these technologies can (a) be combined in order to weave a truly open and distributed social network on the Web and (b) be used efficiently in a ubiquitous environment. Also, the use of a distributed, social semantic network should be as *simple* as the use of the currently widely used centralized social networks (if not even simpler). In this section, we present the general strategy for weaving a distributed social semantic network based on the above mentioned standards and best-practices. In order to foster its adoption we developed an implementation for the Android platform, which seamlessly integrates into the commonly used interfaces for contact and profile management on ubiquitous devices.

After briefly reviewing some use cases and requirements for a ubiquitous, semantic social network application (in Section 5.2.3), we make in particular the following contributions:

- We outline a strategy to combine current bits and pieces of the Semantic Web technology realm in order to realize a distributed, semantic social network (Subsection 5.2.2),
- We develop an architecture for making ubiquitous devices endpoints for the Social Semantic Web (Subsection 5.2.2),
- A comprehensive implementation of the architecture is performed for the Android platform (Subsection 5.2.3),
- We perform an evaluation of our implementation according to the W3C Social Web Acid Test and interoperability tests with OntoWiki and Dydra (Subsection 5.2.4),

Furthermore, the section concludes with a discussion in 5.2.5.

5.2.2. Distributed Semantic Social Networking

In this section, we describe an architecture for distributed semantic social networking which guided the requirements definition as well as the implementation of the mobile DSSN client. After introducing a few design principles on which the architecture is based, we present its different layers, i.e. the data, protocol, service and application layers. The overall architecture is depicted in Figure 5.7. Our architecture is based on the following three design principles.

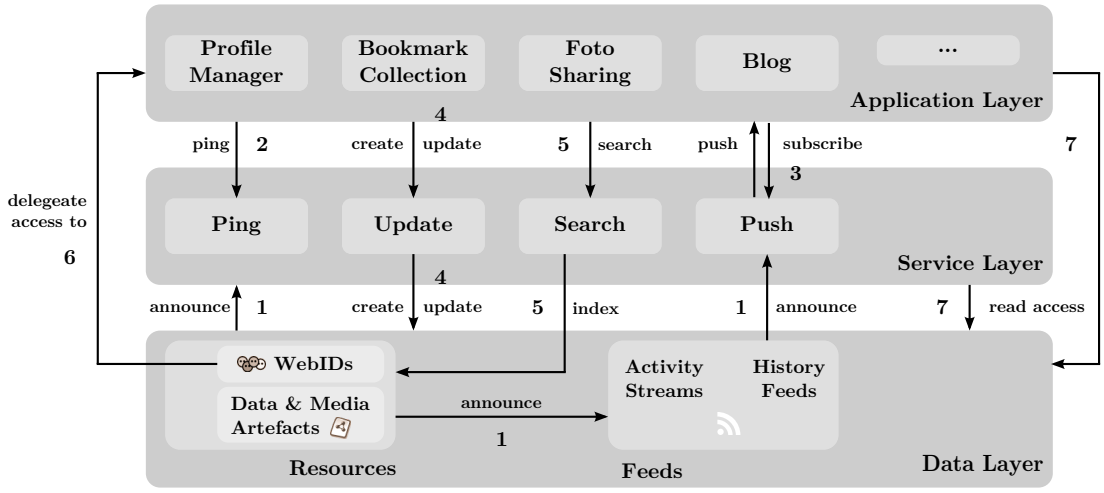


Figure 5.7.: Architecture of a Distributed Semantic Social Network (without protocol layer): (1) Resources announce services and feeds, feeds announce services – in particular a push service. (2) Applications initiate ping requests to spin the Linked Data network. (3) Applications subscribe to feeds on push services and receive instant notifications on updates. (4) Update services are able to modify resources and feeds (e.g. on request of an application). (5) Personal and global search services index social network resources and are used by applications. (6) Access to resources and services can be delegated to applications by a WebID, i.e. the application can act in the name of the WebID owner. (7) The majority of all access operations is executed through standard web requests.

Linked Data. The main protocol for data publishing, retrieval and integration is based on the Linked Data principles [Berners-Lee, 2011]. All of the information contained and accessible in the Distributed Semantic Social Network is represented according to the RDF paradigm, made de-referencable and interlinked with other resources. This principle facilitates heterogeneity and enables the distribution of data and services on the Web.

Service Decoupling. A second fundamental design principle is the decoupling of user data from services as well as applications [Krohn et al., 2007]. It ensures that users of the network are able to choose between different services and applications. In addition, this principle helps users of the social network to distinguish between their own data on one side, which they share with and license to other people and services and foreign data on the other side, which they create by using these services and which they do not own. This decoupling principle can only be achieved by using different methods to allow an automatically discovery of connected and

relevant services. In our architecture, we heavily depend on RDF properties which relate resources to services, as well as HTTP header extensions which represent the same link but allow faster fetching of the relevant information.

Protocol Minimalism. The main task for social networking protocols is to communicate RDF triples between nodes in the network, not to enforce a specific work flow nor an exact interpretation of the data. This constraint ensures the extensibility of the data model and keeps the overall architecture clean.

Based on these design principles, we reviewed the current technology stack for the Social Semantic Web and developed a coherent architecture which uses existing state-of-the-art technologies and allows for social networking activities comparable to centralized social networks (cf. Section 5.2.4 for an evaluation of this claim).

We divided our architecture into four layers, namely *data*, *protocol*, *service* and *application* layer. The **data layer** (depicted in the lower third of Figure 5.7) is formed by a network of interlinked social network data objects. We distinguish between two generic types of data objects: *resources* and *feeds*. While feeds are used to represent temporally ordered information in a machine-readable way, resources represent static artefacts. Feeds are widely used on the Web and play a crucial role in allowing real-time communication between different services. In the context of the DSSN architecture, two types of feeds are worth considering: activity feeds and history feeds.

History feeds represent an ordered list of change sets which allow to re-create the current and former states of the resource as well as to synchronize resources on different services. Currently we use PubSubHubbub[7] as the Social Network wide publish/subscribe protocol, since it is widely supported and allows for custom payloads if used in combination with Atom feeds. Activity feeds represent an ordered timeline of social network activities. They can be used to visualize activity streams which are either centered around an activity object (e.g. activities on an image) or an activity subject (e.g. activities of a specific user).

Besides feeds which are used to communicate certain types of events, linked resources spin the a network of Social Web artifacts. We distinguish between three main categories of resources: WebIDs for persons as well as applications, data artefacts and media artefacts.

WebID [Sporny et al., 2010][8] is a best practice recently conceived in order to simplify the creation of a digital ID for end users. Since its focus lies on simplicity, the requirements for a WebID profile are minimal. In essence, a WebID profile is a de-referenceable RDF document (possibly even an RDFa-enriched HTML page) describing its owner[9]. That is, a WebID profile contains RDF triples which have the IRI identifying the owner as subject. The description of the owner can be performed

[7] `http://code.google.com/p/pubsubhubbub/`

[8] The latest spec is available at `http://webid.info/spec/`.

[9] The usage of an IRI with a fragment identifier allows the indirect identification of an owner by reference to the (FOAF) profile document.

in any (mix of) suitable vocabulary (-ies), but FOAF [Brickley and Miller, 2004] has emerged as the 'industry standard' for that purpose.

Data Artefacts are resources on the Web which are published according to the Linked Data principles. Data artefacts includes posts, comments, taggings, activities and other Social Web artefacts which have been created by services and applications on the Web. Most of them are described using specific vocabularies such as SIOC [Breslin et al., 2006], Common Tag[10] or Activity Streams in RDF [Minno and Palmisano, 2010].

Media Artefacts are also created by services and applications but consist of two parts – a binary data part which needs to be decoded with a specific codec, and a meta-data part which describes this artefact. Usually, such artefacts are audio, video and image files, but office document types are also frequently used on the Social Web. Media artefacts can be easily integrated into the DSSN by using the Semantic Pingback mechanism, which is described later in this section.

The **protocol layer** is kept very simple and consists of the WebID identity protocol and two networking protocols which provide support for two completely different communication schemes, namely resource linking (Semantic Pingback) and push notification (PubSubHubbub). The basic idea of the **WebID protocol** [Sporny et al., 2010] (formerly known as a best practice [Story et al., 2009b]) is to connect an *SSL client certificate* with a WebID profile in a secure manner and thus allowing owners of a WebID to authenticate against 3rd-party websites with support for the WebID protocol. The WebID (i.e. a de-referencable URI) is, therefore, embedded into an *X.509 certificate* by using the Subject Alternative Name (SAN) extension. The document, which is retrieved through the URI, contains the corresponding public key. Given that information, a relying party can assert that the accessing user owns a certain WebID. Furthermore, the WebID protocol provides access control functionality for social networks shaped by WebIDs in order to regulate access to certain information resources for different groups of contacts (e.g. as presented with *dgFOAF* [Schwagereit et al., 2010]).

The purpose of **Semantic Pingback** [Tramp et al., 2010a] in the context of DSSN architecture is twofold: (a) It is used to facilitate the first contact between two WebIDs and establish a new connection (Friending). (b) It is used to ping the owner of different social network artefacts if there are activities related to these artefacts (e.g. commenting on a blog post, tagging an image, sharing a website from the owner). The Semantic Pingback approach is based on an extension of the well-known Pingback technology [Langridge and Hickson, 2002], which is one of the technological cornerstones of the overwhelming success of the blogosphere in the Social Web. The Semantic Pingback mechanism enables bi-directional links between WebIDs, RDF resources as well as weblogs and websites in general. It facilitates contact/author/user notifications in case a link has been newly established. It also allows to publish backlinks automatically from the original WebID profile (or other content, e.g. status message) to comments or references of the WebID (or

[10] `http://commontag.org/Specification`

other content) elsewhere on the Web, thus *facilitating timeliness and coherence* of the Social Web. As a result, the distributed network of WebID profiles, RDF resources and social websites can be much tighter and timelier interlinked by using the Semantic Pingback mechanism than conventional websites, thus rendering a network effect, which is one of the major success factors of the Social Web. Semantic Pingback is completely downwards compatible with the conventional Pingback implementations, thus allowing the seamless connection and interlinking of resources on the Social Web with resources on the DSSN. As requested by our third design paradigm (protocol minimalism), Semantic Pingback is a generic data networking protocol which allows to spin relations between any two Social Web resources. In the context of the DSSN Architecture, Semantic Pingback is used in particular for friending, commenting and tagging.

PubSubHubbub is a web-hook-based publish/subscribe protocol, as an extension to Atom and RSS, which allows for near instance distribution of feed entries from one publisher to many subscribers. Since feed entries are not described as RDF resources, PubSubHubbub is not the best solution as a transport protocol for a DSSN from our perspective. However, PubSubHubbub with atom feeds is widely in use and has good support in the web developer community which is why we decided to use it in our architecture. Similar to Semantic Pingback, it is agnostic to its payload and can be used for all publish/subscribe communication connections. In the DSSN architecture, two specific feeds are important and linked to a WebID allowing subscriptions to them: *activity feeds* which are used for activity distribution[11] and *change set feeds* which are used for resource synchronization.

Applications which are part of the **service layer** (depicted in the middle part of Figure 5.7 provide crucial infrastructure as part of the architecture (in contrast to applications which are build on top of the service layer such as the mobile client). WebIDs can be equipped with different services in order to allow manipulation and other actions on the user's data by other applications. As depicted in Figure 5.7, we have defined four essential services for the distributed semantic Social Network architecture.

The **ping service** provides an endpoint for any incoming pingback request for the resources of a user. First and foremost, it is used with the WebID for friending but also for comment notification and discussions. One application instance can provide its services for multiple resources. In a minimal setup, a ping service provides only a notification service via email. In a more complex setup, the ping service has access to the update service of a user (via access delegation) and can do more than just notification. As we described in [Tramp et al., 2010a], pingback services are announced in conjunction with resources using defined object property or an HTTP header field.

The **push service** is a PubSubHubbub hub and is used for activity distribu-

[11] Activity Distribution is a fundamental communication channel for any social network. A personal activity feed publishes the stream of all activities on social network resources (artefacts and WebIDs) with a specific user as the actor.

tion and resource synchronization. To equip social network resources with its corresponding activity and change set feeds, we have defined two OWL object properties which are sub-properties of the more generic `sioc:feed` relation from the SIOC vocabulary [Breslin et al., 2006]: `dssn:activityFeed` and `dssn:syncFeed`. In addition to these RDF properties, DSSN agents should pay attention to the corresponding HTTP header fields `X-ActivityFeed` and `X-SyncFeed`, which are alternative representations of the OWL object properties to allow the simple integration of media artefacts.

Search and index services are used in two different contexts in the Social Network architecture: (1) They are used to search for public web resources, which are not yet part of a user's social network. These search services are well-known semantic search engines as Swoogle [Ding et al., 2004] or Sindice [Tummarello et al., 2007]. (2) They are used to search and index private data as well as to cache resources for faster access. A private search service is used for all users and application queries from applications which act on behalf of the user. The underlying resource index of a private search service is used as a callback for all push notifications from feeds to which the user has subscribed. In our architecture we assume that search services accept SPARQL queries. However, this assumption is not true for all public Semantic Web search engines at the moment.

Finally, an **update service** provides an interface to modify and create user resources by means of SPARQL update queries. In the same way as private search services, update services are secured by means of the WebID protocol and accept requests only by the user itself and by agents in access delegation mode[12].

5.2.3. A Mobile DSSN Client

In this section, we outline how the Mobile DSSN Client was built[13]. First, we describe how it fits into the DSSN architecture in section 5.2.3. In section 5.2.3 we define requirements in order to make the client part of the DSSN and make it compatible with the widest variety of devices possible. Then we describe the platform independent implementation of the client in section 5.2.3 and discuss platform specific aspects in section 5.2.3.

Requirements and Integration into the Architecture

The mobile client described in this section is part of the application layer of the DSSN architecture. The client should able to fetch and display Linked Data resources as well as feeds, send pingback requests and use SPARQL to search and update resources. The only protocol not usable by the mobile client is the

[12] We defined `dssn:updateService` as a relation between a WebID and an update service

[13] The Mobile DSSN Client project page is available at `http://aksw.org/Projects/MobileDSSN`. It is an open source software and its source code can be found at `https://github.com/AKSW/MobileDSSN`. It can be also directly tested by accessing the following URL on a mobile device: `http://m.ontowiki.net/dssn/`

publish/subscribe push of PubSubHubbub since a push depends on a static callback URL which is not easy to provide in mobile scenarios.

Independent from the technical and integration requirements below, the mobile application has to comply to these functional requirements:

WebID profiles of the user as well as any other person should be displayable with the application. The users WebID should be writeable, if a connected SPARQL service allows SPARQL Update. The user should be able to traverse the social network by following outgoing `foaf:knows` relations and she should be able to make a new friend connection to a shown WebID. In order to find and display any WebID on the Web easily, a suitable semantic search engine should be integrated.

Activity streams in form of feeds should be accessible in a timeline interface, both for WebIDs and for other artefacts (e.g. images). In addition to consuming activities, a user should be able to react and act in the Social Network by means of creating new activities in a stream. The client should not only be able to create new activities but also to notify and **ping** all relevant resources which have a relation with this activity.

WebID profiles of a user's contacts should not only be visible in the network application but should be synchronized with the smartphone's contact database in order to integrate these data into the system for use by other applications.

Platform independent DSSN client

HTML5 and JavaScript were picked as main development languages to meet the requirements defined before. Creating the client as a web application allows to run the application on any modern mobile device. Supporting graceful degradation in the design of the web application allows the application to work correctly but with limitations on most devices released in the past few years. The usage of Model-View-Controller[14] (MVC) architectural pattern facilitates functionality extensions as well as the adaptation of the user interface behaviour and the integration for different platforms. The HTML5 API grants access to the device's hardware (e. g. camera, GPS) and persistent storage. Hardware access is used in the DSSN client for posting status updates with attached geographic coordinates and user's camera shots.

HTML5's local storage[15] functionality was used for persistent data storage. Local storage is a part of the HTML5 application caches and is a persistent data storage of key-value pair data in Web client applications. It is used to replicate parts of FOAF profiles at the client-side. The client stores RDF data as JSON-encoded strings for offline usage and to increase page loading speed while online (in cases where the resource has not been changed and does not require reloading). Usage of local storage allows to export and import user-gathered data, for example, to

[14] `http://en.wikipedia.org/wiki/Model\OT1\textendashview\OT1\textendashcontroller`

[15] `http://www.w3.org/TR/offline-webapps/#offline`

do backups (i.e. snapshots) of data to an external SD card or to share data with other mobile devices via Bluetooth.

Client architecture. As we mentioned before, the Mobile DSSN Client was built according to the MVC architectural pattern. Since JavaScript is mostly used as event-driven programming language, we decided to use the popular jQuery library[16] for simplifying the handling events and document manipulations. jQuery is a fast and concise JavaScript library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. Another advantage of jQuery is its extensibility. There is a large number of extensions and libraries based upon jQuery for almost any purpose.

To simplify the implementation and handling of the model part in our Javascript MVC application, we selected the Backbone.js[17] library. Backbone.js simplifies the structure of complex JavaScript applications by providing a data model abstraction with key-value binding and storage mapping as well as custom events, collections with a rich set of enumerable functions, and views with declarative event handling. Though Backbone.js supports two parts of the MVC architecture – model and view, we have used only the model part, since the view is handled better by jQuery Mobile (cf. section 5.2.3). The standard Backbone.js Model was extended to utilize local storage and caching.

Since the Mobile DSSN Client will handle FOAF profiles directly we have used *rdfQuery*[18] to simplify this process. rdfQuery is an easy-to-use JavaScript library for RDF-related processing. It can be used to parse RDFa embedded within a page, query over the contained facts and perform some simple reasoning to infer some implicit information. Together with a server-side triplestore, rdfQuery can be used to easily create authoring interfaces for the semantic web. rdfQuery comes in three components:

- *Core rdfQuery* – allows to create simple client-side triplestores and query them with JavaScript
- *rdfQuery with RDFa* – supports parsing of RDFa and adding RDFa to web pages.
- *rdfQuery with rules* – supports reasoning within triplestores using rules.

Since we do not use any client-side reasoning or RDFa parsing in the Mobile DSSN Client, only the Core rdfQuery was included into project.

Since user feeds are presented in the Atom syndication format, the jFeed library[19] was used for parsing feeds. Since our Mobile DSSN Client is a HTML5 Web

[16] `http://jquery.com/`
[17] `http://documentcloud.github.com/backbone/`
[18] `http://code.google.com/p/rdfquery/`
[19] jFeed is a lightweight JavaScript RSS/ATOM feed parser based on jQuery: `http://plugins.jquery.com/project/jFeed`

application some security restrictions apply. For example, the client is only allowed to perform AJAX requests to the same server where it was loaded from. In order to make the Mobile DSSN Client more flexible, functionality to proxy AJAX requests was added. If the server hosting the FOAF profile or a SPARQL endpoint does not have the client in its access white list for cross-side scripting and the user does not have the ability to change this configuration, the proxy needs to be used. The AJAX proxy was implemented in PHP and can be deployed on same server where the Mobile DSSN Client web application is hosted.

User Interface. The user interface was built using HTML5 and the jQuery Mobile[20] framework to ensure compatibility across all of the major mobile platforms. Built on a jQuery and jQuery UI[21] foundation, it allowed to create a unified user interface regardless of the actual platform the user's device runs on. The resulting source code presents a thin JavaScript layer, built with Progressive Enhancement principles so as to allow for a minimal footprint.

The Mobile DSSN Client user interface currently has two different usage patterns: profile and stream browsing and network traversing; adding friends, user's profile data editing and posting to user's feed.

Profile and Stream browsing. Figure 5.8 shows the Mobile DSSN Client user interface in different browsing states.

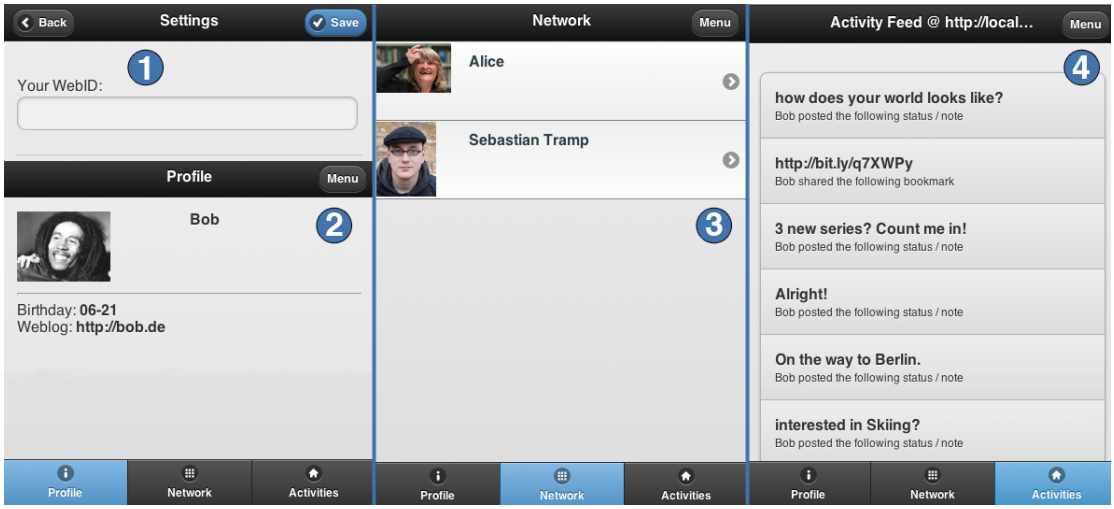


Figure 5.8.: Mobile DSSN Client standard browsing interface.

In accordance with popular touch-oriented mobile software platforms, the user interface was based on lists so as to simplify navigating through interlinked resources.

[20] http://jquerymobile.com/
[21] http://jqueryui.com/

The first screenshot (Figure 5.8.1) shows the settings screen invoked upon first run of the application. This screen allows the user to input his WebID URI. All the required data is gathered from the user's WebID after submission. The second screenshot shows how the user's profile is displayed. Currently, the profile screen is limited to several fixed fields, but with a bit of JavaScript knowledge a user can easily adjust it to his likings. We plan to provide simple user interface for this in future. The third screenshot shows the user's network as a simple list with names and depictions. Selecting any entry in this list will navigate to selected profile. The last screenshot (Figure 5.8.4) shows the user's activities stream which is also represented as a simple list. Bottom bar with screen selection is common to all of those views and aims to make the Mobile DSSN Client feel native even when using it from web browser. The menu button on top of the screen invokes a menu for data manipulation (e.g. profile editing, activity addition) that is described below.

Profile traversing can be accomplished in several ways:

- Simply selecting an interesting profile in network tab,
- Opening a WebID by its URI from the menu,
- Searching for a WebID using the Sindice search engine.

Selecting another WebID will open the same views as shown above for a user's own profile. Two additional buttons will appear when browsing through WebIDs: The "Home" button will be added to the top bar to navigate back to the user's profile, an "Add to network" entry will be added to the menu to add current WebID to the user's network.

Profile and Stream editing. Figure 5.9 shows editing interfaces of the Mobile DSSN Client.

The first screenshot (Figure 5.9.1) shows the profile editing interface. By using predicates names as labels and objects as text in inputs, this interface allows editing any field available in the user's profile. The Mobile DSSN Client tracks changes of every input and saves only the ones that actually have been changed. After data is saved to the model, the synchronization algorithm immediately applies changes to local storage and waits for an available data connection to send them to the DSSN node where the user's profile is hosted. At the moment the only supported way of updating a profile on the server is via SPARQL update queries.

Screenshot 2 shows the management of the user's network. After enabling this mode a simple selection of a list entry representing a relation will remove this relation from the user's profile. Addition of a new relation is shown in Screenshot 3. It is accomplished by entering the desired WebID URI. When the user browses through other WebIDs and wants to add a relation with one, he also reaches that screen, but with pre-filled input. It is also possible to use the Sindice search engine to find and add people. A Sindice search result is shown in Screenshot 4. Selection of a search result item will result in loading this item as WebID in browsing

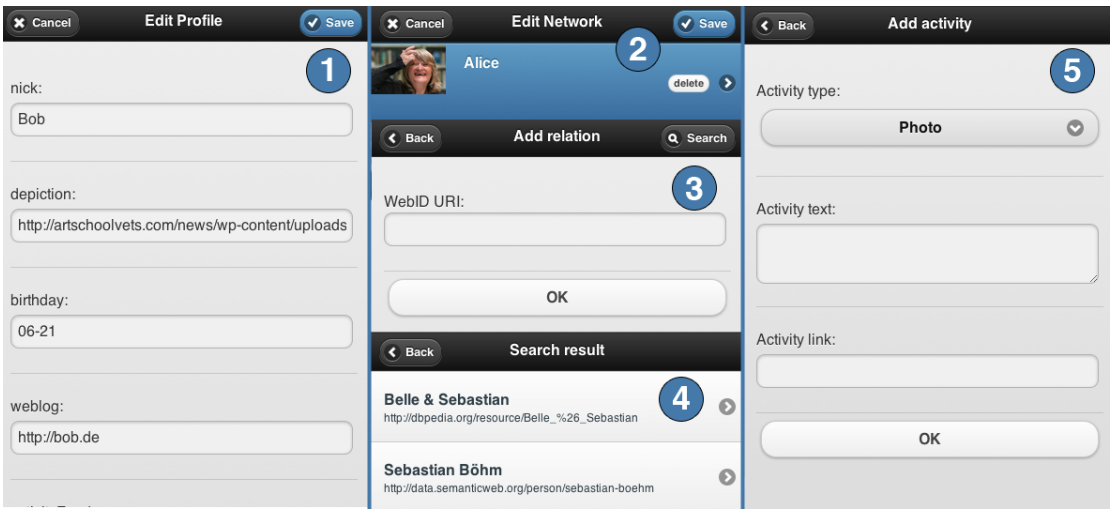


Figure 5.9.: Mobile DSSN Client editing interfaces.

interface. The last screenshot (Figure 5.9.5) shows the interface for the creation of new activities. It consist of three fields: activity type, activity text and URI for activity. It is also possible to post text-only activities without any URI included. Also activity creation is, at the moment, only possible when using a SPARQL endpoint.

Platform specific components

Since the Mobile DSSN Client was created in a platform-independent way, it is possible to use variety of application frameworks to turn it into platform-specific hybrid application. A hybrid application combines elements of both native and web applications. Such a combination allows access to native features of the device from within the web application. However, all the layout rendering is done using the native HTML5 browser engine on the phone.

There are currently more than 15 mobile development frameworks[22] available each with different strengths and weaknesses. We selected PhoneGap to compile and package the Mobile DSSN Client for different platforms. PhoneGap was chosen because of its openness and large user community. PhoneGap being open source allows us to change practically anything inside the framework, if required.

Using PhoneGap to package the Mobile DSSN Client for specific platforms allows access to the default PhoneGap JavaScript APIs[23] (e.g. accelerometer, compass, media, etc.). Also, the powerful PhoneGap plugin system[24] is available. It allows to create platform-specific plugins using native platform SDKs and binding certain

[22] http://en.wikipedia.org/wiki/Multiple_phone_web_based_application_framework
[23] http://docs.phonegap.com/
[24] http://wiki.phonegap.com/w/page/36752779/PhoneGap-Plugins

functionality to respective JavaScript interfaces.

As an example for employing the plugin-based extension for the Mobile DSSN Client we decided to write a contacts synchronization provider for the Android platform. Android allows the creation of a custom contacts providers for synchronization with the smartphones contact book. Our WebID provider now synchronizes all `foaf:knows` and other relations from the WebID with the Android phone's contact book. PhoneGap plugins consist of two parts: native code and JavaScript interfaces. In the case where Android native code is a Java class that class extends the PhoneGap plugin system. The JavaScript interface is implemented using the PhoneGap execute method that can call any native code (or the default PhoneGap API). Since this contact synchronization provider was already implemented during the development of the Mobile Social Semantic Web client for Android (MSSW [Tramp et al., 2011b]), all that was required is to convert the existing code into a PhoneGap plugin.

5.2.4. Evaluation

The evaluation of our DSSN concept and implementation has been divided into three parts:

1. Social Web Acid Test (SWAT),
2. OntoWiki-based interoperability evaluation,
3. Dydra-based interoperability evaluation.

SWAT was used as a general check for the completeness and integration of the client into the DSSN architecture. The OntoWiki and Dydra based evaluations access how the client behaves with different data providers.

Social Web Acid Test

The Social Web Acid Test (SWAT) is an integration use case test that was conceived by the Federated Social Web Incubator Group of the W3C. Currently, only the first and very basic level of the test (SWAT0[25]) has been developed and described completely. Nevertheless, the parts of the next level (SWAT1), which are currently published, are discussed here as well.

SWAT0: The objectives of the first SWAT level are clearly described by the following use case[26]:

[25] `http://www.w3.org/2005/Incubator/federatedsocialweb/wiki/SWAT0`

[26] For this use case the following assumptions are made: (1) Users employ at least two (ideally, three) different services each of which is built with a different code base. (2) Users only need to have one account on the specific service of their choice. (3) Ideally, participants A, B, and C use their own sites (personal URLs).

Listing 5.2: Social Web Acid Test - Level 0

```
User A takes a photo of user B from her phone and posts
   it
User A explicitly tags the photo with user B
User B gets notified that she is in a photo
User C who follows user A gets the photo
User C leaves a comment on the photo
User A and user B get notified about the comment
```

Utilizing all technologies described before, the Mobile DSSN Client passes the SWAT0 without any problems. The following enumeration describes the details:

1. *User A takes a photo of user B and shares it* (using the activity creation interface): The web space returns a link to the user's pingback server in the HTTP header of the uploaded image.

2. *User A explicitly tags a photo with user B:* This is done by creating a tag resource using the same activity creation interface which links both to the image and to the WebID of user B. A pingback client sends a ping request to all of these resources after publishing the tag on the Web.

3. *User B is notified that she is on a photo:* The notification is created by the pingback service of User B who has received a request from the tagging application which was used by User A.

4. *User C, who follows user A, receives the photo:* User C is instantly provided with an update in her activity stream, informing her about the new image.

5. *User C leaves a comment on the photo:* This is done in the same way as publishing the tag.

6. *User A and user B are notified about the comment:* User A will be notified because her pingback service informs her about this ping. User B will be notified only if she has subscribed to the activity feed of the photo provided that it exists.

SWAT1 is currently not finally defined[27], thus the evaluation is only preliminary at the moment. The next SWAT level will require a few different use cases which introduces some new Social Web concepts. However, most of the user stories are already satisfied as a consequence of the fully distributed nature of the DSSN architecture (e.g. data portability and social discovery). The more interesting user

[27] Available online at `http://www.w3.org/2005/Incubator/federatedsocialweb/wiki/SWAT1_use_cases` (receive 29.07.2011).

stories are: (1) The *Private content* and *Groups* use cases will require a distributed ACL management. Some ideas for using WebIDs for group ACL management were already published with dgFOAF [Schwagereit et al., 2010] and we deem this is a good starting point for further research. (2) The *Social News* use case introduces a new vote activity. Since our architecture applies schema agnostic social network protocols, this new type of activity can be communicated as any other activity. Since most of those use cases easily fit into the DSSN architecture, they could be easily executed from the Mobile DSSN Client. The most problematic case at the moment is authorization using the FOAF+SSL protocol. Using the FOAF+SSL protocol is not yet fully supported even by the newest mobile platforms, so it is currently not yet possible to authenticate and handle private content.

OntoWiki-based interoperability evaluation

OntoWiki was developed to address the need for a Web application for rapid and simple knowledge acquisition in a collaborative way. OntoWiki can be used for presenting, authoring and managing knowledge bases adhering to the RDF data model. In order to render its functionality, OntoWiki relies on several APIs that are also available to third-party developers. Usage of these programming interfaces enables the users to extend, customize and tailor OntoWiki in several ways.

OntoWiki was selected because first DSSN architecture implementation was done using OntoWiki framework. Since OntoWiki is Linked Data enabled application, it can be used as WebID provider. Implemented during DSSN architecture development, the OntoWiki activity stream extensions provides means to create ATOM feed upon existing activities data. Also, the SPARQL endpoint provided by OntoWiki is used to update data both for WebIDs and for activity streams.

SWAT0 was used as simplest available test case. Since there are two persons in SWAT0 two OntoWiki instances were set up for Bob[28] and Alice[29]. Bob was picked as a user of the Mobile DSSN Client. The SWAT0 scenario was followed step by step. During the test no errors or problems with client or server were encountered. The Mobile DSSN Client showed that it can be used with OntoWiki as a data and update provider without any problems.

Dydra-based interoperability evaluation

Dydra[30] is a cloud-based graph database service which is free to use (currently in a private beta) and allows read/write access to different graph models in a user space. Importing and exporting of graph data is done using different APIs including SPARQL and a Dydra REST API. For querying, Dydra offers a SPARQL endpoint for each saved graph. Unfortunately Dydra currently does not support access to stored resources following the Linked Data best practices at the moment.

[28] `http://bob.lod2.eu/`
[29] `http://alice.lod2.eu/`
[30] http://dydra.com

To overcome this limitation, we created and used a Dydra linked data proxy. This proxy maps resource URLs in the namespace of the proxy installation onto `SPARQL ASK` and `CONSTRUCT` queries, which are executed on the Dydra SPARQL endpoint[31]. In order to interlink Dydra resources with DSSN services, the Linked Data proxy adds most of the auto-discovery object properties and header fields to the HTTP response. This is especially useful for WebIDs and data artefacts as comments and notes. In addition to Dydra as a WebID provider, we used a standalone Semantic Pingback service[32] as well as an activity feed service (an OntoWiki instance).

As in the OntoWiki case, SWAT0 was used as a test case. Bob's WebID was moved from OntoWiki to the Dydra store and the user's URI was changed accordingly in the client. Again, the SWAT0 scenario was followed step by step. During the test no errors or problems on client or server sides were encountered. The Mobile DSSN Client showed that it can be used with Dydra as a data provider as well as with OntoWiki as activity stream provider at the same time and stand alone Semantic Pingback service.

5.2.5. Conclusion

We see the work described in this section to be a further crucial piece in the medium-term agenda of realizing a truly distributed social network based on semantic technologies. Since ubiquitous devices are playing an increasingly important role as clients and platforms for social networks, our realization focused on providing an extensible framework for social semantic networking on the Android platform. With this work we aimed at showcasing how different (social) Semantic Web standards, technologies and best practices can be integrated into a comprehensive architecture for social networking (on ubiquitous devices).

5.3. Fat client approach

In this section, we describe *Mobile Social Semantic Web Client* that was developed using fat client approach. This section is based on [Tramp et al., 2011a] that was written in collaboration with Sebastian Tramp, Philipp Frischmuth, Natanael Arndt and Sören Auer[33].

The section is structured as follows: We introduce the idea of Social Semantic Web in subsection 5.3.1. We outline the requirements for the approach in subsection 5.3.2. We describe the general architecture of the approach in subsection 5.3.3. We explain our implementation in subsection 5.3.4. Finally, we conclude in subsection 5.3.6.

[31] In a similar way as described in Pubby [Cyganiak and Bizer, 2011], but more specific to the Dydra user/graph URL scheme.

[32] This service is available at `http://pingback.aksw.org` and was described in detail in [Tramp et al., 2010a].

[33] Author's main contribution to the paper is development of the client architecture and implenetation of the client itself

5.3.1. Introduction

Smartphones, which contain a large number of sensors and integrated devices, are becoming increasingly powerful and fully featured computing platforms in our pockets. For many people they already replace the computer as their window to the Internet, to the Web as well as to social networks. Hence, the management and presentation of information about contacts, social relationships and associated information is one of the main requirements and features of today's smartphones.

The problem is currently solved solely for *centralized* proprietary platforms (such as Google mail, contacts & calendar) as well as data-silo-like social networks (e.g. Facebook). As a result of this data centralization, users' data is taken out of their hands, they have to accept the predetermined privacy and data security regulations; users are dependent of the infrastructure of a single provider, they experience a lock-in effect, since long-term collected profile and relationship information cannot be easily transferred. Increasingly, many people argue that social networks should be evolving. That is, they should allow users to control what to enter and to keep a control over their own data. Also, the users should be able to host the data on an infrastructure, which is under their direct control, the same way as they host their own website [Berners-Lee, 2010].

A possibility to overcome these problems and to give the control over their data back to the users is the realization of a truly *distributed* social network. Initial approaches for realizing a distributed social network appeared with *GNU social* and more recently *Diaspora*. However, we argue that a distributed social network should be also based on semantic resource descriptions and de-referenceability so as to ensure versatility, reusability and openness in order to accommodate unforeseen usage scenarios.

Within the Semantic Web initiative already a number of standards and best-practices for social, Semantic Web applications such as *FOAF*, *WebID* and *Semantic Pingback* emerged. However, there is no comprehensive strategy, how these technologies can (a) be combined in order to weave a truly open and distributed social network on the Web and (b) be used efficiently in a mobile environment. Also, the use of a distributed, social semantic network should be as *simple* as the use of the currently widely used centralized social networks (if not even simpler). In this section we present the general strategy for weaving a distributed social semantic network based on the above mentioned standards and best-practices. In order to foster its adoption we developed an implementation for the Android platform, which seamlessly integrates into the commonly used interfaces for contact and profile management on mobile devices.

After briefly reviewing some use cases and requirements for a mobile, semantic social network application (in subsection 5.3.2), we make in particular the following contributions:

- We outline a strategy to combine current bits and pieces of the Semantic Web technology realm in order to realize a distributed, semantic social network (subsection 5.3.3),

- We develop an architecture for making mobile devices endpoints for the Social Semantic Web (subsection 5.3.3),
- A comprehensive implementation of the architecture was performed for the Android platform (subsection 5.3.4 and subsection 5.3.5).

Furthermore, this section concludes in subsection 5.3.6 with a discussion and outlook on future work.

5.3.2. Mobile Use Cases and Requirements

Before describing the overall strategy, the technical architecture and our implementation we want to briefly outline in this section the key requirements, which guided our work. These requirements are common sense in the context of social networks and are not newly coined by us. Unfortunately most of them are not achieved in the context of semantics enabled and distributed social networks, so we describe them especially from this point of view.

Make new friends. Adding new contacts to our social network is the precondition in order to gather useful information from this network. Maintaining our social network directly from your mobile phones means that we are able to instantly connect with new contacts (e.g. on conferences or parties). In the context of a distributed social network, this use-case also includes the employment of semantic search engines to acquire the WebID of a new contact based on parts of its information (typically the contacts name). In order to shorten the overall effort for adding new contacts, functionality for scanning and decoding a contacts business cards QR code[34] are also included in this use-case.

Be in sync with your social network. Once our social network is woven and social connections are established, we want to be able to gather information from this network. For a distributed social network this means, that a combination of push and pull communications is needed to be as timely updated as needed and as fast synced as possible. Especially this use-case is bound to a bunch of access control requirements[35]. where people want to permit and deny access to specific information in fine grained shades and based on groups, live contexts and individuals.

[34] QR codes are two-dimensional barcodes which can encode URIs as well as other information. They are especially famous in Japan, but their popularity grows more and more worldwide since mobile applications for decoding them with a standard camera can be used on a wide range of devices.

[35] A typical requirement: Disallow access to my mobile number except for friends and family members.

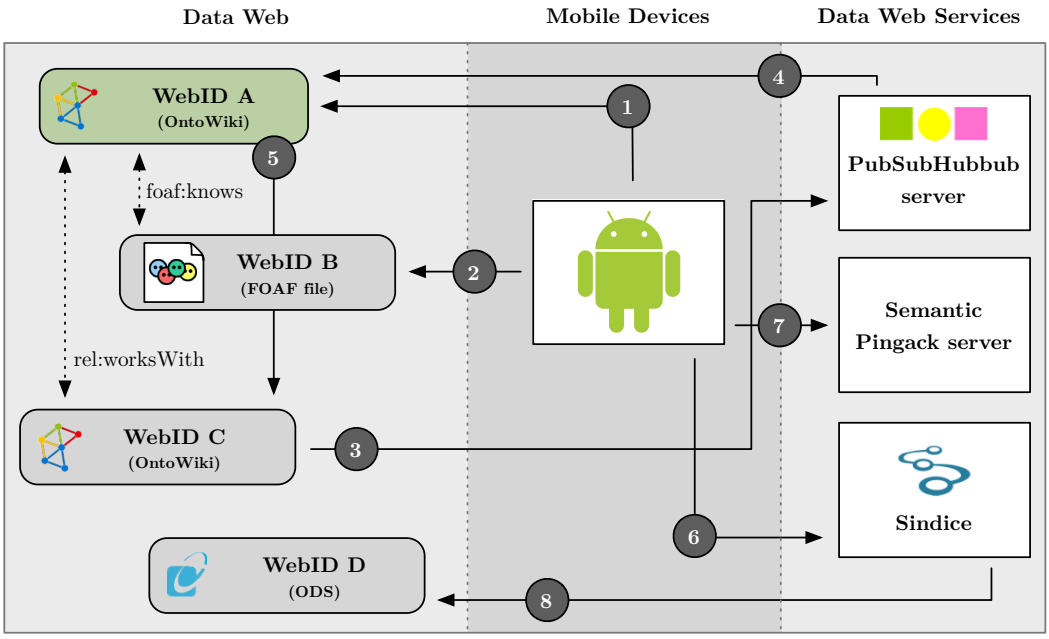


Figure 5.10.: Architecture of a distributed, semantic social network: (1) A mobile user may retrieve updates from his social network via his WebID provider, e.g. from OntoWiki. (2) He may also fetch updates directly from the sources of the connected WebIDs. (3) A WebID provider can notify a subscription service, e.g. a PubSubHubbub server, about changes. (4) The subscription service notifies all subscribers. (5) As a result of a subscription notification, another node can update its data. (6) A mobile user can search for a new WebID by using a semantic search engine, e.g. Sindice. (7) To connect to a new WebID he sends a Pingback request which (8) notifies of the resource owner.

Annotate contacts profiles. It should be possible to annotate profiles of contacts freely, e.g. with updated information, contact group categorizations (e.g. friends, family, co-workers). These annotations should be handled in the same way as the original data from the friend's WebID except that this data is not updated with the WebID but persists as an annotation. One additional feature request in this use-case is to share these annotations across ones personal devices on the web, e.g. by pushing them to a triple store which is attached to ones WebID.

General requirements. The development of the Mobile Social Semantic Web Client was driven by a few general requirements which derived from our own experience with mobile phones and FOAF-based WebIDs:

- *Be as decent as possible:* Today's FOAF-based social networks are mostly driven by uploaded RDF files. In order to support such low end profiles, there should be no other required feature on a WebID than the availability

as Linked Data[36]. All other features (FOAF+SSL, Semantic Pingback, subscription service) should be handled as optional and our client should require as little infrastructure as possible.

- *Be as transparent as possible:* Mobile user interfaces are built for efficiency and daily use. People become accustomed with them and any changes in the daily work flow of using information from the social network will annoy them. The client we had in mind should work mostly invisible from the user, which means it should be well integrated into the hosting mobile operating system.

- *Be as flexible as possible:* This is especially needed in an environment where vocabularies are not yet standardized and are subject to changes and extensions. Our solution should be flexible in the sense that we do not want built-in rules on how to deal with specific attributes or relations.

Based on these preliminaries as well as based on the Social Semantic Web state of the art, we describe an architecture of a distributed social semantic network in the next section.

5.3.3. Architecture of a Distributed Semantic Social Network

In this section we describe the main ingredients for a distributed, semantic social network as well as their interplay. The overall architecture is depicted in Figure 5.10. The semantic representation of personal information is facilitated by WebID. FOAF+SSL allow the use of a WebID for authentication and access control purposes. Semantic Pingback facilitates the first contact between users of the social network and subscription services allow obtaining specific information from people in ones social network as near-instant notifications.

WebID. WebID [Sporny et al., 2010] is a best-practice recently conceived in order to simplify the creation of a digital ID for end users. Since its focus lies on simplicity, the requirements for a WebID are minimal. In essence, a WebID is an de-referenceable RDF document (including RDFa) describing its owner[37]. That is, a WebID contains RDF triples, which have the IRI identifying the owner as subject. The description of the owner can be performed in any (mix of) suitable vocabularies, but FOAF [Brickley and Miller, 2004] emerged as the 'industry standard' for that purpose. An example WebID comprising some personal information (lines 8-12) and two `rel:worksWith`[38] links to co-workers (lines 6-7) is shown in Listing 5.3.

[36] In the meaning of Linked RDF Data defined at `http://www.w3.org/DesignIssues/LinkedData`.

[37] The usage of an IRI with a fragment identifier allows the indirect identification of a WebID by reference to the (FOAF) profile document.

[38] Taken from *RELATIONSHIP: A vocabulary for describing relationships between people* at `http://purl.org/vocab/relationship`.

Listing 5.3: A minimal WebID with personal information and two *worksWith* relations to other WebIDs.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://purl.org/vocab/relationship/> .
<http://philipp.frischmuth24.de/id/me> a foaf:Person;
  rdfs:comment "This is my public profile only, more
     information available with FOAF+SSL";
  rel:worksWith <http://sebastian.tramp.name>,
    <http://www.informatik.uni-leipzig.de/~auer/foaf.rdf
       #me>;
  foaf:depiction <http://img.frischmuth24.de/people/me.
     jpg>;
  foaf:firstName "Philipp"; foaf:surname "Frischmuth";
  foaf:mbox <mailto:frischmuth@informatik.uni-leipzig.de
     >;
  foaf:phone <tel:+49-341-97-32368>;
  foaf:workInfoHomepage <http://bis.informatik.uni-
     leipzig.de/PhilippFrischmuth>.
```

FOAF+SSL. The more technical FOAF+SSL best-practice [Story et al., 2009b] aims to incorporate authentication functionality into the WebID concept. The main idea is to link an SSL client certificate to a WebID, thus allowing the owner of the FOAF+SSL enabled WebID to authenticate herself at 3rd party websites. Another goal of FOAF+SSL is to provide access control functionality for a social network shaped by WebIDs in order to allow access to different kinds of information for different groups of contacts (e.g. as presented with dgFOAF [Schwagereit et al., 2010]). An example of a FOAF+SSL WebID extension is shown in Listing 5.4. This WebID now contains a description of an RSA public key (line 15), which is associated to the WebID by using the `cert:identity` property from the W3C certificates and crypto ontology (line 19).

Semantic Pingback. The purpose of Semantic Pingback [Tramp et al., 2010a] in the context of a distributed social network is to facilitate the first contact between different people using the network. The approach is based on an extension of the well-known Pingback technology [Langridge and Hickson, 2002], which is one of the technological cornerstones of the overwhelming success of the blogosphere in the Social Web. The Semantic Pingback mechanism enables bi-directional links between WebIDs, RDF resources as well as weblogs and websites in general. It facilitates contact/author/user notifications in case a link has been newly established. It is based on the advertising of a lightweight RPC service, in the RDF document, HTTP or HTML header of a certain Web resource, which should be called as soon

Listing 5.4: Extension of the minimal WebID from Listing 5.3: Description of an RSA public key, which is associated to the WebID by using the `cert:identity` property from the W3C certificates and crypto ontology.

```
@prefix rsa: <http://www.w3.org/ns/auth/rsa#> .
@prefix cert: <http://www.w3.org/ns/auth/cert#"> .
[] a rsa:RSAPublicKey;
  rdfs:comment "used from my smartphone ...";
  cert:identity <http://philipp.frischmuth24.de/id/me>;
  rsa:modulus           [ cert:hex "C41199E ... 5AB5" ];
  rsa:public_exponent   [ cert:decimal "65537" ] .
```

as a (typed RDF) link to that resource is established. The Semantic Pingback mechanism enables people but also authors of RDF content, a weblog entry or an article in general to obtain immediate feedback, when other people establish a reference to them or their work, thus *facilitating social interactions*. It also allows to automatically publish backlinks from the original WebID (or other content) to comments or references of the WebID (or other content) elsewhere on the Web, thus *facilitating timeliness and coherence* of the Social Web. As a result, the distributed network of WebID profiles, RDF resources and social websites using the Semantic Pingback mechanism can be much tighter and timelier interlinked than conventional websites, thus rendering a network effect, which is one of the major success factors of the Social Web. Semantic Pingback is completely downwards compatible with the conventional Pingback implementations, thus allowing the seamless connection and interlinking of resources on the Social Web with resources on the Data Web. An extension of our example profile with Semantic Pingback functionality making use of an external Semantic Pingback service is shown in Listing 5.5.

Listing 5.5: Extension of the minimal WebID from Listing 5.3: Assignment of an external Semantic Pingback service which can be used to ping this specific resource.

```
@prefix ping: <http://purl.org/net/pingback/> .
<http://philipp.frischmuth24.de/id/me> ping:to <http://
   pingback.aksw.org>.
```

Subscription Service. The purpose of a WebID subscription service is to establish a publish/subscribe communication model to provide near-instant notifications of contact updates. The main idea here is to extend a WebID with a link to a *PubSub-*

Hubbub service[39] where any contact can subscribe to the WebIDs updates. Although such a behavior is described for SPARQL results in [Passant and Mendes, 2010], there is currently no standardized solution for publishing RDF change sets through PubSubHubbub as well as for saving the incoming changes from all friends of a user in some kind of cache or proxy while the mobile device (the subscriber) is not online. As a consequence, our implementation (as described in the next section) does not yet support a full-fledged update subscription. As a fallback, updates are currently polled from the related WebIDs. This increases network bandwidth usage and might lead in some cases to slower user interfaces due to network latency. Please refer to Section 5.3.6 for a description of possible future work in this direction.

5.3.4. Implementation of a Mobile Interface

After describing the architecture of a distributed, semantic social network we now present our implementation of a mobile interface for this network.

Android System Integration

Figure 5.11 depicts the mobile social Semantic Web client consisting of two application frameworks, which are built on top of the Android runtime and a number of libraries. In particular, *androjena*[40] is one of those libraries, which itself is a partial port of the popular Jena framework[41] to the Android platform. Both frameworks provided by the client share the feature that they are accessible through content providers. The Mobile Semantic Web middleware (MSW) is responsible for importing Linked Data resources (in particular via FOAF+SSL) and persisting that data. It operates on triple level and provides access to the various triple stores through a content provider called `TripleProvider`. Each resource is stored separately, since named graphs are currently not supported. The Mobile Social Semantic Web middleware (MSSW) queries the triple data provided by MSW and transforms that data into a format that is more appropriate for social applications. It propagates two content providers, one that integrates well with the layout of contact information on Android phones (`ContactProvider`) and one that is suitable for FOAF based applications (`FoafProvider`).

Model Management

Since WebIDs are Linked Data enabled, they usually return data describing that resource. This circumstance makes it feasible to store a graph (referred to as a model here) for each WebID, since the redundancy between models is expected to

[39] PubSubHubbub is an open, server-to-server web-hook-based publish/subscribe protocol realized as an extension to Atom: `http://code.google.com/p/pubsubhubbub/`

[40] `http://code.google.com/p/androjena/`

[41] `http://jena.sourceforge.net/`

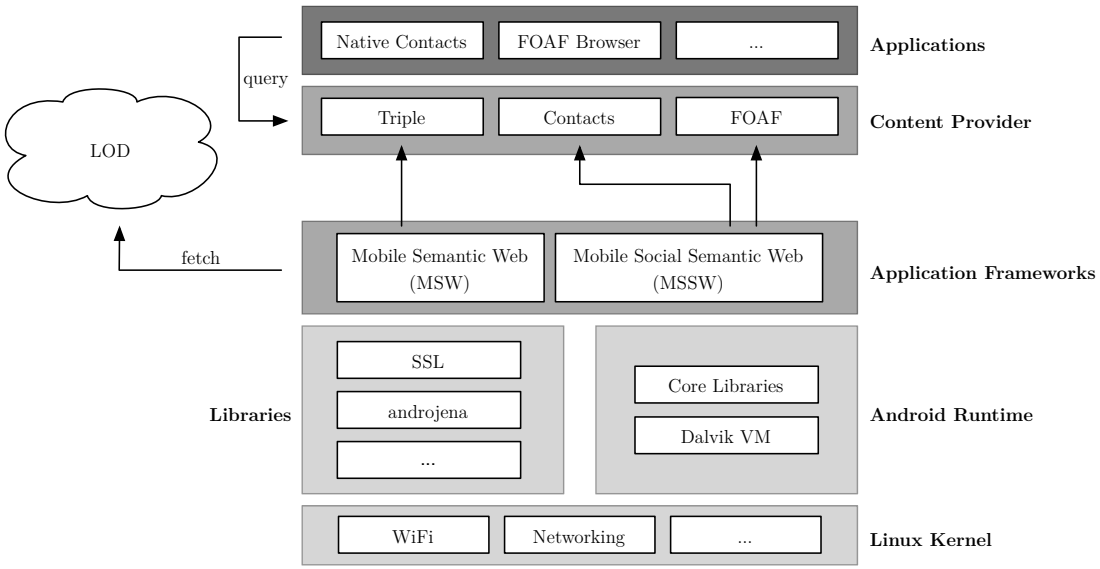


Figure 5.11.: Android Integration Layer Cake

be marginal. In reality MSW keeps more than one model per WebID for different purposes. On the mobile phones' SD-card we keep these models in the following subdirectories:

- `web` – This folder contains exact copies of the documents retrieved from the Web.
- `inf` – Models stored in this folder contain all entailed triples (more on this in Section 5.3.4).
- `local` – The user can annotate all WebIDs with personal information, which will be stored in this folder.

We decided to store all data as RDF files on a swappable SD-card, since we expect the following user benefits:

- Because SD-cards can be exchanged, the data is portable and can be reused on another phone or device. This makes the whole system more fail-proof.
- Most modern computers can handle SD-cards and hence data can be easily backed up.
- Other applications on the Android phone running the mobile Semantic Web client can access and modify the data stored on the card. Thus they can further annotate the information and the client can again take advantage of such annotations.

Rules and Data Processing

One of our initial requirements from Section 5.3.2 is flexibility in the sense that specific vocabulary resources should not be encoded in the source code of the WebID provider. In order to achieve this requirement, we decided to encode as much data processing as possible in terms of user extensible rules. Since we employ the *androjena framework*, we were able to use the included Jena rules engine as well. All rules processed by this rule-based reasoner are defined as lists of body terms (premises), lists of head terms (conclusions) and optional names[42].

Listing 5.6: Example transformation rule: If a `foaf:jabberID` is present with a WebID (line 7), then a new blank node of RDF type `acontacts:Im` is created (line 7), which is of Android IM type `HOME` (line 11) and which gets an IM protocol as well as the IM identifier (line 12 and 10).

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix android: <http://ns.aksw.org/Android/>.
@prefix acontacts: <http://ns.aksw.org/Android/
   ContactsContract.CommonDataKinds.>.
@prefix im: <http://ns.aksw.org/Android/ContactsContract
   .CommonDataKinds.Im.>.

[jabber:
  (?s foaf:jabberID ?o), makeTemp(?d) ->
    (?s android:hasData ?d),
    (?d rdf:type acontacts:Im),
    (?d im:DATA ?o),
    (?d im:TYPE im:TYPE_HOME),
    (?d im:PROTOCOL im:PROTOCOL_JABBER)
]
```

Since we also did not want our implementation to depend on the FOAF vocabulary (alternative solutions include RDF vCards [Iannella et al., 2010]), we decided to create a native Android system vocabulary which represents the Android contacts database defined by the Android API. This vocabulary is deeply integrated into the Android system since it re-uses class and attribute names from the Android API and represents them as OWL class and datatype properties[43].

Based on this vocabulary, the given rules transform the downloaded WebID statements into Android-specific structures which are well suited for a straight-

[42] `http://jena.sourceforge.net/inference/#RULEsyntax`

[43] An example class name is `ContactsContract.CommonDataKinds.StructuredName`, which is represented in the vocabulary as an OWL class with the URI `http://ns.aksw.org/Android/ContactsContract.CommonDataKinds.StructuredName`. We published the vocabulary at `http://ns.aksw.org/Android/`. Please have a look at the Android API reference as well (`http://developer.android.com/`).

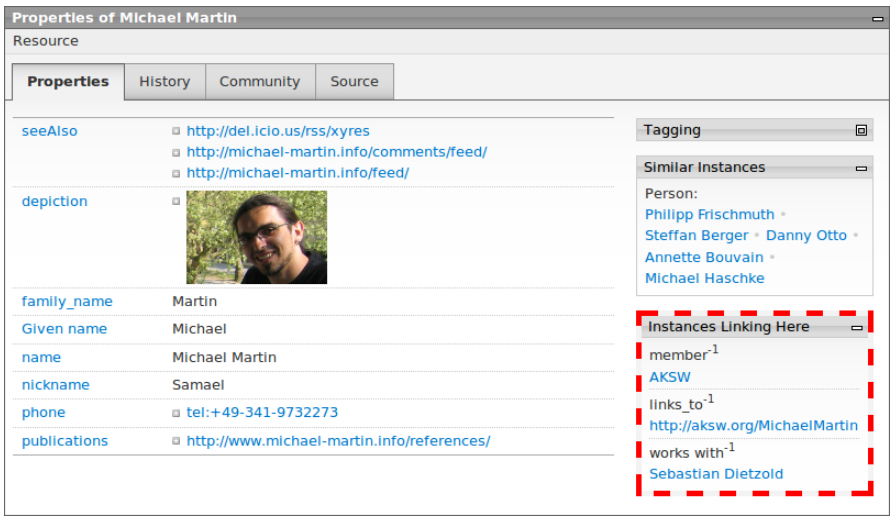


Figure 5.12.: Visualization of a WebID in OntoWiki: incoming backlinks (via Semantic Pingback) are rendered in the "Instances Linking Here" side box.

forward import into the contacts provider. These structures are very flat and relate different Android data objects (e.g. email, photo, structured name etc.) via a `hasData` property to a WebID. An example rule which creates an instant messaging account for the contact is presented in Listing 5.6.

After applying the given set of rules, the application post-processes the generated data in order to apply other constraints which we could not achieve with Jena rules alone. At the moment all `mailto:` and `tel:` resources are transformed to literal values, which is required for instantiating the corresponding Java class. In addition we download, resize and base64-encode all linked images. After that, the application goes through the generated data resources and imports them one by one.

OntoWiki

The mobile Semantic Web client supports arbitrary WebIDs, even those backed by plain RDF files. Nevertheless, some features require special support on the server-side. For our semantic data wiki OntoWiki [Auer et al., 2006b] we implemented all functionalities required for a complete distributed Social Web experience. Any user can setup his own OntoWiki instance, which will then provide him with an enhanced WebID.

If configured properly a user can create a self-signed certificate with very little effort. Such a certificate contains the generated WebID as a Subject Alternative Name (SAN) and is directly imported into supported Web browsers[44]. From the

[44] A list of supported browsers is available at `http://esw.w3.org/Foaf+ssl/Clients`.

browser the certificate can be exported in *PKCS12* format and stored on a SD-card used by a mobile phone running the client. Since OntoWiki supports FOAF+SSL authentication, a user can split his data in publicly visible information and such, that is only accessible by people which have a certain relationship with the user (e.g. a `foaf:knows` relation).

Semantic Pingback is another technology supported by OntoWiki. Thus an arbitrary user can add a relationship to an OntoWiki backed WebID and as a result the WebID owner will be notified, enabling the user to take further actions (see Figure 5.12). In the use case of the mobile Semantic Web client this is especially useful for a first contact between users. In typical social network applications this step would be the "Add as a friend" step. In a distributed scenario, however, if one states that she is a friend of someone else, she would allow that person to view the data dedicated to be displayed by friends only. If both endpoints add that relation on their respective side, they can see each other's private data and thus are considered friends (in the Social Web sense). The Social Web has a very dynamic nature and information is changed frequently or new data is added. Hence, editing functionality is another important aspect and OntoWiki supports editing via SPARQL/Update.

5.3.5. User perspective

The Mobile Semantic Social Web client implementation consists of two software packages - the *Android Semantic Web Core library* containing the triple store and the *WebID content provider for Android*. Both are available on the *Android Market* since August 2010 (cf. screenshot A in Figure 5.13). According to the market statistics, they were downloaded overall more than 400 times and are currently installed on more than 100 devices.

Once installed few initial configuration options have to be supplied. Screenshot B in Figure 5.13 shows the accounts and sync settings configuration menu, which allows a user to associate his WebID with his profile on the smartphone (the same way as adding an LDAP or Exchange account) and to configure synchronization intervals. Screenshot C shows an actual WebID with the last synchronization date and the option to trigger the synchronization manually.

After the user associated his profile with his WebID, information from linked WebIDs of the users contacts are synchronized regularly and the information are made available via the Android content provider to all applications on the device. During the import of the WebID contacts, they are merged based on the assumption of unique names. Independent of this automatic merge, the user can split and merge contacts manually in the edit view of these contacts. Screenshot D shows the standard Android contact application, where our WebID content provider seamlessly integrates information obtained from WebIDs. Information obtained from WebIDs is not editable, since it is retrieved from the authoritative sources, i.e. the WebIDs of the respective contacts.

Screenshot E shows the FOAF browser, allowing people to add contacts or to

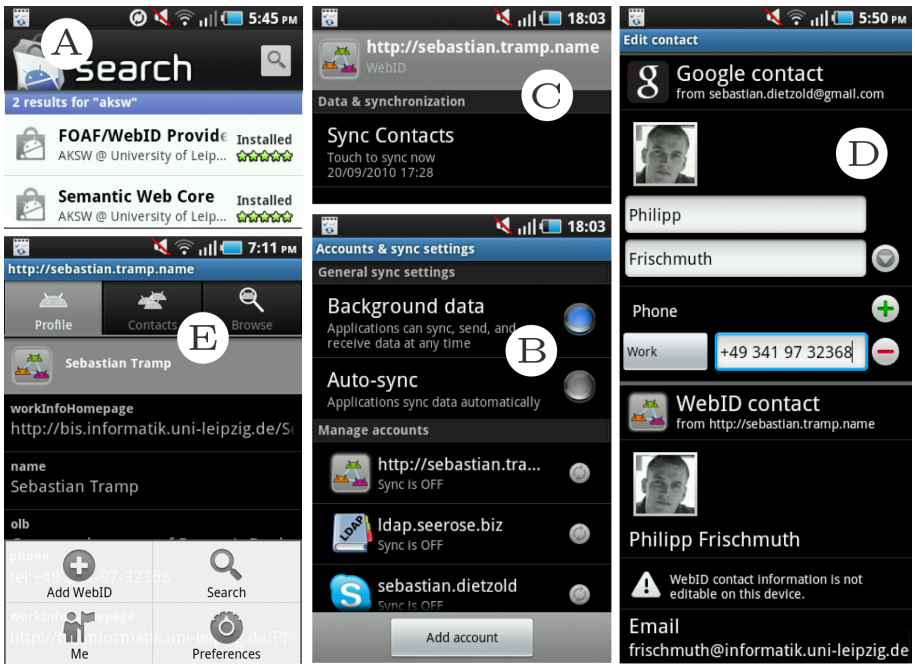


Figure 5.13.: Screenshots of the Mobile Social Semantic Web Client, the FOAF Browser and the Android components which integrate the WebID account: (A) The client as well as the triple store can be found in the official Google application market. (B) After installation, users can add a WebID account the same way they add an LDAP or Exchange account. (C) The account can be synchronized on request or automatically. (D) A contacts profile page merges the data from all given accounts. (E) By using the FOAF browser, people can add contacts or browse the contacts of their friends.

browse the contacts of their friends. In order to facilitate the process of connecting with new contacts the Android implementation also allows to scan QR-codes of WebIDs (e.g. from business cards) and to search for WebIDs using Sindice.

5.3.6. Conclusion

We see the work described in this section to be a further crucial piece in the medium-term agenda of realizing a truly distributed social network based on semantic technologies. Since mobile devices are playing an increasingly important role as clients and platforms for social networks, our realization focused on providing a extensible framework for social semantic networking on the Android platform. With this work we aimed at showcasing how different (social) Semantic Web standards, technologies and best practices can be integrated into a comprehensive architecture for social networking (on mobile devices).

6. Provider Approaches

This chapter provides a general overview of existing provider approaches. There are two provider approaches: *Fat Provider* and *Hybrid Provider*.

6.1. Fat provider approach

In this section, we describe *Embedded Linked Data Server* approach that was developed using fat provider approach.

This section is based on [Ermilov and Auer, 2013].

The section is structured as follows: We introduce the Internet of Things and ELDS in subsection 6.1.1. We describe our approach for adding a linked data interface to smart devices in subsection 6.1.2. We discuss our implementation in subsection 6.1.3. The evaluation methodology along with the corresponding results for our implementation is described in subsection 6.1.4. Finally, we conclude in subsection 6.1.5 with an outlook on future work.

6.1.1. Introduction

The term Internet of Things [Atzori et al., 2010] refers to the vision, that all kinds of physical objects are uniquely identifiable and have a virtual representation on the Internet. The unique identification can be realized using barcodes, RFIDs [Wang et al., 2006] or embedded systems and smart internet-enabled devices. In the former two cases the object itself can only identify itself and a virtual representation has to be hosted elsewhere. However, increasingly often some form of intelligence is embedded into the objects themselves (e.g. by integrating a system on a chip into a TV set or manufacturing equipment) or the object itself is a smart device (e.g. a smartphone or tablet PC). As a result, these devices can not only identify but also describe themselves by providing comprehensive information. There have been first attempts of integrating Web servers and hosting Web-accessible information within such devices (e.g. [Guinard and Trifa, 2009]). However, as we meanwhile complemented the Web of Documents with a Web of Semantic Data, information provided on the Internet of Things should be made available in standardized and semantically structured form as well.

In this section, we present an approach for equipping embedded and smart devices with a Linked Data interface. The approach is based on mapping existing structured data on on the device to vocabularies and ontologies and exposing this information as dereferencable RDF directly from within the device. The technical

architecture comprises a Web server running on the device, which serves content provided by a management service from a embedded triple store and device specific data stores. We implemented our approach for Android, which is an increasingly popular operating system not only for smartphones and tablet PCs, but also for smart TVs, navigation systems, cash registers and many other smart devices. Also, our implementation is easily adaptable for other Linux or Unix based embedded OS, such as *FritzOS*, *Firefox OS*[1] or *Sailfish OS*[2]. As a result, all smart devices can easily provide standardized structured information and become first class citizens on the Data Web.

Equipping smart devices with Linked Data interfaces has a number of advantages including:

- *Standardization.* Other data syndication and integration techniques are mostly proprietary and require integration at design time. With Linked Data interfaces, smart devices can expose, exchange and integrate data in ways unforeseen at design time.
- *Timeliness.* Since the data is directly exposed from the device where it originates, there are no delays related to data replication, caching etc. Persistence proxy services (similar to `purl.org`) can be used to maintain access to the data when devices are offline.
- *Privacy and data security.* Users' data is kept where it belongs (on their devices) and does not have to be centrally stored in order to be exchanged. Also, using FOAF+SSL and access control mechanisms data can be exposed in a fine-grained way.

A particular specific requirement when dealing with smart and embedded devices are resource constraints. Due to progress in miniaturization, memory and processing power is meanwhile not a constraining factor anymore for most applications. Power consumption on the other hand is a key aspect, when equipping devices with additional functionality. Hence, a particular focus of our implementation is limitation of the impact on power consumption and for this purpose and due to lack of existing standards in the area we develop a performance vs. power benchmarking methodology. The evaluation of our approach using this methodology shows, that the overhead introduced by equipping a device with a Linked Data interface is neglectable given modern software and hardware environments and moderate usage.

6.1.2. Approach: Embedded Linked Data Server

The main goal of our approach is to enable any smart device (e.g. tablets, smart phones, TVs) to identify and describe itself by providing comprehensive

[1] `http://www.mozilla.org/en-US/firefox/partners/#os`
[2] `http://sailfishos.org/`

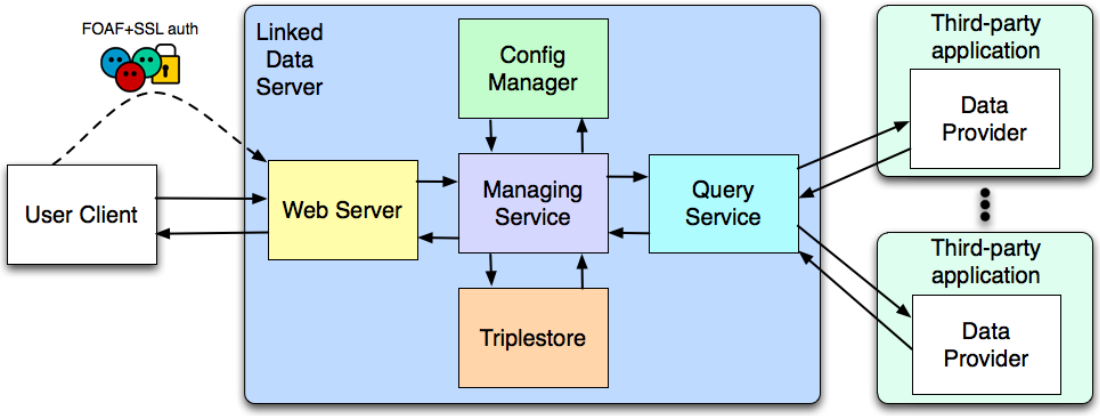


Figure 6.1.: Overview of the Embedded Linked Data Server architecture.

information in accordance with the Linked Data principles. As a way to accomplish this goal, we propose the concept of an *Embedded Linked Data Server* (ELDS) that comprises a Web server, which hosts Web-accessible RDF information as Linked Data directly from within the smart device. The ELDS concept is based on an on-demand transformation of internal smart device data structures to RDF by using user-provided mappings.

Architecture

The overall ELDS architecture is shown in Figure 6.1 and includes three layers:

1. *User layer* – comprises a user client,
2. *ELDS layer* – comprises ELDS internal components,
3. *Third-party application layer* – comprises third-party applications.

The separation in these three layers not only facilitates modularity and maintainability, but also aims at minimizing the impact on power consumption. Components of the user and third-party layer, for example, can easily be suspended thus minimizing the main memory and power consumption requirements.

The *User layer* includes user client application that can process RDF data and optionally a user WebID that is used for authentication and access control to the data provided by ELDS. *ELDS layer* includes all of the solution internal components. In particular, the *Web Server* component that receives requests from and sends replies to the user client. It also handles authentication and access control by utilizing FOAF+SSL protocol [Story et al., 2009a], if necessary. The *Managing Service* acts as the main controller that directs all other internal components and manages most of the ELDS workflow. It receives request parameters from the Web Server, determines which data mapping configuration file should be applied, passes

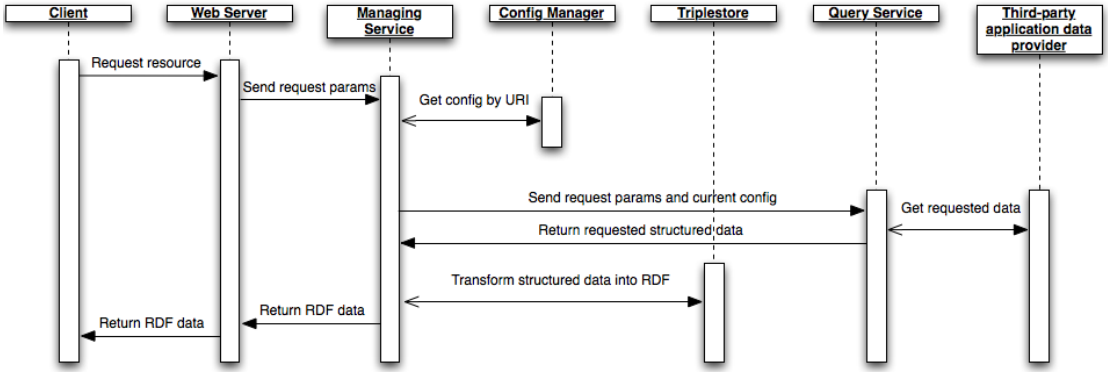


Figure 6.2.: ELDS workflow during Linked Data access.

parameters to the Query Service and transforms structured data to RDF using the Triplestore component. The *Configuration Manager* component is responsible for loading and managing the data mapping configuration files provided by the user. The *Query Service* is responsible for fetching the data from third-party application data providers using a configuration object that is passed to it. The *Triplestore component* is used by the Managing Service to transform structured data that was received from the Query Service component into the requested RDF serialization. *Third-party application layer* includes external applications that can act as structured data providers inside of the smart device or object (e.g. Android contacts data provider).

The basic ELDS workflow during Linked Data access is shown in Figure 6.2 and consists of following steps:

1. The client requests a specific resource description from ELDS on the smart device using an URI.

2. If ELDS requires authentication and enforces access control, the client must complete a FOAF+SSL authentication procedure (optional).

3. The Web Server component receives the request and passes its parameters to the Managing Component.

4. The Managing Component identifies configuration ID and fetches the required configuration object from Configuration Manager.

5. The configuration object together with other request parameters is sent to the Query Service.

6. The Query Service executes the query on third-party data providers applying the given parameters and returns results to the Managing Component.

7. The Managing Component transforms the results it has received from the Query Service into requested RDF serialization using the triple store component in accordance to the current configuration object.

8. The Managing Component passes resulting RDF serialization back to Web Server component.

9. The Web Server component returns the resulting RDF serialization back to client.

Use cases

There are a vast number of potential use cases for ELDS. Examples include (a) integrating data from an ELDS-based cash register into an Enterprise Resource Planning (ERP) system, (b) autonomous weather stations directly publishing measurements, (c) exposing contact data in a distributed social network, (d) setting or publishing usage history and contexts from a smart TV or game console for content recommendation. We briefly outline the latter two use cases in the sequel.

The first use case where ELDS fits perfectly is using it as a part of Distributed Semantic Social Network (DSSN) [Tramp et al., 2011c]. Including ELDS in the DSSN architecture allows easy access to device internal data in a standardized non-proprietary way, e.g. location information or address book. Enforcing WebID authorization will help protect the data from unwanted access by third parties. This would enable usage scenarios, where a user would not need to contact his friend to ask for her location, but could just poll this information directly from her friend's smart device without the need for a central service. Similarly, when knowing that a user's friend has third-party contact data that the user is interested in, she could obtain the required contact data directly from smart device without bothering her friend (as long as she was granted access to this information).

A second use case is related to growing popularity of smart TVs and gaming consoles. For example, devices such as the Android-based *Google TV*[3] or the gaming console *Ouya*[4] can be easily turned into linked data providers. By exposing usage history from such devices (e.g. movies that user watched from Google TV, or games that user played from Ouya) in Linked Data format, it is possible to enable personalized recommendations in a non-intrusive way. After history information is accessible as Linked Data, history entries can be interlinked with existing databases like DBpedia or IMDB to enhance the performance of a recommendations system. It is also possible to display extracted history information to user using some form of user interface where she can specify whether she liked a specific entry or not thus enhancing recommendations.

[3] `http://www.google.com/tv/`
[4] `http://www.ouya.tv/`

6.1.3. Implementation: Android Linked Data Server

We have implemented an Android specific version of ELDS – *Android Linked Data Server* (ALDS)[5]. We used existing Android implementations of two components as integral parts of ALDS. The Android version of *Jetty*[6] was used as Web Server and the *Androjena* library[7] as a triple store. We developed and integrated a *Configuration Manager* to dynamically load configuration files based upon JSON formatted files. A *Query Service* was developed to query third-party Android Content providers on demand.

Content providers manage access to structured (i.e. tabular) data on Android platform. Content providers are the standard interface that connects data from one process with code running in another process. Content providers can be addressed using a URI. Querying is performed by specifying the content provider URI, query conditions and columns that must be returned.

The best way to implement a long running service on the Android OS is to utilize the *Bound Service*. Running ADLS as a service allows to work in background without the need for any additional user interface or application running. This will also allow to throttle resource usage in favor of foreground tasks more easily. Another feature of the Android Bound Service is dynamic event handling. This can be used to dynamically update and reload configuration files by a user interface or third-party applications. The steps to add a new configuration file (or reload update file) are:

- Generate (or update) a JSON description file for a specific content provider,
- Place the JSON description file into the application configurations folder,
- Trigger the ALDS Configuration Manager to reload (either via the UI or Bound Service).

After receiving the configuration, the object ALDS will expose a Content Provider that was described by the given configuration using Linked Data. The internal Android structures are transformed into RDF using Androjena according to the description it has received from a configuration file.

Data Access and Mapping Configuration

To simplify creation and implementation of the ALDS configurations, we decided to use simple JSON formatted text files. As JSON is a text-based open standard designed for human-readable data interchange such configuration files can be easily generated by both users and third-party applications. As an example, ALDS includes configuration file that transform basic contact information (i.e. name

[5]Available at: `https://github.com/AKSW/ALDS`
[6]`http://www.eclipse.org/jetty/`
[7]`https://code.google.com/p/androjena/`

and phone number) using FOAF vocabulary to RDF. This example is shown on Listing 6.1.

The first step to follow when creating a configuration for ALDS is to define a URI of a content provider. This is done by specifying the `provider_uri` string field. Optionally, custom prefixes can be defined to simplify latter creation of the bindings to the data. This is accomplished by specifying `rdf_prefixed` key-value array, where keys are prefixes and values are prefix URIs used during transformation. The next step is to define a set of columns that should be fetched from the Content provider as well as rules on how the column data should be transformed to RDF representation. This is accomplished by specifying the `columns` array. It consists of objects that describe content provider columns and their data bindings to RDF. The objects that describe columns include three fields:

- *id* – column identifier in third-party Content provider,
- *name* – user-readable name (optional, can be blank),
- *predicate* – predicate used during data transformation to RDF, can use prefixes defined earlier, if `null` the column will be skipped.

The final step is to define rules which facilitate the generation of external URLs. This is done by specifying the `uri_generation` object. It includes two fields:

- *prefix* – defines a prefix that is used for current configuration (should be unique across all configurations),
- *values* – defines how the resource unique URI part is generated from specified column data values.

Basic workflow example. Listing 6.1 is a configuration exposing phone contacts as Linked Data. Imagine a scenario where a user client requests an URL for Angela Merkel's contact information using:

```
http://device/phones/1912__Angela+Merkel
```

Following the workflow, ALDS will determine the configuration ID from the given URL, which is `phones`. The configuration object for `phones` along with a request parameters string (i.e. "`1912__Angela+Merkel`") will then be passed to the Query Service. The Query Service splits the parameters string into separate values according to the configuration object. In this case the values `1912` and `Angela Merkel` are matched to columns `_id` and `display_name`. The Query Service then executes a query to the content provider that has the following URI:

```
content://com.android.contacts/data/phones
```

Listing 6.1: Example configuration which maps the contacts provider to the FOAF vocabulary.

```
provider_uri: "content://com.android.contacts/data/phones",
rdf_perifxes: {
  "foaf": "http://xmlns.com/foaf/0.1/"
},
columns: [
  {
    id: "_id",
    name: "id",
    predicate: null
  },
  {
    id: "display_name",
    name: "username",
    predicate: "foaf:name"
  },
  {
    id: "data1",
    name: "number",
    predicate: "foaf:phone"
  }
],
uri_generation: {
  prefix: "phones",
  values: ["_id", "display_name"]
}
```

The query requests a specific contact with parameters that were extracted before and set of columns defined in configuration object, i.e.: `_id`, `display_name` and `data1`. After fetching the data the Query Service passes it to the Managing Component. The Managing Component then applies the transformation rules that are described in the configuration object. Specifically it binds data from all columns with defined predicated to the current resource URI. The resulting output can be seen in Listing 6.2.

Listing 6.2: Example output

```
@prefix foaf:     <http://xmlns.com/foaf/0.1/> .

<http://device/phones/1912__Test+User>
      foaf:name "Test User" ;
      foaf:phone "+491761234567" .
```

6.1.4. Evaluation

One of the most important things to consider regarding mobile and embedded device usage is power consumption. Thus, the ALDS impact on power consumption of mobile devices is our main evaluation target. In addition, we measure the ALDS response time that influences the user experience during interaction with the ALDS service.

Evaluation methodology

Our evaluation methodology is inspired by existing power consumption research (e.g. [Thiagarajan et al., 2012], [Balasubramanian et al., 2009]) and adopted to our specific needs (i.e. measuring impact of just one application instead of multiple and avoidance of low level voltage measurements). For a single evaluation run we picked a 2 hours timespan in order to truly observe the power consumption impact. During the evaluation period a 54 Mbit/s WLAN connection and a 3G GSM network connection were enabled and permanently maintained. Over the evaluation duration, the device was not touched, had its display turned off in order to increase the measurement precision of the ALDS impact on power consumption (except for the heavy front end load case). Measures of power consumption were taken using the *Battery Log* application[8]. A desktop computer running a simple evaluation script was acting as a client. The script executed requests with a set interval, verified response from ALDS and recorded the response time. In order to exclude a possible influence on WLAN capacity and performance by other devices, the only two devices connected to the WLAN were the mobile device under evaluation and the desktop computer running the evaluation script.

Before beginning the evaluation, a warm-up phase was performed for the ALDS and Android components by accessing RDF resources provided by ALDS three times without logging the results. That was required to prevent distortion of the evaluation results, since directly after the launch of the services it takes 3100 ms on average to get response from ALDS.

Evaluation testbed

Two different Android devices were used for evaluation. The first device is the mobile phone Samsung Galaxy i9003 SL running stock Android v2.3.6 (Gingerbread). It features the TI OMAP 3630 chipset, comprising a 1 GHz single core Cortex-A8 CPU, 2 GB of internal storage complemented by 16 GB SD card, 478 MB RAM and a 1650 mAh Li-Ion battery. The device was in constant use since June 2011, which means that battery was 23 month old at the time of evaluation.

The second device is the tablet Smartbook Surfer 360 MN10U running custom Android 4.1.2 (JellyBean). It features the Nvidia Tegra 250 chipset, comprising a 1 GHz dual core Cortex-A9 CPU, 512 MB of internal storage complemented by 16

[8] `https://play.google.com/store/apps/details?id=kr.hwangti.batterylog`

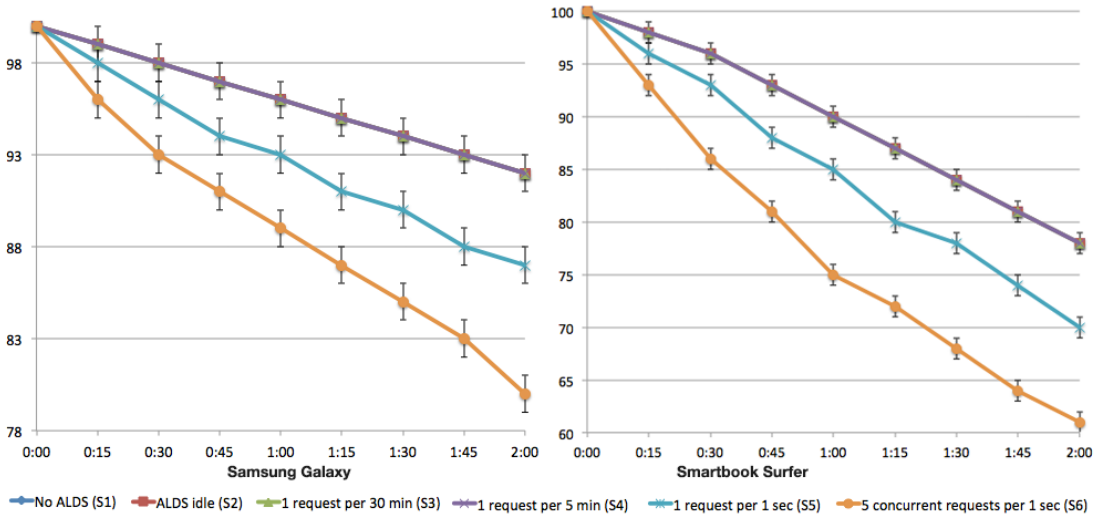


Figure 6.3.: ALDS power consumption in terms of battery charge percentage (y-axis) over time (x-axis).

GB SD card, 512 MB RAM and 3300 mAh Li-Ion battery. The device was not used before performing evaluation, which means that battery was completely new at the time of evaluation.

Power consumption and average response time were measured in the following six scenarios:

- **S1:** Device without ALDS
- **S2:** Device with ALDS in stand-by mode
- **S3:** Device with ALDS with 1 request every 30 minutes
- **S4:** Device with ALDS with 1 request every 5 minutes
- **S5:** Device with ALDS with 1 request every 1 second
- **S6:** Device with ALDS with 5 concurrent requests per 1 second

For each scenario we performed all measurements 5 times in order to average out power consumption variation (e.g. due to background OS processes or network overhead activity).

Benchmarking results

The results of the power consumption evaluation are show in Figure 6.3. As it can be seen, ALDS in idle state (S2) as well as periodic requests with 30 minute (S3) and 5 minute (S4) frequency have no observable impact on battery consumption.

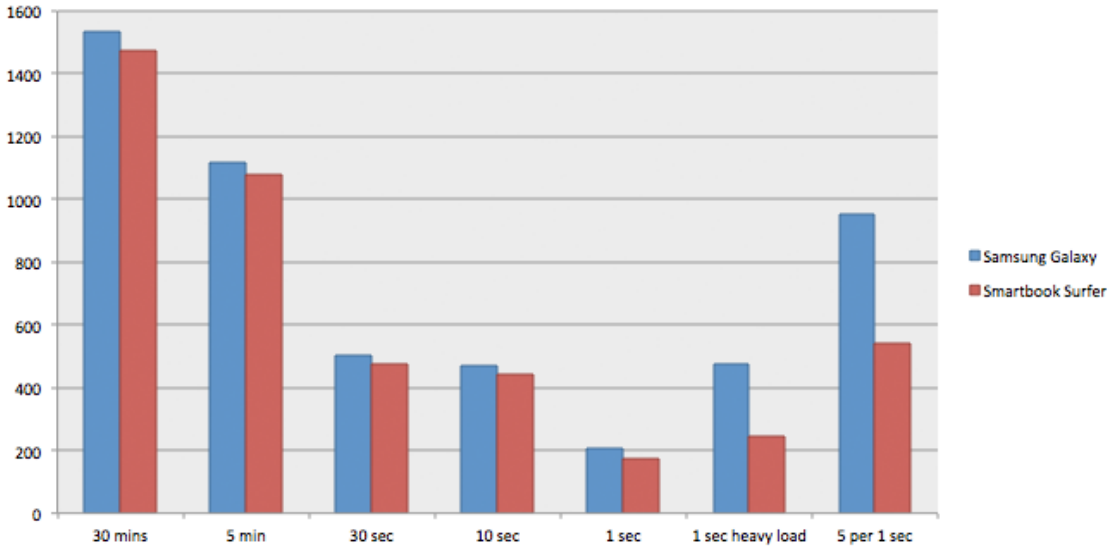


Figure 6.4.: Average ALDS response time per linked data access (in ms).

Periodic requests with 1 second frequency (S5) have a small impact. The difference to S1 (no ALDS installed) is in range of 3-7% points of the charge. Five concurrent requests per 1 second (S6) have the worst impact on power consumption. The difference to S1 (no ADLS installed) is in range of 10-14% point of the charge.

Due to bugs encountered in the Androjena library, in the scenario with 5 concurrent requests per second (S6), ALDS has stopped sending proper responses on average 2.3 times per evaluation duration. At that point, the service had to be restarted and the evaluation started from the beginning. Because it was impossible to go through the whole 2 hour evaluation cycle, we used two or more iterations to go through the complete 2 hours timespan. Our observation shows that a possible cause of such behavior could be due to Androjena not freeing resources properly (or enough) after usage thus exceeding the limited Android VM heap size (32 MB for devices used in evaluation) and causing the library to stop functioning.

Because measuring only power consumption by ALDS was not possible or the difference with already existing evaluation scenarios was insignificant, we measured only the average response times in the following additional scenarios:

- **S7:** Device with ALDS with 1 request every 30 seconds
- **S8:** Device with ALDS with 1 request every 10 seconds
- **S9:** Device with heavy front end load and ALDS with 1 request every 1 second

The results of response time evaluation are shown in Figure 6.4. As can be seen, ALDS response time differs depending on request frequency. In the scenario with one request per 30 minutes (S3) the average response time for the Samsung

Galaxy is 1,535 ms and for the Smartbook Surfer 1,474 ms. This long response time is caused by Android suspending ALDS and the third-party content provider to save power. Most of the response time can be attributed to waking up those components from suspension mode.

In the scenario with one request per 5 minutes (S4) the average response time for the Samsung Galaxy is 1,116 ms and for the Smartbook Surfer 1,078 ms. Also here, the response time is high due to suspending ALDS and the third-party content provider. However, since requests are performed more frequently, ALDS is not suspended completely, but only partially (i.e. web server is still running, but the query component is woken up upon requests).

In the scenarios with one request per 30 seconds (S7) and 10 seconds (S8) the average response times for the Samsung Galaxy are 502 ms and 472 ms respectively, for the Smartbook Surfer 473 ms and 442 ms respectively. In these cases, ADLS is not suspended at all, while content providers can be suspended completely or partially depending on their architecture. For example, if a content provider stores parts of the data using different storage containers, one of the routes can be suspended.

In the scenario with one request per 1 second (S5) the average response time for the Samsung Galaxy is 205 ms and for the Smartbook Surfer 176 ms. This case is optimal in terms of response time, since ALDS and third-party content providers are not being suspended at all due to frequent requests.

In the scenario with one request per 1 second with heavy front end load (S9) the average response time for the Samsung Galaxy is 478 ms and for the Smartbook Surfer 245 ms. During the heavy front end load scenario, ALDS was running in background, while the device itself was used to navigate through a series of memory, CPU and bandwidth intensive applications. Applications include: Instagram (browsing images), Pinterest (browsing images), Facebook (browsing feeds, photos, using chat), Play Store (browsing and applying updates to installed apps), YouTube (watching videos), Cut The Rope: Time Travel (playing game). During the benchmark there was from a user perspective no noticeable impact observable on active application performance, application switching, network speed or other features. The response time increased in comparison to one request per 1 second case by approximately a factor 2.5 for the Samsung Galaxy and by approximately a factor 1.4 for the Smartbook Surfer. This is due to Android prioritizing applications running in foreground compared to background services. The difference between response times between the Samsung Galaxy and the Smartbook Surfer can be explained by the dualcore processor of the Smartbook Surfer that allows better handling of multitasking.

In the scenario with five concurrent requests per 1 second (S6) the average response time for the Samsung Galaxy is 952 ms and for the Smartbook Surfer 542 ms. The increase in response time in this case is caused by Androjena conversion delay, meaning that Androjena cannot process transformation of structured data into RDF fast enough. This might as well be related to the heap size issue mentioned before. The difference between response times of Samsung Galaxy

and Smartbook Surfer can again be explained by the dualcore processor of the Smartbook Surfer that allows Androjena to perform the conversion slightly faster.

As our evaluation shows, the ALDS impact on device power consumption while answering infrequent requests (requests every 30 and 5 min, DSSN use case) can be considered insignificant. On other hand, if ALDS is going to be used intensively, an external power source would be required after several hours of work.

6.1.5. Conclusion

In this section, we presented an approach for equipping embedded and smart devices with Linked Data interfaces. Our approach is based on mapping structured data hosted on the device to RDF and exposing this data as Linked Data using an embedded webserver. Our implementation is currently implemented for Linux and contains some Android-specifics (i.e. using content providers). However, it is easily transferable to other embedded platforms. Android's content provider, for example, are simple tabular SQLite tables, which can be used in a similar fashion on other systems.

We also showed with a newly developed benchmark methodology, that device power consumption does not increase significantly until Linked Data is retrieved frequently (<1 request per second). We argue that this resource demands can be accommodated by most mobile use cases and power consumption is not an issue for stationary smart devices (e.g. TVs). We expect Android and similar OS to be deployed on more and more smart devices ranging from watches, smartphones, routers, devices with displays (e.g. refrigerators), cash registers and many other device categories currently not even yet available. As a result, the Web and Internet being accessed from desktop computers will loose importance compared to these novel usage scenarios. With ALDS, we made a first step towards extending the Web of Data towards a Data Internet of Things, which comprises these scenarios.

6.2. Hybrid provider approach

In this section, we describe *Hybrid Linked Data Server* approach that was developed using hybrid provider approach.

The section is structured as follows: We introduce the hybrid approach for providing linked data for ubiquitous devices with limited data connection in subsection 6.2.1. We describe the approach for adding a hybrid linked data interface to smart devices in subsection 6.2.2. Finally, we conclude in subsection 6.2.5 with an outlook on future work.

6.2.1. Introduction

As it was discussed in subsection 6.1.1, the unique identification can be realized using barcodes, RFIDs [Wang et al., 2006] or embedded systems and smart internet-

enabled devices. In the former two cases the object itself can only identify itself and a virtual representation has to be hosted elsewhere. Some of the existing devices can not only identify but also describe themselves by providing comprehensive information. However, there is a set of devices that might not have data connection available at all times. Person driving into tunnel with his smartphone or an autonomous robot digging deeper into the wreckage could be examples when connection is unavailable.

In this section, we present an approach for equipping embedded and smart devices with a Linked Data interface capable of providing data even when the device itself is offline. The approach is based on replicating existing semantic data on the device to external server (or a group of servers) and exposing this data as dereferencable RDF. The technical architecture comprises an ELDS running on the device as presented earlier in subsection 6.1.2, dedicated online server that serves as copy of the device and replication agent that ensures consistency of the data provided by the server. As a result, all smart devices can easily provide standardized structured information and become first class citizens on the Data Web even in poor networking conditions or hazardous environments.

As always, specific requirement when dealing with smart and embedded devices are resource constraints. In this case there are two constrainst:

- *Power consumption* is a key aspect, when equipping devices with additional functionality.
- *Size of the data* during replication is important as well, since the device itself might have poor or limite connection at any time.

Hence, a particular focus of our approach is limitation of the impact on power consumption as well as minimization of data transfer.

6.2.2. Approach: Hybrid Linked Data Server

The main goal of our approach is to enable any smart device (e.g. smart phones, robots) to identify and describe itself by providing comprehensive information, in accordance with the Linked Data principles, even during poor connectivity or absence of such. As a way to accomplish this goal, we propose the concept of an *Hybrid Linked Data Server* (HLDS) that comprises of an ELDS, which hosts Web-accessible RDF information as Linked Data directly from within the smart device. The HLDS concept is partially based on ELDS presented in section 6.1 and partially on replication approach presented in section 5.1.

Architecture

The overall HLDS architecture is shown in Figure 6.5 and includes four layers:

1. *User layer* – comprises a user client,

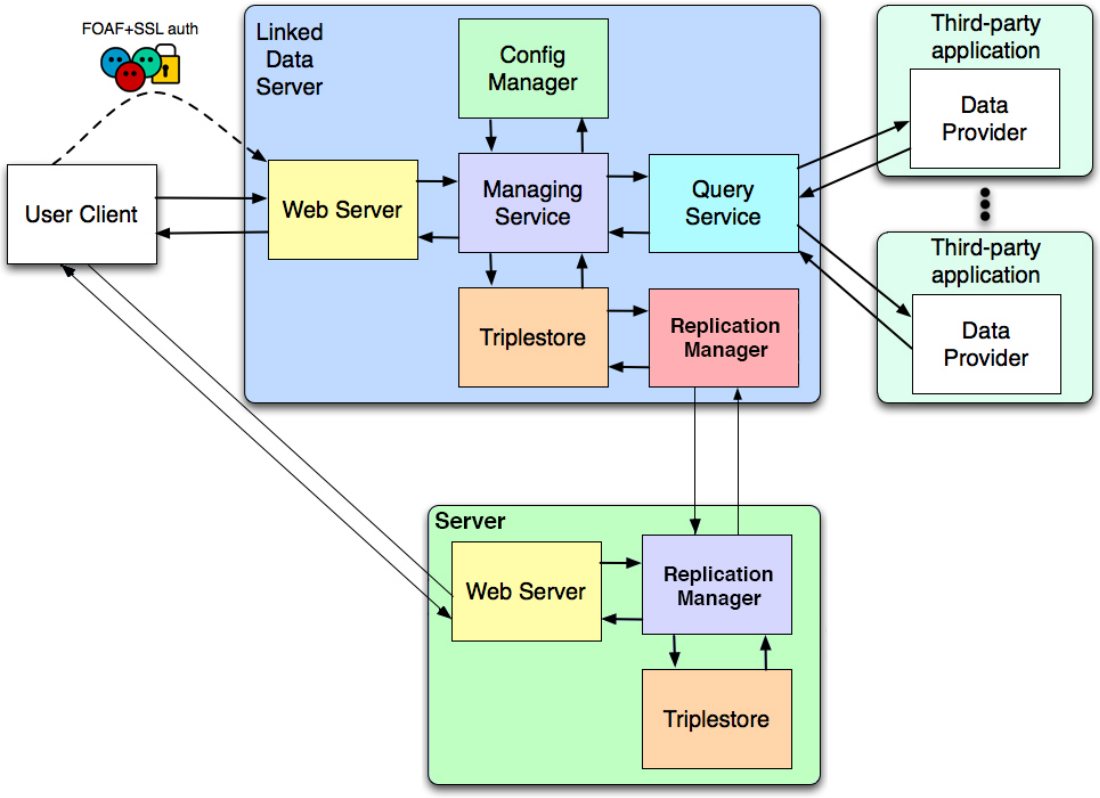


Figure 6.5.: Overview of the Hybrid Linked Data Server architecture.

2. *ELDS layer* – comprises ELDS internal components,
3. *Replication layer* – comprises replication component,
4. *Server layer* – comprises third-party server.

The separation in these four layers not only facilitates modularity and maintainability, but also aims at minimizing the impact on power consumption and data availability. Components of the replication layer, for example, can easily be suspended thus minimizing the main memory and power consumption requirements, while server layer is able to provide data even when ubiquitous device itself is not available.

The *User layer* includes user client application that can process RDF data and optionally a user WebID that is used for authentication and access control to the data provided by ELDS. *ELDS layer* includes all of the ELDS internal components as described in subsection 6.1.2. *Replication layer* includes data managing component which ensures that third-party server has newest possible replica of the data provided by the device. *Server layer* includes third-party server capable of exposing data as dereferencable RDF.

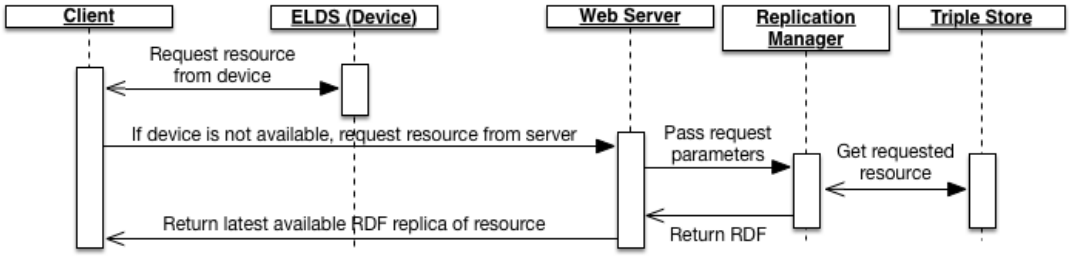


Figure 6.6.: HLDS workflow during Linked Data access.

The basic HLDS workflow during Linked Data access is shown on Figure 6.6 and consists of the following steps:

1. The client requests a specific resource description from HLDS on the smart device using an URI.
2. If HLDS on device is accessible through data connection, workflow follows ELDS steps that was described in subsection 6.1.2.
3. If HLDS on device is not accessible through data connection, client will poll the HLDS server with replicated data.
4. If HLDS server requires authentication and enforces access control, the client must complete a FOAF+SSL authentication procedure (optional).
5. The Replication Manager on the server identifies configuration ID and fetches the required RDF from Triple Store.
6. The Web Server component returns resulting the RDF serialization back to client.

6.2.3. Use cases

There are a number of potential use cases for HLDS. Examples include (a) retrieving data from an HLDS-based robotic unit that might not always have data connection, (b) exposing contact data in a distributed social network without need for provider to always have data connection. We briefly outline given use cases in the sequel.

The first use case where HLDS fits perfectly is using it to provide access to the data from robotic units that might be unavailable at times. For example, Mars rover that has very limited data connection capabilities can use HLDS to push new data to server once the connection is available. Replication managers on both - client and server - would allow lossless synchronisation even on the slowest and worst data connections.

The second use case is using HLDS as a part of Distributed Semantic Social Network (DSSN) [Tramp et al., 2011c]. Including HLDS in the DSSN architecture allows easy access to device internal data in a standardized non-proprietary way, e.g. location information or address book. Enforcing WebID authorization will help protect the data from unwanted access by third parties. This would enable usage scenarios, where a user would not need to contact his friend to ask for her location, but could just poll this information directly from her friend's smart device without the need for a central service. In addition to these features of ELDS, HLDS server allows access to this information even when the client does not have active data connection.

6.2.4. Implementation

Android specific version of HLDS could be used as an example of implementation – *Hybrid Android Linked Data Server* (HALDS). It is possible to use existing Android implementation of ELDS as a core for HALDS. Then the only thing that would be required is adding a replication manager which can handle synchronisation of data between Android device and the server.

The server side, however, would require implementation of all the components since ELDS approach was used purely on the client side. That means that the server needs to run a triple-store that can store and process triples. It is possible to use existing ones, e.g. Virtuoso[9]. The replication manager will have to be created from scratch the same way as was done for the client. It is also necessary to keep in mind that both replication managers - on the server and on the client - should be created in a way that would allow them to easily communicate with each other. We suggest to use existing means for this (e.g. HTTP protocol for data transfer, diff approach for merging data etc.).

6.2.5. Conclusion

In this section, we presented an approach for equipping embedded and smart devices with a Linked Data interface capable of providing data even when the device itself is offline. Our approach is based on combining ELDS with data managing component capable of replicating existing semantic data from the ELDS to external server (or a group of servers). With HLDS we made a first step towards extending the Web of Data towards a Data Internet of Things, which comprises different scenarios with possible connectivity issues.

[9] `http://virtuoso.openlinksw.com/`

7. Conclusions and Future Work

This chapter summarizes our research work, highlights our main contributions, and gives the general conclusion over the work. It then pinpoints the future directions in which we intend to move further to extend and broaden the research conducted in these areas.

7.1. Conclusions

The aim of this thesis was to present and evaluate an approaches for bringing the Semantic Web to ubiquitous devices. We have focused on two different areas client and provider approaches. To see how different client approaches perform, we have developed several ubiquitous applications. The provider approaches have been evaluated in the same manner. Overall, we have shown that three different client approaches can be used to bring the Semantic Web to the ubiquitous devices. We have as well shown that two different provider approaches can be used to expose the data from the ubiquitous devices into the Semantic Web. Each of the approaches has it's own set of advantages and downsides and works best in a specific use cases. In each of the following subsections, we discuss the each of the researched approaches in detail.

7.1.1. Thin client approach

To evaluate the thin client approach and address challenges discussed in section 1.2.1 and subsection 1.2.3, we have developed an OntoWiki Mobile application. With OntoWiki Mobile, we have undertaken one crucial aspect – the provisioning of a comprehensive knowledge management tool for mobile use. We have used the new HTML5 application cache functionality to support offline work. Advanced conflict resolution features was built in to improve offline capabilities. We also demonstrated that a comprehensive semantic collaboration platform is possible to implement for mobile devices with minimal requirements based on recent Web standards. Due to its general purpose architecture, OntoWiki Mobile is particularly suited to support the long tail of domain-specific mobile applications, for which the development of individual tools would not be (economically) feasible.

It should also be mentioned that due to the ubiquitous nature of OntoWiki Mobile, it has a number of limitations. The biggest limitation is that the possibility of adding new features almost completely depends on advances in HTML5 development and support of new HTML5 APIs on variety of devices. It should

also be noted that while OntoWiki Mobile can run on older device, some functions might be limited or disabled due to the previously mentioned issue.

7.1.2. Hybrid client approach

To evaluate the hybrid client approach and address challenges discussed in section 1.2.1 and subsection 1.2.3, we have developed a Mobile DSSN Client application. Work done on Mobile DSSN Client is a crucial piece in the medium-term agenda of realizing a truly distributed social network based on semantic technologies. Our realization focused on providing an extensible framework for social semantic networking with truly ubiquitous approach of implementing core features in HTML5 and relying on platform specific APIs only when required. We have done an example implementation for the Android platform demoing both - generic and platform specific features. With this work we showcased how different (social) Semantic Web standards, technologies and best practices can be integrated into a comprehensive architecture for social networking and adopted to work on variety of ubiquitous devices.

Even though hybrid client approach used for Mobile DSSN Client is more flexible than aforementioned thin client approach, it has some limitations. One of the biggest disadvantages is that even though this approach provides a way to access platform specific APIs, it is required to build a platform specific version of the application. That could require much longer periods of time for development in comparison to thin client approach.

7.1.3. Fat client approach

To evaluate the fat client approach and address challenges discussed in section 1.2.1, we have developed an MSSW application. Work done on MSSW is a further crucial piece (in addition to Mobile DSS Client) in the medium-term agenda of realizing a truly distributed social network based on semantic technologies. Our realization focused on providing a extensible framework for social semantic networking for one specific platform (Android) while utilizing as many platform specific APIs as possible to enhance the work of the application. With MSSW, we showcased how different (social) Semantic Web standards, technologies and best practices can be integrated into a comprehensive architecture for social networking on mobile devices for one specific platform.

Even though fat client approach used for MSSW is the most flexible in comparison to thin or hybrid client approaches, it has a greater number limitations than both of aforementioned approaches. The biggest disadvantages is that while using this approach, the developer creates a platform specific version of the application which sometimes is nearly impossible to directly port to another platform because of the differences in SDKs or even programming languages (e.g. Objective C used on iOS vs. Java used on Android). Additionally, it should be noted that using this

approach requires much deeper understanding of the target platform, it's APIs and specifics (e.g. memory management on Android).

7.1.4. Fat provider approach

To evaluate the fat provider approach and address challenges discussed in section 1.2.1, we have developed an ALDS application. Using ALDS we have presented an approach for equipping embedded and smart devices with Linked Data interfaces. The approach is based on exposing the structured data hosted on the device as Linked Data using an embedded webserver by utilizing the data mappings to RDF. We presented an implementation for Linux that contains some Android-specific parts (i.e. using android content providers). We argue that this resource demands can be accommodated by most mobile use cases and power consumption is not an issue for stationary smart devices (e.g. TVs). With ELDS we made a first step towards extending the Web of Data towards a Data Internet of Things.

Our evaluation has shown that this approach has a set of limitations. One of the most important limitations is the fact that ubiquitous device equipped with ELDS must be accessible from the client computer. That might not always be the case because, for example, cell operators tend to put devices that use data connection behind proxies to limit number of IPv4 addresses that are given out to clients. That thought should change in time with migration to IPv6 addresses. One more limitation is again related to data connection, this time to its quality and stability. The ELDS should have a stable and persistent data connection to be able to provide that data to client which might not always be the case for ubiquitous devices.

7.1.5. Hybrid provider approach

We presented an approach for equipping embedded and smart devices with a hybrid Linked Data interface capable of providing data even when the device itself is offline. The approach is based on combining aforementioned ELDS approach with a data managing component capable of replicating existing semantic data from the ELDS to external server (or a group of servers). That can provide this data to clients even when the original data provider is offline. With HLDS we made a first step towards extending the Web of Data towards a Data Internet of Things, which comprises different scenarios with possible connectivity issues.

Even though HLDS approach is devoid of the limitation related to data connection presence mentioned in the ELDS approach, there are still some other limitations present. First, the limitation related to connectivity (same network or IPv6 address) is still present. In addition, to implement this approach it is required to have a dedicated server (or a group of servers) which is installed, available online and properly configured. Additionally, there is a fact that client must know the address of the server with replica beforehand, otherwise the client will not be able to get the data (or location of that server) when the ELDS provider is offline.

7.2. Directions for Future Work

Each research area has its own direction(s), in which we can move further and expand the work.

7.2.1. Thin client approach

Future work for the thin client approach in general and OntoWiki Mobile in particular will focus on representation of provenance and use of the mobile device's sensors for context-aware knowledge base exploration. With regard to the replication, we plan to develop a rule-based approach for the selection of knowledge base parts to replicate on the mobile device. The approach will take mobile context information (such as the time, location) as well as usage patterns (e. g. browsing history) and manually supplied user preferences into account.

7.2.2. Hybrid client approach

Future work for the hybrid client approach in general and Mobile DSSN Client in particular will focus on further decreasing the entrance barrier for ordinary users. A current obstacle is that users are required to have a WebID and - if they want to use authentication and access control features - a FOAF+SSL enabled WebID. In particular, creating a FOAF+SSL enabled WebID is, due to the certificate creation, still a cumbersome process. A possible simplification of this process would be to enable mobile phone users to create and upload the required profile and certificates directly from their mobile device.

A further important aspect to be developed is the standardization and realization of social networking applications, which seamlessly integrate with and run on top of the distributed social semantic network. Such applications would comprise everything we know from centralized social networks (e.g. games, travel, quizzes etc.), but would make use of FOAF+SSL and the other distributed social networking components for authentication, access control, subscription/notification etc.

7.2.3. Fat client approach

Future work for the fat client approach in general and MSSW in particular will focus on further decreasing the entrance barrier for ordinary users. The reason is very similar to the one mentioned in subsection 7.2.2 users are required to have a WebID and a FOAF+SSL enabled WebID. A possible simplification of this process would be to enable mobile phone users to create, upload and expose the required profile and certificates directly from their mobile device.

We also plan to implement a more efficient and user-friendly way for subscribing to updates of contacts. These will include profile changes, status updates, (micro-)blog posts as well as updates retrieved from social networking apps. This feature would be facilitated by a proxy infrastructure, which caches updates until the

device re-connects to the network after a period of absence (e.g. due to limited network connection or switched-off devices).

A further important aspect to be developed is the standardization and realization of social networking applications, which seamlessly integrate with and run on top of the distributed social semantic network. Such applications would comprise everything we know from centralized social networks (e.g. games, travel, quizzes etc.), but would make use of FOAF+SSL and the other distributed social networking components for authentication, access control, subscription/notification etc.

7.2.4. Fat provider approach

Future work for the fat provider approach in general and ALDS in particular will focus on extending our approach along several dimensions. First, we plan to implement smart caching for data fetched from a content provider, so that subsequent requests do not require re-retrieving and processing data from a content provider. Then, we aim to support complex configurations for queries across several content providers and allow the execution of SPARQL queries. In order to improve availability we plan to realize a hybrid provider approach, where the server has access to client data and responds if the client is not available.

7.2.5. Hybrid provider approach

Since this approach was not yet implemented and evaluated, the primary aim for the future work is to actually do an example implementation and evaluate it using same approach as for ELDS. It is also interesting to tackle the issue with location of the replica server. Currently, the client must first request this location from the ELDS otherwise the client will not be able to connect to it once ELDS is not accessible. Finding a way to remove this requirement would be an additional plan for future work.

A. Curriculum Vitae

Timofey Ermilov

Hallesche Str. 207
04159 Leipzig, Germany.
Phone: (+49) 17632092245
Email: ermilov@informatik.uni-leipzig.de

Personal Data

Birth date: January 1st, 1987
Birth place: Murmansk, Russia
Nationality: Russian
Marital status: Married

Education

2010 – Present
University of Leipzig (Leipzig, Germany)
Ph.D., Faculty of Mathematics and Computer Science, Department of Computer Science.
Thesis title: **Ubiquitous Semantic Applications**.

2004 – 2009
Saint-Petersburg University of Telecommunications (Saint-Petersburg, Russia)
Dipl.-Ing., Faculty of Multichannel Telecommunications Systems.
Thesis title: **Computer knowledge testing system**.

2001 – 2004
Saint-Petersburg University of Telecommunications Lyceum (Saint-Petersburg, Russia)
Secondary Education.

Research Interests

- Semantic Web.
- Ubiquitous devices.
- Human-Computer Interaction.

Publications

1. **Timofey Ermilov**, Ali Khalili, Sören Auer
   "Ubiquitous Semantic Applications: A Systematic Literature Review". In International Journal On Semantic Web and Information Systems, 2014.

2. **Timofey Ermilov**, Sören Auer
   "Enabling Linked Data access to the Internet of Things". At International Conference on Information Integration and Web-based Applications Services, 2013.

3. Jens Lehmann, Quan Nguyen, and **Timofey Ermilov**
   "Can we Create Better Links by Playing Games?". At 7th IEEE International Conference on Semantic Computing, 2013.

4. **Timofey Ermilov**, Sebastian Tramp and Sören Auer "A Mobile Client for the Distributed Semantic Social Network". At Knowledge Engineering and Semantic Web Conference, 2012.

5. Sebastian Tramp (geb. Dietzold), Philipp Frischmuth, **Timofey Ermilov**, Saeedeh Shekarpour, and Sören Auer.
   "An Architecture of a Distributed Semantic Social Network ". In Semantic Web Journal, 2012.

6. Sebastian Tramp (geb. Dietzold), **Timofey Ermilov**, Philipp Frischmuth, and Sören Auer.
   "Architecture of a Distributed Semantic Social Network". At the Federated Social Web Europe 2011, Berlin June 3rd-5th 2011.

7. Michael Martin, Daniel Gerber, Norman Heino, Sören Auer, and **Timofey Ermilov**.
   "Managing Multimodal and Multilingual Semantic Content". In Proceedings of the 7th International Conference on Web Information Systems and Technologies, 2011.

8. **Timofey Ermilov**, Norman Heino, Sebastian Tramp, and Sören Auer.
"OntoWiki Mobile — Knowledge Management in your Pocket". In Proceedings of the ESWC2011, 2011.

9. Amrapali Zaveri, Ricardo Pietrobon, Sören Auer, Jens Lehmann, Michael Martin, and **Timofey Ermilov**.
"ReDD-Observatory: Using the Web of Data for Evaluating the Research-Disease Disparity", In Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence, 2011.

10. Sebastian Tramp, Philipp Frischmuth, Natanael Arndt, **Timofey Ermilov**, and Sören Auer.
"Weaving a Distributed, Semantic Social Network for Mobile Users", In Proceedings of the ESWC2011, 2011.

11. Andreas Thalhammer, **Timofey Ermilov**, Katariina Nyberg, Ario Santoso, John Domingue.
"MovieGoer – Semantic Social Recommendations and Personalised Location-Based Offers", In Proceedings of ISWC2011, October 2011.

12. Sebastian Tramp, Philipp Frischmuth, **Timofey Ermilov**, and Sören Auer.
"Weaving a Social Data Web with Semantic Pingback", In Proceedings of the EKAW 2010, October 2010.

13. Amrapali Zaveri, Ricardo Pietrobon, **Timofey Ermilov**, Michael Martin, Norman Heino, and Sören Auer.
"Evaluating the disparity between active areas of biomedical research and the global burden of disease employing Linked Data and data-driven discovery", at OBML 2010 Workshop Proceedings, IMISE Report, Mannheim, IMISE, September 2010.

Languages Skills

- Russian: Native proficiency.
- English: Full professional proficiency.
- German: Elementary proficiency.

Technical and Programming Skills

- **Programming Languages Skills:**
  - PHP, 6 years of experience.
  - Javascript, 6 years of experience.
  - C# / Mono, 5 years of experience.
  - Java, 5 years of experience.

  - ActionScript 3, 4 years of experience.
  - Python, 3 years of experience.
  - Objective C, 3 years of experience.
  - Ruby, 2 years of experience.
  - C / C++, 2 years of experience.

- **Database Systems:**
  - MySQL.
  - MongoDB.
  - Virtuoso.

Projects

- **Mobile Social Semantic Web:**
  An Android-based social web client as well as a contacts provider, which integrates your distributed FOAF/WebID social network into your mobile phone. It is available at `http://aksw.org/Projects/MobileSocialSemanticWeb.html`. Implemented in Java and Android SDK.

- **OntoWiki Mobile:**
  OntoWiki Mobile was developed to address the need for a mobile web application for rapid and simple knowledge acquisition in a collaborative way. It allows users to collect instance data, refine the structure of knowledge bases and browse data using hierarchical or faceted navigation on-the-go even without a present data connection. It is available at `http://aksw.org/Projects/OntoWikiMobile.html`. Implemented in PHP and OntoWiki framework.

- **Android Linked Data Server (ALDS):**
  Linked Data Server for Android used to expose the data from device in RDF format. It is available at `https://github.com/AKSW/ALDS`. Implemented in Java and Android SDK.

- **Clerkd:**
  iOS location-based social network for music sharing and discovery. It is available at `http://clerkd.com/`. Implemented in C (client) and Python (server).

- **Mielophone:**
  Simple music search engine that mashes up together services like MusicBrainz, It is available at `https://github.com/mielophone/`. Implemented in AS3 and Flex SDK.

List of Tables

List of Figures

Bibliography

[Aranda-Corral et al., 2009] Aranda-Corral, G. A., Borrego-Díaz, J., and Gómez-Marín, F. (2009). Toward semantic mobile web 2.0 through multiagent systems. In *Proceedings of the Third KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, KES-AMSTA '09, pages 400–409, Berlin, Heidelberg. Springer-Verlag.

[Atzori et al., 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787–2805.

[Auer et al., 2006a] Auer, S., Dietzold, S., and Riechert, T. (2006a). Ontowiki – a tool for social, semantic collaboration. In Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., and Aroyo, L., editors, *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer Berlin / Heidelberg.

[Auer et al., 2006b] Auer, S., Dietzold, S., and Riechert, T. (2006b). OntoWiki - A Tool for Social, Semantic Collaboration. In Cruz, I. F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., and Aroyo, L., editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer.

[Auer et al., 2009] Auer, S., Lehmann, J., and Hellmann, S. (2009). Linkedgeodata: Adding a spatial dimension to the web of data. In Bernstein, A., Karger, D., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., and Thirunarayan, K., editors, *The Semantic Web - ISWC 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 731–746. Springer Berlin / Heidelberg.

[Balasubramanian et al., 2009] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. (2009). Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293. ACM.

[Bechhofer et al., 2004] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). OWL Web Ontology Language Reference. Technical report, W3C, http://www.w3.org/TR/owl-ref/.

[Becker and Bizer, 2009] Becker, C. and Bizer, C. (2009). Exploring the Geospatial Semantic Web with DBpedia Mobile. *J. Web Sem.*, 7(4):278–286.

[Beckett, 2004] Beckett, D. (2004). RDF/XML syntax specification (revised). W3C recommendation, W3C.

[Bellavista et al., 2012] Bellavista, P., Corradi, A., Fanelli, M., and Foschini, L. (2012). A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys (CSUR)*, 44(4):24.

[Bellini et al., 2012] Bellini, P., Bruno, I., Cenni, D., Fuzier, A., Nesi, P., and Paolucci, M. (2012). Mobile medicine: semantic computing management for health care applications on desktop and mobile devices. *Multimedia Tools and Applications*, 58(1):41–79.

[Berners-Lee, 2010] Berners-Lee, T. (2010). Long Live the Web. *Scientific American*.

[Berners-Lee, 2011] Berners-Lee, T. (2011). Linked Data - Design Issues. website. last change: 2009/06/18; retrieved: 2011/07/25.

[Berners-Lee and Connolly, 2011] Berners-Lee, T. and Connolly, D. (2011). Notation3 (N3): A readable RDF syntax. Technical report, W3C.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

[Bishop et al., 2011] Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., and Velkov, R. (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1):1–10.

[Braun et al., 2010] Braun, M., Scherp, A., and Staab, S. (2010). *Collaborative creation of semantic points of interest as linked data on the mobile phone*. Arbeitsberichte aus dem Fachbereich Informatik. Inst. WeST.

[Breslin et al., 2006] Breslin, J. G., Decker, S., Harth, A., and Bojars, U. (2006). SIOC: an approach to connect web-based communities. *International Journal of Web Based Communities*, 2(2):133–142.

[Brickley and Guha, 2004] Brickley, D. and Guha, R. V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, W3C.

[Brickley and Miller, 2004] Brickley, D. and Miller, L. (2004). FOAF Vocabulary Specification. Namespace Document 2 Sept 2004, FOAF Project. http://xmlns.com/foaf/0.1/.

[Broekstra et al., 2002] Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. In *ISWC*, number 2342 in LNCS, pages 54–68. Springer.

[Cano et al., 2012] Cano, A.-E., Dadzie, A.-S., Uren, V., and Ciravegna, F. (2012). Sensing presence (presense) ontology: User modelling in the semantic sensor web. In *The Semantic Web: ESWC 2011 Workshops*, pages 253–268. Springer.

[Celino et al., 2011] Celino, I., Dell'Aglio, D., Valle, E., Balduini, M., Huang, Y., Lee, T., Kim, S.-H., and Tresp, V. (2011). Bottari: Location based social media analysis with semantic web. ISWC.

[Charland and Leroux, 2011] Charland, A. and Leroux, B. (2011). Mobile application development: web vs. native. *Commun. ACM*, 54(5):49–53.

[Chen and Babar, 2011] Chen, L. and Babar, M. A. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Information & Software Technology*, 53(4):344–362.

[Chen et al., 2010] Chen, Y.-S., Chang, W.-H., Fang, H.-M., Yeh, Y.-M., and Cheng, R.-S. (2010). A context-aware reasoning framework with owl for mobile web information acquisition. *Journal Of Internet Technology*, 11(2):203–213.

[Clark et al., 2008] Clark, K. G., Feigenbaum, L., and Torres, E. (2008). SPARQL Protocol for RDF. World Wide Web Consortium, Recommendation REC-rdf-sparql-protocol-20080115.

[Costabello et al., 2012] Costabello, L., Villata, S., Delaforge, N., Gandon, F., et al. (2012). Linked data access goes mobile: Context-aware authorization for graph stores. In *LDOW-5th WWW Workshop on Linked Data on the Web-2012*.

[Cyganiak and Bizer, 2011] Cyganiak, R. and Bizer, C. (2011). Pubby-a linked data frontend for sparql endpoints. Retrieved September 20, 2011.

[Dave and Berners-Lee, 2011] Dave, D. and Berners-Lee, T. (2011). Turtle - Terse RDF Triple Language. Technical report, W3C.

[Dietze et al., 2009] Dietze, S., Gugliotta, A., and Domingue, J. (2009). Bridging the gap between mobile application contexts and semantic web resources. In Stojanovic, D., editor, *Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications*, Premier Reference Source, pages 217–234. Information Science Reference.

[Ding et al., 2004] Ding, L., Finin, T. W., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V., and Sachs, J. (2004). Swoogle: a search and metadata engine for the semantic web. In Grossman, D. A., Gravano, L., Zhai, C., Herzog, O., and Evans, D. A., editors, *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 652–659. ACM.

[Dyba et al., 2007] Dyba, T., Dingsoyr, T., and Hanssen, G. K. (2007). Applying systematic reviews to diverse study types: An experience report. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ESEM '07, pages 225–234, Washington, DC, USA. IEEE Computer Society.

[Erling and Mikhailov, 2007] Erling, O. and Mikhailov, I. (2007). RDF support in the virtuoso DBMS. In Auer, S., Bizer, C., Müller, C., and Zhdanova, A. V., editors, *CSSW*, volume 113 of *LNI*, pages 59–68. GI.

[Ermilov and Auer, 2013] Ermilov, T. and Auer, S. (2013). Enabling linked data access to the internet of things.

[Ermilov et al., 2011a] Ermilov, T., Heino, N., and Auer, S. (2011a). Ontowiki mobile: knowledge management in your pocket. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 33–34, New York, NY, USA. ACM.

[Ermilov et al., 2011b] Ermilov, T., Heino, N., Tramp, S., and Auer, S. (2011b). OntoWiki Mobile — Knowledge Management in your Pocket. In *Proceedings of the ESWC2011*.

[Ermilov et al., 2014] Ermilov, T., Khalili, A., and Auer, S. (2014). Ubiquitous semantic applications: A systematic literature review. *International Journal On Semantic Web and Information Systems*, 10.

[Ermilov et al., 2012] Ermilov, T., Tramp, S., and Auer, S. (2012). A mobile client for the distributed semantic social network.

[Farooq Ali et al., 2005] Farooq Ali, M., Pérez-quiñones, M. A., and Abrams, M. (2005). *Building Multi-Platform User Interfaces with UIML*, chapter 22, pages 93–118. John Wiley and Sons, Ltd.

[Fenton, 2006] Fenton, A. (2006). Weft qda user's manual. *http://www. pressure. to/qda/doc/. Acesso em*, 18(03):2009.

[Ferrucci et al., 2013] Ferrucci, D., Levas, A., Bagchi, S., Gondek, D., and Mueller, E. T. (2013). Watson: Beyond jeopardy! *Artificial Intelligence*, 199–200(0):93 – 105.

[Gartner, 2013] Gartner (2013). Worldwide mobile phone sales. website.

[Glaser and Strauss, 1967] Glaser, B. G. and Strauss, A. L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine de Gruyter, New York, NY.

[Grant and Beckett, 2004] Grant, J. and Beckett, D. (2004). RDF test cases. W3C recommendation, World Wide Web Consortium.

[Grimm et al., 2002] Grimm, M., Tazari, M.-R., and Balfanz, D. (2002). Towards a framework for mobile knowledge management. In *Proceedings of the 4th International Conference on Practical Aspects of Knowledge Management*, PAKM '02, pages 326–338, London, UK, UK. Springer-Verlag.

[Guinard and Trifa, 2009] Guinard, D. and Trifa, V. (2009). Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*.

[Gümüs et al., 2006] Gümüs, O., Kardas, G., Dikenelli, O., Erdur, R., and Önal, A. (2006). Smop: A semantic web and service driven information gathering environment for mobile platforms. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*, pages 927–940. Springer Berlin / Heidelberg.

[Hachey, 2011] Hachey, G. (2011). Semantic web user interface: A systematic survey. Master's thesis, Athabasca University.

[Heflin, 2004] Heflin, J. (2004). OWL Web Ontology Language Use Cases and Requirements. Technical report, W3C.

[Heino et al., 2009] Heino, N., Dietzold, S., Martin, M., and Auer, S. (2009). Developing Semantic Web Applications with the Ontowiki Framework. In Pellegrini, T., Auer, S., Tochtermann, K., and Schaffert, S., editors, *Networked Knowledge – Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 61–77. Springer, Berlin/Heidelberg.

[Hu and Moore, 2007] Hu, B. and Moore, P. (2007). "smartcontext": An ontology based context model for cooperative mobile learning. In Shen, W., Luo, J., Lin, Z., Barthès, J.-P., and Hao, Q., editors, *Computer Supported Cooperative Work in Design III*, volume 4402 of *Lecture Notes in Computer Science*, pages 717–726. Springer Berlin / Heidelberg.

[Hu et al., 2009] Hu, D. H., Dong, F., and Wang, C.-L. (2009). A semantic context management framework on mobile device. In *Proceedings of the 2009 International Conference on Embedded Software and Systems*, ICESS '09, pages 331–338, Washington, DC, USA. IEEE Computer Society.

[Iannella et al., 2010] Iannella, R., Halpin, H., Suda, B., and Walsh, N. (2010). Representing vCard Objects in RDF. W3c Member Submission, W3C.

[Index, 2013] Index, C. V. N. (2013). Global mobile data traffic forecast update, 2012–2017 http://www. cisco. com/en. *US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862. html (Son erişim: 5 Mayıs 2013)*.

[Kitchenham, 2004] Kitchenham, B. (2004). Procedures for performing systematic reviews. Technical report, Keele University and NICTA.

[Korpipää and Mäntyjärvi, 2003] Korpipää, P. and Mäntyjärvi, J. (2003). An ontology for mobile device sensor-based context awareness. In *Proceedings of the 4th international and interdisciplinary conference on Modeling and using context*, CONTEXT'03, pages 451–458, Berlin, Heidelberg. Springer-Verlag.

[Krohn et al., 2007] Krohn, M., Yip, A., Brodsky, M., Morris, R., and Walfish, M. (2007). A World Wide Web Without Walls. In *6th ACM Workshop on Hot Topics in Networking (Hotnets)*, Atlanta, GA.

[Lane et al., 2010] Lane, N. D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., and Campbell, A. T. (2010). A survey of mobile phone sensing. *Comm. Mag.*, 48(9):140–150.

[Langridge and Hickson, 2002] Langridge, S. and Hickson, I. (2002). Pingback 1.0. Technical report, `http://hixie.ch/specs/pingback/pingback`.

[Lauesen, 2005] Lauesen, S. (2005). *User Interface Design: A Software Engineering Perspective*. Addison Wesley.

[Lee et al., 2004] Lee, V., Schneider, H., and Schell, R. (2004). *Mobile Applications: Architecture, Design, and Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Liao et al., 2005] Liao, L., Xu, K., and Liao, S. S. (2005). Constructing intelligent and open mobile commerce using a semantic web approach. *J. Inf. Sci.*, 31(5):407–419.

[Martin and Auer, 2010] Martin, M. and Auer, S. (2010). Categorisation of semantic web applications. In *proceedings of the 4th International Conference on Advances in Semantic Processing (SEMAPRO2010) 25 October – 30 October, Florence, Italy*.

[Miles and M., 1994] Miles, M. B. and M., H. (1994). *Qualitative Data Analysis: An Expanded Sourcebook(2nd Edition)*. Sage Publications, Inc, 2nd edition.

[Minno and Palmisano, 2010] Minno, M. and Palmisano, D. (2010). *Atom Activity Streams RDF mapping*. NoTube Project. `http://xmlns.notu.be/aair/`.

[Motik et al., 2012] Motik, B., Horrocks, I., and Kim, S. M. (2012). Delta-reasoner: a semantic web reasoner for an intelligent mobile platform. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 63–72. ACM.

[Niemelä and Latvakoski, 2004] Niemelä, E. and Latvakoski, J. (2004). Survey of requirements and solutions for ubiquitous software. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, MUM '04, pages 71–78, New York, NY, USA. ACM.

[Ostuni et al., 2013] Ostuni, V. C., Gentile, G., Di Noia, T., Mirizzi, R., Romito, D., and Di Sciascio, E. (2013). Mobile movie recommendations with linked data. In *Availability, Reliability, and Security in Information Systems and HCI*, pages 400–415. Springer.

[Otto and Dietzold, 2007] Otto, S. and Dietzold, S. (2007). Caucasian Spiders – A faunistic Database on the spiders of the Caucasus – http://caucasus-spiders.info. *Newsl. Brit. Arachn. Soc.*, 108:14.

[Passant and Mendes, 2010] Passant, A. and Mendes, P. (2010). sparqlPuSH: Proactive notification of data updates in RDF stores using PubSubHubbub. In *SFSW2010*.

[Patel and Khuba, 2009] Patel, D. R. and Khuba, S. A. (2009). Realization of semantic atom blog. *CoRR*, abs/0912.3957.

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation, W3C.

[Rieß et al., 2010] Rieß, C., Heino, N., Tramp, S., and Auer, S. (2010). EvoPat – Pattern-Based Evolution and Refactoring of RDF Knowledge Bases. In *Proceedings of the 9th International Semantic Web Conference (ISWC2010)*, Lecture Notes in Computer Science (LNCS), Berlin/Heidelberg. Springer.

[Ringland and Scahill, 2003] Ringland, S. and Scahill, F. (2003). Multimodality — the future of the wireless user interface. *BT Technology Journal*, 21:181–191.

[Roto, 2006] Roto, V. (2006). Web browsing on mobile phones – characteristics of user experience. *Helsinki University of Technology*.

[Ruta et al., 2010a] Ruta, M., Scioscia, F., Di Sciascio, E., and Piscitelli, G. (2010a). Semantic-based geographical matchmaking in ubiquitous computing. In *The Fourth International Conference on Advances in Semantic Processing (SEMAPRO2010)*, pages 166–172. IARIA.

[Ruta et al., 2012] Ruta, M., Scioscia, F., Di Sciascio, E., and Piscitelli, G. (2012). Semantic matchmaking for location-aware ubiquitous resource discovery. *International Journal On Advances in Intelligent Systems*, 4(3 and 4):113–127.

[Ruta et al., 2010b] Ruta, M., Scioscia, F., and Sciascio, E. D. (2010b). Mobile semantic-based matchmaking: A fuzzy dl approach. In *ESWC (1)'10*, pages 16–30.

[Sakkopoulos, 2009] Sakkopoulos, E. (2009). Semantic technologies for mobile web and personalized ranking of mobile web search results. In Sicilia, M.-A. and Lytras, M. D., editors, *Metadata and Semantics*, pages 299–308. Springer US.

[Salber et al., 1998] Salber, D., Dey, A. K., and Abowd, G. D. (1998). Ubiquitous computing: Defining an hci research agenda for an emerging interaction paradigm.

[Schandl and Zander, 2009] Schandl, B. and Zander, S. (2009). A framework for adaptive rdf graph replication for mobile semantic web applications. In *Joint Workshop on Advanced Technologies and Techniques for Enterprise Information Systems (Session on Managing Data with Mobile Devices)*, pages 154–163. INSTICC Press.

[Schwagereit et al., 2010] Schwagereit, F., Scherp, A., and Staab, S. (2010). Representing Distributed Groups with dgFOAF. In *ESWC2010*, pages 181–195.

[Seaman, 1999] Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Trans. Software Eng.*, 25(4):557–572.

[Shekarpour et al., 2013] Shekarpour, S., Marx, E., Ngomo, A.-C. N., and Auer, S. (2013). Sina: Semantic interpretation of user queries for question answering on interlinked data. *Submitted to Journal of Web Semantics*.

[Sheshagir et al., 2004] Sheshagir, M., Sade, N., and Gandon, F. (2004). Using Semantic Web Services for Context-Aware Mobile Applications. In *Proceedings of MobiSys2004 Workshop on Context Awareness*.

[Sonntag et al., 2007] Sonntag, D., Engel, R., Herzog, G., Pfalzgraf, A., Pfleger, N., Romanelli, M., and Reithinger, N. (2007). SmartWeb Handheld – Multimodal Interaction with Ontological Knowledge Bases and Semantic Web Services. In Huang, T. S., Nijholt, A., Pantic, M., and Pentland, A., editors, *Artifical Intelligence for Human Computing*, volume 4451 of *Lecture Notes in Computer Science*, pages 272–295. Springer.

[Soriano et al., 2006] Soriano, J., Lopez, G., Jimenez, M., Fernandez, R., and Hierro, J. J. (2006). Semanticweb content adaptation and services delivery on morfeo's semantic mobility channel. In *Proceedings of the 7th International Conference on Mobile Data Management*, MDM '06, pages 78–, Washington, DC, USA. IEEE Computer Society.

[Soylu et al., 2012] Soylu, A., Mödritscher, F., and De Causmaecker, P. (2012). Ubiquitous web navigation through harvesting embedded semantic data: a mobile scenario. *Integrated Computer-Aided Engineering*, 19(1):93–109.

[Sporny et al., 2010] Sporny, M., Corlosquet, S., Inkster, T., Story, H., Harbulot, B., and Bachmann-Gmür, R. (2010). WebID 1.0: Web identification and Discovery. Unofficial draft. `http://payswarm.com/webid/`.

[Stair and Reynolds, 2011] Stair, R. M. and Reynolds, G. W. (2011). *Principles of information systems*. CengageBrain. com.

[Steller et al., 2009] Steller, L., Krishnaswamy, S., and Gaber, M. (2009). Enabling scalable semantic reasoning for mobile services. *International Journal on Semantic Web & Information Systems*, 5(2):91–116.

[Story et al., 2009a] Story, H., Harbulot, B., Jacobi, I., and Jones, M. (2009a). Foaf+ ssl: Restful authentication for the social web. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*.

[Story et al., 2009b] Story, H., Harbulot, B., Jacobi, I., and Jones, M. (2009b). FOAF+SSL: RESTful Authentication for the Social W. In *SPOT2009*.

[Sun et al., 2005] Sun, S., Zhou, X., and Shen, H. T. (2005). Semantic caching for multiresolution spatial query processing in mobile environments. In *Proceedings of the 9th international conference on Advances in Spatial and Temporal Databases*, SSTD'05, pages 382–399, Berlin, Heidelberg. Springer-Verlag.

[Thiagarajan et al., 2012] Thiagarajan, N., Aggarwal, G., Nicoara, A., Boneh, D., and Singh, J. P. (2012). Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50. ACM.

[Tramp et al., 2011a] Tramp, S., Frischmuth, P., Arndt, N., Ermilov, T., and Auer, S. (2011a). Weaving a distributed, semantic social network for mobile users. In *Proceedings of the 8th extended semantic web conference on The semantic web: research and applications - Volume Part I*, ESWC'11, pages 200–214, Berlin, Heidelberg. Springer-Verlag.

[Tramp et al., 2011b] Tramp, S., Frischmuth, P., Arndt, N., Ermilov, T., and Auer, S. (2011b). Weaving a Distributed, Semantic Social Network for Mobile Users. In *Proceedings of the ESWC2011*.

[Tramp et al., 2010a] Tramp, S., Frischmuth, P., Ermilov, T., and Auer, S. (2010a). Weaving a Social Data Web with Semantic Pingback. In Cimiano, P. and Pinto, H., editors, *Proceedings of the EKAW 2010 - Knowledge Engineering and Knowledge Management by the Masses; 11th October-15th October 2010 - Lisbon, Portugal*, volume 6317 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 135–149, Berlin / Heidelberg. Springer.

[Tramp et al., 2011c] Tramp, S., Frischmuth, P., Ermilov, T., Shekarpour, S., and Auer, S. (2011c). An architecture of a distributed semantic social network. *Semantic Web*.

[Tramp et al., 2010b] Tramp, S., Heino, N., Auer, S., and Frischmuth, P. (2010b). Rdfauthor: Employing rdfa for collaborative knowledge engineering. In Cimiano,

P. and Pinto, H., editors, *Knowledge Engineering and Management by the Masses*, volume 6317 of *Lecture Notes in Computer Science*, pages 90–104. Springer Berlin / Heidelberg.

[Tramp et al., 2010c] Tramp, S., Heino, N., Auer, S., and Frischmuth, P. (2010c). RDFauthor: Employing RDFa for collaborative Knowledge Engineering. In Cimiano, P. and Pinto, H., editors, *Proceedings of the EKAW 2010 – Knowledge Engineering and Knowledge Management by the Masses; 11th October–15th October 2010 – Lisbon, Portugal*, volume 6317 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 90–104, Berlin/Heidelberg. Springer.

[Tummarello et al., 2007] Tummarello, G., Delbru, R., and Oren, E. (2007). Sindice.com: Weaving the Open Linked Data. In Aberer, K., Choi, K.-S., Noy, N. F., Allemang, D., Lee, K.-I., Nixon, L. J. B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., and Cudré-Mauroux, P., editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 552–565. Springer.

[Van Woensel et al., 2011a] Van Woensel, W., Casteleyn, S., Paret, E., and De Troyer, O. (2011a). Mobile querying of online semantic web data for context-aware applications. *IEEE Internet Computing*, 15(6):32–39.

[Van Woensel et al., 2011b] Van Woensel, W., Casteleyn, S., Paret, E., and De Troyer, O. (2011b). Transparent mobile querying of online rdf sources using semantic indexing and caching. In *Proceedings of the 12th international conference on Web information system engineering*, WISE'11, pages 185–198, Berlin, Heidelberg. Springer-Verlag.

[Veijalainen et al., 2006] Veijalainen, J., Nikitin, S., and Tormala, V. (2006). Ontology-based semantic web service platform in mobile environments. In *Proceedings of the 7th International Conference on Mobile Data Management*, MDM '06, pages 83–, Washington, DC, USA. IEEE Computer Society.

[Viana et al., 2007] Viana, W., Filho, J. B., Gensel, J., Oliver, M. V., and Martin, H. (2007). Photomap - automatic spatiotemporal annotation for mobile photos. In *Proceedings of the 7th international conference on Web and wireless geographical information systems*, W2GIS'07, pages 187–201, Berlin, Heidelberg. Springer-Verlag.

[Villalonga et al., 2009] Villalonga, C., Strohbach, M., Snoeck, N., Sutterer, M., Belaunde, M., Kovacs, E., Zhdanova, A. V., and Goix, L. W. (2009). Mobile ontology: Towards a standardized semantic model for the mobile domain. In *In Proc. of the 1st International Workshop on Telecom Service Oriented Architectures (TSOA 2007) at the 5th International Conference on Service-Oriented Computing, 17 September 2007.*

[W3C, 2004] W3C (2004). Resource description framework (rdf). `http://www.w3.org/RDF/`.

[W3C, 2009] W3C (2009). W3C semantic web activity. Última visita 8/6/2010.

[Wang et al., 2006] Wang, F., Liu, S., Liu, P., and Bai, Y. (2006). Bridging physical and virtual worlds: complex event processing for rfid data streams. In *Advances in Database Technology-EDBT 2006*, pages 588–607. Springer.

[WeiBenberg et al., 2006] WeiBenberg, N., Gartmann, R., and Voisard, A. (2006). An ontology-based approach to personalized situation-aware mobile service supply. *Geoinformatica*, 10(1):55–90.

[Weiser, 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific american*, 265(3):94–104.

[Wikipedia, 2013] Wikipedia (2013). SPARQL — Wikipedia, The Free Encyclopedia. [Online; accessed 31-March-2013].

[Wilson et al., 2005a] Wilson, M., Russell, A., Smith, D. A., Owens, A., and Schraefel, M. C. (2005a). mSpace Mobile: A Mobile Application for the Semantic Web. In *Proc. of the ISWC 2005 Workshop on End User Semantic Web Interaction, Galway, Ireland, November 7, 2005*, volume 172 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Wilson et al., 2005b] Wilson, M. L., Russell, A., Smith, D. A., Owens, A., and m.c. schraefel (2005b). mspace mobile: A mobile application for the semantic web. In *End User Semantic Web Workshop, ISWC2005*.

[Yu et al., 2012] Yu, H. Q., Zhao, X., Reiff-Marganiec, S., and Domingue, J. (2012). Linked context: A linked data approach to personalised service provisioning. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 376–383. IEEE.

[Yu, 2007] Yu, L. (2007). *Introduction to Semantic Web and Semantic Web services*. Chapman & Hall/CRC, Boca Raton, FL.

[Zargayouna and Amara-Hachmi, 2006] Zargayouna, H. and Amara-Hachmi, N. (2006). Litemap: An ontology mapping approach for mobile agents' context-awareness. In *OTM Workshops (2)'06*, pages 1934–1943.

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den 3.2.2014

Timofey Ermilov