SEBASTIAN TRAMP

DISTRIBUTED SEMANTIC SOCIAL NETWORKS

[April 15, 2013 at 13:37]

DISTRIBUTED SEMANTIC SOCIAL NETWORKS

SEBASTIAN TRAMP

Architecture and Applications

Oktober 2011

[April 15, 2013 at 13:37]

Sebastian Tramp: *Distributed Semantic Social Networks*, Architecture and Applications, © Oktober 2011

Ohana means family.
Family means nobody gets left behind, or forgotten.

— Lilo & Stitch

Dedicated to the loving memory of Rudolf Miede.
1939 – 2005

ABSTRACT

Short summary of the contents in English...

ZUSAMMENFASSUNG

Kurze Zusammenfassung des Inhaltes in deutscher Sprache...

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

Put your publications from the thesis here. The packages `multibib` or `bibtopic` etc. can be used to handle multiple different bibliographies in your document.

ACKNOWLEDGMENTS

Put your acknowledgments here.

We would like to thank our colleagues from AKSW research group (in particular Jonas Brekle, Nadine Jänicke and Norman Heino) for their helpful comments and inspiring discussions during the development of this approach. This work was partially supported by a grant from the European Union's 7th Framework Programme provided for the project LOD2 (GA no. 257943).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

Part I

INTRODUCTION AND PRELIMINARIES

1 MOTIVATION

2 RESEARCH QUESTIONS

The research reported in this dissertation addresses the following principal question:

> To what extent can Semantic Web technologies be deployed to support the structure, the maintenance as well as the usage of distributed social networks on the web?

This principal question can be broken down into a number of specific research questions:

1. How can we support the transformation of social data from existing social applications?
2. What is a minimal Social Semantic Web environment and which specific Semantic Web technologies are needed to form such a minimal environment?
3. How can we enable existing social applications to be part of the Social Semantic Web?
4. Which types of applications can provide user interfaces to the Social Semantic Web?
5. Which application characteristics are particularly convenient to provide social network access?
6. Which user interface patterns are qualified for distributed social networks?

3 APPROACH

4 STATE OF THE ART AND TERM DEFINITIONS

4.1 SEMANTIC WEB

4.2 SOCIAL WEB / WEB 2.0

5 STRUCTURE OF THE DISSERTATION

Part II

AN ARCHITECTURE OF A DISTRIBUTED SEMANTIC SOCIAL NETWORK

todo

6 INTRODUCTION AND REQUIREMENTS

Online social networking has become one of the most popular services on the Web. Especially Facebook with its 845Mio+ monthly active users and 100Mrd+ friendship relations creates a Web inside the Web[1]. Drawing on the metaphor of islands, Facebook is becoming more like a continent. However, users are locked up on this continent with hardly any opportunity to communicate easily with users on other islands and continents or even to relocate trans-continentally. Users are bound to a certain platform and hardly have the chance to migrate easily to another social networking platform if they want to preserve their connections. Once users have published their personal information within a social network, they often also lose control over the data they own, since it is stored on a single company's servers. Interoperability between platforms is very rudimentary and largely limited to proprietary APIs. In order to keep data up-to-date on multiple platforms, users have to modify the data on every single platform or information will diverge. Since there are only a few large social networking players, the Web also loses its distributed nature. According to a recent comScore study[2], Facebook usage times already outnumber traditional Web usage by factor two and this divergence is continuing to increase.

We argue that solutions to social networking should be engineered in distributed fashion so that users are empowered to regain control over their data. The currently vast oceans between social networking continents and islands should be bridged by high-speed connections allowing data and users to travel easily and quickly between these places. In fact, we envision the currently few social networking continents to be complemented by a large number of smaller islands with a tight network of bridges and ferry connections between them. Compared with the currently prevalent centralized social networks such an approach has a number of advantages:

- *Privacy.* Users of the distributed semantic social network (DSSN) can setup their own DSSN node or chose a DSSN node provider with particularly strict privacy rules in order to ensure a maximum of privacy. This would facilitate a competition of social network operators about the privacy rules most beneficial for users. Currently, due to the oligopoly in the social networking market, which is dominated by big players such as Facebook,

1 http://www.sec.gov/Archives/edgar/data/1326801/000119312512034517/d287954ds1.htm

2 http://allthingsd.com/20110623/the-web-is-shrinking-now-what/

Google or Twitter, privacy regulations are often more driven by the commercialization interests.

- *Data security.* Due to the distributed nature it is more difficult to steal large amounts of private data. Also, security is ensured through public review and testing of open-standards and to a lesser extend through obscurity due to closed proprietary implementations. As is confirmed very frequently, centralized solutions are always more endangered of attacks on data security. Even with the best technical solutions in place, insider threats can hardly be prevented in a centralized setting but can not cause that much harm to a DSSN.

- *Data ownership.* Users can have full ownership and control over the use of their data. They are not restricted to ownership regulations imposed by their social network provider. Instead DSSN users can implement fine-grained data licensing options according to their needs. A DSSN would moreover facilitate a competition of DSSN node providers for the most liberal and beneficial data ownership regulations for users.

- *Extensibility.* The representation of social network resources like WebIDs and data artefacts is not limited to a specific schema and can grow with the needs of the users[3]. Although extensibility is also easily to realize in the centralized setting (as is confirmed by various APIs, e.g., Open Social), a centralized social network setting could easily prohibit (or censor) certain extensions for commercial (or political) reasons and thus constrain the freedom of its users.

- *Reliability.* Again due to the distributedness the DSSN is much less endangered of breakdowns or cyberterrorism, such as denial-of-service attacks.

- *Freedom of communication.* As we observed recently during the Arab Spring where social networking services helped protesters to organize them self, social networks can play a crucial role in attaining and defeating civil liberties. A DSSN with a vast amount of nodes is much less endangered of censorship as compared to centralized social networks.

In this part of the thesis we describe the main technological ingredients for a DSSN as well as their interplay. The semantic representation of personal information is facilitated by a *WebID profile*. The WebID protocol allows for using a WebID profile for authentication and access

3 We already experimented with activities like git commits and comments on lines of source code – both usecases integrate very well because of the schema-agnostic transport protocols and the Linked Data paradigm: https://github.com/seebi/lib-dssn-php

control purposes. *Semantic Pingback* facilitates the first contact between users of the social network and provides a method for communication about resources (such as images, status messages, comments, activities) on the social network. Finally, *PubSubHubbub*-based subscription services allow for obtaining near-instant notifications of specific information as WebID profile change sets and activity streams from people in one's social network. Together, these standards and protocols provide all necessary ingredients to realize a distributed social network having all the crucial social networking features provided by centralized ones.

7 ARCHITECTURAL OVERVIEW

In this chapter, we describe the DSSN reference architecture. 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 1, where the arrows 1-7 can be described in the following way:

(1) Resources announce services and feeds via links or header fields, 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 demand of an application).

(5) Personal and global search services can index resources and are used by applications.

(6) Access to resources and services can be delegated to applications by a WebIDs, 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.

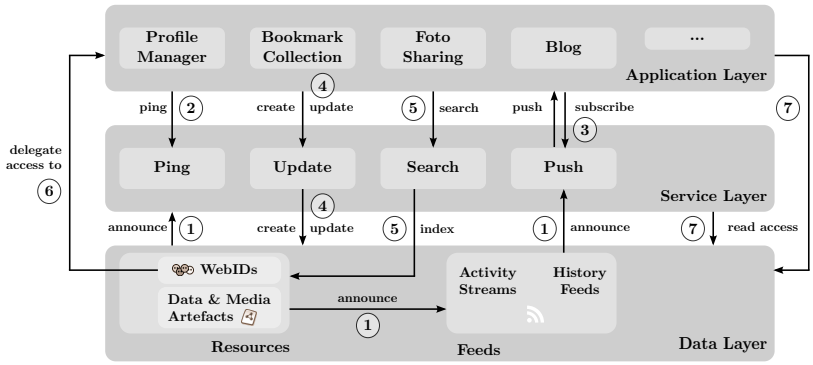

Figure 1: Architecture of a Distributed Semantic Social Network

7.1 BASIC DESIGN PRINCIPLES

Our DSSN architecture is based on the following three design principles.

LINKED DATA. The main protocol for data publishing, retrieval and integration is based on the Linked Data principles [**?**]. All of the information contained in the DSSN is represented according to the RDF paradigm, made de-referencable and interlinked with other resources. This principle facilitates heterogeneity as well as extensibility and enables the distribution of data and services on the Web. The resulting overall distributive character of the architecture fosters reliability and freedom of communication and leads to more data security by design.

SERVICE DECOUPLING. A second fundamental design principle is the decoupling of user data from services as well as applications [**?**]. It ensures that users of the network are able to choose between different services and applications. As a result, this principle enables an even more distributed character of the social network which stresses the same issues as distributed Linked Data. In addition, this principle helps users of the DSSN to distinguish between their own data, which they share with and license to other people and services, and foreign data, which they create by using these services and which they do not own. This turns the un-balanced power structure of centralized social networks upside down by strictly settling the ownership of the data to the user side and allowing access to that data in an opt-in way, which leads to more privacy.

PROTOCOL MINIMALISM. The main task for social networking protocols is to communicate RDF triples between DSSN nodes, 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 and reliable.

7.2 DATA LAYER

The data layer comprises two main data structures: *resources* for the description of static entities and *feeds* for the representation and publishing of events and activities.

7.2.1 *Resources*

We distinguish between three main categories of DSSN resources: *WebIDs* for persons as well as applications, *data artefacts* and *media artefacts*. The properties, conditions and roles in the network of these resources are described in the next paragraphs.

WEBID [?][1] 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[2]. 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 in any mix of suitable vocabularies and FOAF [?] as the fundamental 'industry standard' which can be extended[3]. An example WebID profile comprising some personal information (lines 8-12) and two `rel:worksWith`[4] links to co-workers (lines 6-7) is shown in Listing 1.

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

Listing 1: A minimal WebID profile with personal information and two `worksWith` relations to other WebIDs.

Apart from the main focus on representing user profiles, our architecture extends the WebID concept by facilitating two additional tasks: *service discovery* and *access delegation*.

Service discovery is used to equip a WebID with relations to trusted services which have to be used with that WebID. The usage of the WebID itself ensures that an agent can trust this service in the same way as she trusts the owner of the WebID. The most important service in our DSSN architecture is the Semantic Pingback service, which we describe in detail in Section 7.3.2.

In addition to this service, we introduce access delegation for the WebID protocol. WebID access delegation is an enhancement to the current WebID authentication process in order to allow applications to access resources and services on behalf of the WebID owner, but without the need to introduce additional application certificates in a

1 Formerly know as the FOAF+SSL best practice [?], the latest specification is available at http://webid.info/spec/.

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

3 In theory, FOAF can be replaced by another vocabulary but as a grounding for semantic interoperability, we suggest to use it.

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

WebID. We describe the WebID protocol as well as our access delegation extension in detail in Section 7.3.1.

Agents and Applications play an important role in today's social networks[5]. They have access to large parts of the profile data and can add or change some of the profile information, e.g. create activity descriptions or create and link images. Applications on the DSSN are also identified by using WebID profiles, but are not described as a person but as an application. They can act on behalf of a person but rely on delegated access rights for such an activity. This process is described in Section 7.3.1.

DATA ARTEFACTS are resources on the Web which are published according to the Linked Data principles. Data artefacts includes posts, comments, tag assignments, activities and other Social Web artefacts which have been created by services and applications on the Web. Most of them are described by using specific Web ontologies such as SIOC [**?**], Common Tag[6] or Activity Streams in RDF [**?**].

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[7]. 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 in Section 7.3.2, and a link to a push-enabled activity stream. An example photo-sharing application is described in Section 7.5.

7.2.2 *Feeds*

Feeds are used to represent temporally ordered information in a machine-readable way. Feeds are widely used on the Web and play a crucial role in combination with the *PubSubHubbub protocol* to enable near real-time communication between different services. In the context of the DSSN architecture, two types of feeds are worth considering:

ACTIVITY FEEDS describe the latest social network activities of a user in terms of an `actor - verb - object` triple where activity verbs are used as types of activities (e.g. to post, to share or to bookmark a

5 Social network games such as FarmVille can have more than 80 million users (according to `appdata.com`), which constantly create activity descriptions.

6 `http://commontag.org/Specification`

7 Typically, the user uploads the binary part and the service creates the meta-data part based on additional form data and extracted meta-data from the binary part.

specific object)[8]. Activity feeds can be used to produce a merged view of the activities of one's own social network. In our DSSN architecture, each activity is created as a Linked Data resource (i.e. a DSSN data artefacts), which links to the actor and object of the activity. In addition, each activity is equipped with a Pingback Server in order to allow for receiving reactions on this activity (called pingbacks) and thus to spin a content network between these artefacts.

HISTORY FEEDS are used to allow syndication of change sets of specific resources between a publisher of a resource and many subscribers of the resource. History feeds describe changes in RDF resources in terms of added and deleted statements which are boxed in an Atom feed entry. A subscriber's social network application can use this information to maintain an exact copy of the original resource for caching and querying purposes. History feeds are in particular important for the syndication of changes of WebID profiles (e.g. if a contact changes its phone number).

7.3 PROTOCOL LAYER

The protocol layer consists of the WebID identity protocol and two networking protocols which provide support for two complete different communication schemes, namely resource linking and push notification.

7.3.1 *WebID (protocol)*

From a more technical perspective, the WebID protocol [**?**] incorporates authentication and trust into the WebID concept. The basic idea 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*[9] 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 can provide 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* [**?**]). An example of a WebID profile, which is annotated with a public key, is shown in Listing 2. This WebID profile contains

8 Activity streams (http://activitystrea.ms) are Atom format extensions to describe activity feeds. It is extensible in a way that allows publishers to use new verb- or object-type IRIs to identify site-specific activities.

9 http://www.ietf.org/rfc/rfc2459.txt

additionally 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).

```
13 @prefix rsa:  <http://www.w3.org/ns/auth/rsa#>.
14 @prefix cert: <http://www.w3.org/ns/auth/cert#">.
15 [] a rsa:RSAPublicKey;
16   rdfs:comment "used from my smartphone ...";
17   cert:identity <http://philipp.frischmuth24.de/id/me>;
18   rsa:modulusc "C41199E ... 5AB5"^^cert:hex;
19   rsa:public_exponent "65537"^^cert:int.
```

Listing 2: An extension of the minimal WebID from Listing 1: 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.

Nevertheless, the described approach requires the user to access a secured resource directly, e.g. through a Web browser which is equipped with a WebID-enabled certificate. However, in the scenario of a distributed social network this arrangement is not always the case. If, for example, the software of user A needs to update its local cache of user B's profile, it will do so by fetching the data in the background and not necessarily when user A is connected. An obvious solution would be to hand out a WebID-enabled certificate to the software (agent), but then the user needs to create a dedicated certificate for all tools that have to access secured information and simultaneously allows all participating tools to "steal" her identity, which is not the preferred solution from a security perspective.

To resolve this dilemma, we have extended the WebID protocol by adding support for *access delegation*. By delegating access to an agent, a user allows a particular agent to deputy access-secured information resources. The agent itself authenticates against the relying party by using its own credentials, e.g. by employing the WebID protocol, too. Additionally, it sends a `X-DeputyOf` HTTP header, which indicates that a resource is accessed on behalf of a certain WebID user[10]. The relying party then fetches the WebID and checks for a statement as shown in Listing 3[11].

10 We have recently discussed the motivation and solution of this extension with a few members of the W3Cs WebID community group (http://www.w3.org/community/webid/) and will propose a change request to extend the specification regarding access delegation.

11 The `dssn:deputy` relation and other related schema resources introduced in this paper are part of the DSSN namespace, which is available at http://purl.org/net/dssn/.

```
20 @prefix dssn: <http://purl.org/net/dssn/>.
21 <http://philipp.frischmuth24.de/id/me> dssn:deputy <http://myagent.org/> .
```

Listing 3: Access delegation through the `dssn:deputy` property.

If such a statement is found and the relying party trusts the accessing agent, then access to the secured resource is granted. Given that the user is always able to modify the information provided by her WebID, she stays in control with regard to the delegation of access to other parties. In contrast to other access control solutions such as OAuth[12], where an API provider needs to serve and handle access tokens, a user only has to maintain one single central resource, her WebID.

7.3.2 *Semantic Pingback*

The purpose of *Semantic Pingback* [**?**] in the context of a DSSN architecture is twofold:

- It is used to facilitate the first contact between two WebIDs and establish a new connection (*friending*).
- 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 [**?**], which is one of the technological cornerstones of the overwhelming success of the blogosphere in the Social Web. The overall architecture is depicted in Figure 2.

The Semantic Pingback mechanism enables bi-directional links between WebIDs, RDF resources as well as weblogs and websites in general (cf. Figure 8). It facilitates contact/author/user notifications in case a link has been newly established. It is based on the advertisement of a lightweight RPC service[13] in the RDF document, HTTP or HTML header of a certain Web resource, which should be called as soon as a (typed RDF) link to that resource is established. The Semantic Pingback mechanism allows casual users and authors of RDF content, of weblog entries or of 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 publish backlinks automatically from the original WebID profile (or other content, e.g. status messages) 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 can be much more tightly and timelier interlinked by using the Semantic Pingback mechanism than conventional websites, thus rendering a network effect, which is one of

12 http://oauth.net/

13 In fact, we experimented with different service endpoints. Based on the results, which are described in more detail in [**?**], we now prefer simple HTTP post requests which are not compatible with standard XML-RPC pingbacks.

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. An extension of our example profile with Semantic Pingback functionality making use of an external Semantic Pingback service is shown in Listing 4. In line 23, the subject resource is linked with the `ping:to` relation to the Semantic Pingback service.

As requested by our third DSSN 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 activities.

FRIENDING is the process of establishing a symmetric `foaf:knows` relation between two WebIDs. A relationship is approved when both persons publish this relation in their WebIDs. A typical friending work flow can be described by the following steps:

- Alice publishes a `foaf:knows` relation to Bob in her WebID profile.
- Alice's WebID hosting service pings Bob's WebID to inform Bob about this new statement.
- Bob receives a message from his Pingback Service.
- Bob can approve this relation by publishing it in his WebID profile, which sends again a ping back to Alice[14].

This basic model of communication can be applied to different events and activities in the social network. Table 1 lists some of the more important pingback events[15]. In any case, the owner of these resources can be informed about the event and in most cases specific actions should be triggered (refer to Section 15.3 for more details). However, a specific reaction is not enforced by the protocol.

14 Please note that the Semantic Pingback protocol does not enforce any specific reaction on a certain relation type or any reaction at all.

15 The prefix `sioct` refers to the SIOC types ontology module namespace (`http://rdfs.org/sioc/types#`) while `ctag` refers to the Common Tag Ontology namespace (`http://commontag.org/ns#`). The prefix `aair` refers to the Atom Activity Streams RDF mapping ontology (`http://xmlns.notu.be/aair#`).

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

Listing 4: Extension of the minimal WebID profile from Listing 1: Assignment of an external Semantic Pingback service which can be used to ping this specific resource.

Table 1: Typical RDF statements which can cause ping activities.

source resource	object property	target resource	description
WebID (`foaf:Person`)	`foaf:knows`	WebID (`foaf:Person`)	friending
`sioct:Comment`	`sioc:about`	`foaf:Image`	commenting an image (or an
`sioc:Post`	`sioc:reply_of`	`sioc:Post`	replying to a (friends) post
`ctag:Tag`	`ctag:tagged`	*	any resource is tagged by a u
`aair:Activity`	`aair:activityObject`	*	any resource is object of an a

7.3.3 *PubSubHubbub*

PubSubHubbub[16] 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 a Linked Data 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.

The main work flow of establishing a PubSubHubbub connection can be described as follows: The feed publisher advertises a hub service in an existing feed. A subscriber follows this link and requests a subscription on this feed. If the feed changes, the feed publisher informs the hub service which instantly broadcasts the changes to all subscribers[17]. The main advantage of this communication model is to avoid frequent and unnecessary pulls of all interested subscribers from this feed and to allow a faster broadcast to the subscriber.

In the DSSN architecture, two specific feeds are important and interlinked with a WebID to allow for subscriptions: *activity feeds* which are used for activity distribution and *history feeds* which are used for resource synchronization.

ACTIVITY DESCRIPTION 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. These activity descriptions can and should be created by any application which is allowed to update the feed (ref. access delegation). In addition, activity

16 http://code.google.com/p/pubsubhubbub/

17 During the subscription a callback endpoint is supplied by the subscribing endpoint, which is later used for pushing the data.

feeds can be created for data and media artefacts in order to allow object-centered push notification. Typically, activity feed updates are pushed to a personal search / index service of a subscribed user (see next section).

RESOURCE SYNCHRONIZATION is an additional communication scheme based on feeds. It is required, especially in distributed social networks, to take into account that relevant data is highly distributed over many locations and that access to and querying of this data can be very time-consuming without caching. A properly connected resource synchronization tackles this problem by allowing users to subscribe to changes of certain resources over PubSubHubbub. For a WebID, this process can be included into the *friending* process, while for other resources a user can subscribe manually (e.g. if a user is member of a certain group, then she may subscribe to the feed of a group resource to receive updates). Resource synchronization is a well-known topic when dealing with distributed resources. We have designed our data model as Linked Data update logs [**?**] based on the work previously published by the Triplify project[18].

7.4 SERVICE LAYER

Services are applications which are part of the DSSN infrastructure (in contrast to applications from the application layer). 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 8, we have defined four essential services for a DSSN.

Ping Service

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 sending notification.

A ping service can be announced with the `ping:to` relation as shown in Listing 4.

18 http://triplify.org

Push Service

The push service is used for activity distribution and resource synchronization. Both introduced types of feeds announce its push service in the same way as using the `rel="hub"` link in the feed head. Since both types of feeds are valid atom feeds, a DSSN push service can be a standard PubSubHubbub-based instance.

To equip social network resources with its corresponding activity and history feeds, we have defined two OWL object properties which are sub-properties of the more generic `sioc:feed` relation from the SIOC project [**?**]: `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 integration of media artefacts without too much effort.

Search and Index Service

Search and index services are used in two different contexts in the DSSN 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 [**?**] or Sindice [**?**]. They use crawlers to keep their resource cache up-to-date and provide user interfaces as well as application programming interfaces to integrate and use their services in applications.

   In our architecture, these public services are used to search for new contacts as well as other artefacts in the same way as people can use a standard Web search engine. The main advantage in using Semantic Web search engines lies in their ability to use graph patterns for a search[19].

2. In addition to public search services, we want to emphasize the importance of private search services in our architecture. Private search services are used in order to have a fast resource cache not only for public, but also for private data which a user is allowed to access.

   A private search service is used for all users and 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.

19 A motivating example in our context is the search for resources of type `foaf:Person`, which are related to the DBpedia topic `dbpedia:DataPortability` (e.g. with the `foaf:interest` object property).

> That is, she is able to query over the latest up-to-date data by using her private search service. In addition, she can query for data which has never been public and is published for a few people only.

Since private search services are used by applications which act on behalf of the user, they must be WebID-protocol-enabled. That is, they accept requests from the user and her delegated agents only. In addition, applications need to know which private search service should be accessed on behalf of the user. This mode is again made possible by providing a link from the WebID to the search service[20]

In our architecture we assume that search services accept SPARQL queries.

Update Service

Finally, an update service provides an interface to modify and create user resources in terms 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[21].

Typical examples of how to use this service are the creation of activities on behalf of the user or the modification of the user's WebID, e.g. by adding a new `foaf:knows` relation.

In the next section, we give a detailed description of the service interplay and usage by applications.

7.5 APPLICATION LAYER

Social Web applications create and modify all kinds of resources for a user. In our architecture, they have to use the trusted services which are related to a WebID instead of their own. Since access to these services is exclusively delegated by the user to an application, the user has full control over her data[22]. To illustrate how DSSN applications work with a WebID and its services, we will describe a simple photo-sharing application:

20 In our prototypes we use a simple OWL object property `dssn:searchService`, which is a sub-property of `dssn:trustedService`. We assume that such an easy vocabulary is only the first step to a fully featured service auto-discovery ontology and consider all `dssn` terms as unstable.

21 We defined `dssn:updateService` as a relation between a WebID and an update service.

22 At the moment we distinguish only between access and no access to a service. As an extension, we can imagine that a private search service can handle access on parts of the private Social Graph differently (an online game does not need to know which other activities you pursue on the Web). Access policies for RDF knowledge bases is a topic of ongoing research and we hope that the results of this research area can be adapted here.

When a user creates an account on this service, she uses her WebID for the first login and delegates access to this application. The application analyses the WebID and discovers the trusted services and some meta-data of the user (e.g. name, short bio and depiction). The user then uploads her first image to the application. The application creates a new image resource and an activity stream for that resource. After that, the application creates two activities for the user: one in the stream of the image resource and one in the personal stream of the user, employing the recently delegated access right[23]. Furthermore, it equips the newly uploaded image with the pingback service of the user; thus enabling the image for backlinks and comments. New comments can arrive from everywhere on the Web, but the application also provides its own commenting service (integrated in the image Web view). If another user writes a comment on this image, a data artefact is created in the namespace of the application and a ping request is sent to the user's pingback service (since this service is related to the image).

This simple example demonstrates the interplay and rules of the DSSN service architecture. A more complex social network application is described in the next section.

23 This activity in the users stream is instantly pushed to all of the user's friends and is not part of the data of the image publishing service.

8

SEMANTIC PINGBACK

8.1 INTRODUCTION

Recently, the publishing of structured, semantic information as Linked Data has gained much momentum. A number of Linked Data providers meanwhile publish more than 200 interlinked datasets amounting to 13 billion facts[1]. Despite this initial success, there are a number of substantial obstacles, which hinder the large-scale deployment and use of the Linked Data Web. These obstacles are primarily related to the *quality, timeliness and coherence* of Linked Data. In particular for ordinary users of the Internet, Linked Data is not yet sufficiently visible and (re-) usable. Once information is published as Linked Data, authors hardly receive feedback on its use and the opportunity of realising a network effect of mutually referring data sources is currently unused.

In this paper we present an approach for complementing the Linked Data Web with a social dimension. The approach is based on an extension of the well-known Pingback technology [**?**], which is one of the technological cornerstones of the overwhelming success of the blogosphere in the Social Web. The Pingback mechanism enables bidirectional links between weblogs and websites in general as well as author/user notifications in case a link has been newly established. It is based on the advertising of a lightweight RPC service, in the HTTP or HTML header of a certain Web resource, which should be called as soon as a link to that resource is established. The Pingback mechanism enables authors of a weblog entry or article to obtain immediate feedback, when other people reference their work, thus *facilitating reactions and social interactions*. It also allows to automatically publish backlinks from the original article to comments or references of the article elsewhere on the Web, thus *facilitating timeliness and coherence* of the Social Web. As a result, the distributed network of social websites using the Pingback mechanism (such as the blogosphere) is 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.

With this work we aim to apply this success of the Social Web to the Linked Data Web. We extend the Pingback mechanism towards a Semantic Pingback, by adding support for typed RDF links on Pingback clients, servers and in the autodiscovery process.

1 http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics

When an RDF link from a Semantic Pingback enabled Linked Data resource is established with another Semantic Pingback enabled Linked Data resource, the latter one can be automatically enriched either with the RDF link itself, with an RDF link using an inverse property or additional information. When the author of a publication, for example, adds bibliographic information including RDF links to co-authors of this publication to her semantic wiki, the co-authors' FOAF profiles can be enriched with backlinks to the bibliographic entry in an *automated or moderated* fashion. The Semantic Pingback supports *provenance* through tracking the lineage of information by means of a provenance vocabulary. In addition, it allows to implement a variety of measures for *preventing spam*.

Semantic Pingback is completely downwards compatible with the conventional Pingback implementations, thus allowing to seamlessly connect and interlink resources on the Social Web with resources on the Data Web. A weblog author can, for example, refer to a certain Data Web resource, while the publisher of this resource can get immediately notified and `rdfs:seeAlso` links can be automatically added to the Data Web resource. In order to facilitate the adoption of the Semantic Pingback mechanism we developed three complementary implementations: a Semantic Pingback implementation was included into the semantic data wiki OntoWiki, we added support for Semantic Pingbacks to the Triplify database-to-RDF mapping tool and provide a standalone implementation for the use by other tools or services.

The paper is structured as follows: We describe the requirements which guided the development of Semantic Pingback in section 15.2. We present an architectural overview including communication behaviour and autodiscovery algorithms of our solution in section 8.3. A description of our implementations based on OntoWiki and Triplify as well as the standalone software is given in section 15.3. Finally, we survey related work in section 15.9 and conclude with an outlook on future work in section 15.10.

8.2 REQUIREMENTS

In this section we discuss the requirements, which guided the development of our Semantic Pingback approach.

8.2.0.1 *Semantic links.*

The conventional Pingback mechanism propagates untyped (X)HTML links between websites. In addition the Semantic Pingback mechanism should be able to propagate typed links (e.g. OWL object properties) between RDF resources.

8.2.0.2 *Use RDFa-enhanced content where available.*

Since most traditional weblog and wiki systems are able to create semantically enriched content based on RDFa annotations[2], these systems should be able to propagate typed links derived from the RDFa annotations to a Semantic Pingback server without any additional modification or manual effort.

8.2.0.3 *Downward compatibility with conventional Pingback servers.*

Conventional Pingback servers should be able to retrieve and accept requests from Semantic Pingback clients. Thus, widely used Social Web software such as WordPress or Serendipity can be pinged by a Linked Data resource to announce the referencing of one of their posts. A common use case for this is a Linked Data SIOC [**?**] comment which replies and refers to a blog post or wiki page on the Social Web. Such a SIOC comment typically uses the `sioc:reply_of` object property to establish a link between the comment and the original post[3].

8.2.0.4 *Downward compatibility for conventional Pingback clients.*

Conventional Pingback clients should be able to send Pingbacks to Semantic Pingback servers. Thus, a blogger can refer to any pingback-enabled Linked Data resource in any post of her weblog. Hence, the conventional Pingback client should be able to just send conventional Pingbacks to the Linked Data server. Unlike a conventional Pingback server, the Semantic Pingback server should not create a comment with an abstract of the blog post within the Linked Data resource description. Instead an additional triple should be added to the Linked Data resource, which links to the referring blog post.

8.2.0.5 *Support Pingback server autodiscovery from within RDF resources.*

The conventional Pingback specification keeps the requirements on the client side at a minimum, thus supporting the announcement of a Pingback server through a `<link>`-Element in an HTML document. Since the Semantic Pingback approach aims at applying the Pingback mechanism for the Web of Data, the autodiscovery process should be extended in order to support the announcement of a Pingback server from within RDF documents.

2 This should be possible at least manually by using the systems HTML source editor, but can be supported by extensions as for example described in [**?**] for Drupal.

3 Since SIOC is a very generic vocabulary, people can also use more specific relations as, for instance, `disagreesWith` or `alternativeTo` from the Scientific Discourse Relationships Ontology [**?**].

8.2.0.6 *Provenance tracking.*

In order to establish trust on the Data Web it is paramount to preserve the lineage of information. The Semantic Pingback mechanism should incorporate the provenance tracking of information, which was added to a knowledge base as result of a Pingback.

8.2.0.7 *Spam prevention.*

Another aspect of trust is the prevention of unsolicited proliferation of data. The Semantic Pingback mechanism should enable the integration of measures to prevent spamming of the Data Web. These measures should incorporate methods based on data content analysis and social relationship analysis.

8.3 OVERVIEW

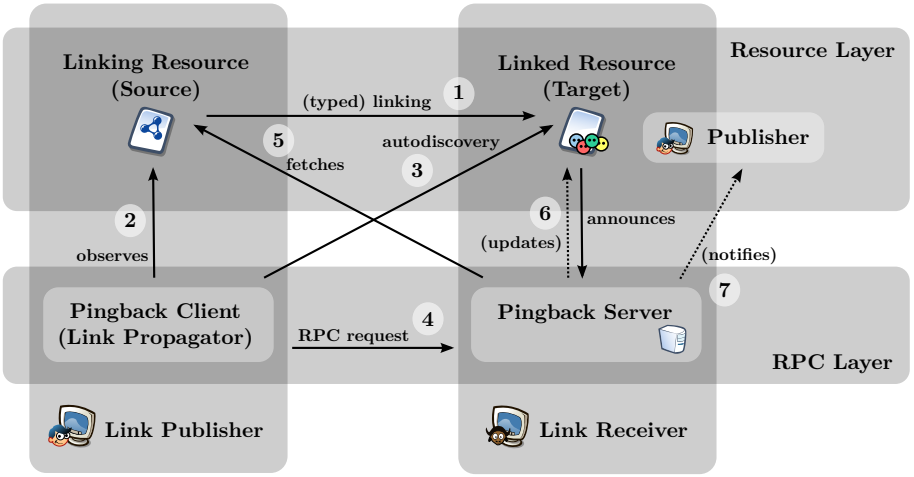

Figure 2: Architecture of the Semantic Pingback approach: (1) A *linking resource* links to another (Data) Web resource, here called *linked resource*. (2) The *Pingback client* is either integrated into the data/-content management system or realized as a separate service, which observes changes of the Web resource. (3) Once the establishing of a link has been noted, the Pingback client tries to auto-discover a Pingback server from the linked resource. (4) If the auto-discovery has been successful, the respective Pingback server is used for a ping. (5) In order to verify the retrieved request (and to obtain information about the type of the link in the semantic case), the Pingback server fetches (or de-references) the linking resource. (6 + 7) Subsequently, the Pingback server can perform a number of actions such as updating the linked resource (e.g. adding inverse links) or notifying the publisher of the linked resource (e.g. via email).

The general architecture of the Semantic Pingback approach is depicted in Figure 8. A *linking resource* (depicted in the upper left) links to another (Data) Web resource, here called *linked resource* (arrow 1). The linking resource can be either an conventional Web resource (e.g. wiki page, blog post) or a Linked Data resource. Links originating from Linked Data resources are always typed (based on the used property), links from conventional Web resources can be either untyped (i.e. plain HTML links) or typed (e.g. by means of RDFa annotations). The *Pingback client* (lower left) is either integrated into the data/content management system or realized as a separate service, which observes changes of the Web resource (arrow 2). Once the establishing of a link was noted, the Pingback client tries to autodiscover a Pingback server from the linked resource (arrow 3). If the autodiscovery was successful, the respective Pingback RPC server is called (arrow 4), with the parameters linking resource (i.e. source) and linked resource (i.e. target). In order to verify the retrieved request (and to obtain information about the type of the link in the semantic case), the Pingback server fetches (or dereferences) the linking resource (arrow 5). Subsequently, the Pingback server can perform a number of actions (arrows 6,7), such as updating the linked resource (e.g. adding inverse links) or notifying the publisher of the linked resource (e.g. via email). This approach is compatible with the conventional Pingback specification [**?**], which illustrates the chain of communication steps with the help of a Alice and Bob scenario. This scenario as well as the general architecture introduce four components, which we now describe in more detail:

8.3.0.8 *Pingback client.*

Alice's blogging system comprises the Pingback client. The Pingback client establishes a connection to the Pingback server on a certain event (e.g. on submitting a new blog post) and starts the Pingback request.

8.3.0.9 *Pingback server.*

Bob's blogging system acts as the Pingback server. The Pingback server accepts Pingback request via XML-RPC and reacts as configured by the owner. In most cases, the Pingback server saves information about the Pingback in conjunction with the target resource.

8.3.0.10 *Target resource.*

Bob's article is called the target resource and is identified by the *target URI*. The target resource can be either a web page or an RDF resource, which is accessible through the Linked Data mechanism. A target resource is called *pingback-enabled*, if a Pingback client is able to glean information about the target resource's Pingback server (see section 8.4 for autodiscovery of Pingback server information).

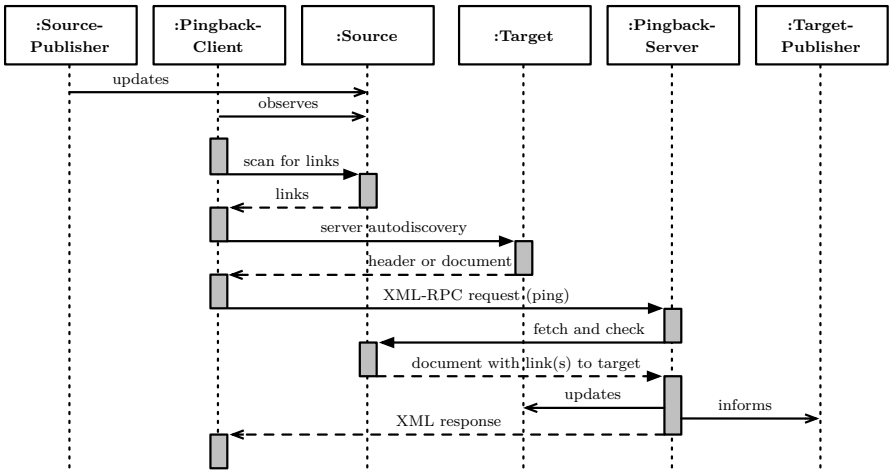

Figure 3: Sequence diagram illustrating the (Semantic) Pingback workflow.

8.3.0.11 *Source resource.*

Alice's post is called the source resource and is identified by the *source URI*. Similar as the target resource, the source resource can be either a web page or an RDF resource. The source resource contains some relevant information chunks regarding the target resource.

These information chunks can belong to one or more of the following categories:

- An *untyped (X)HTML link* in the body of the web page (this does not apply for Linked Data resources).
- A (possible RDFa-encoded) RDF triple linking the source URI with the target URI trough an arbitrary RDF property. That is, the extracted source resource model contains a *direct relation* between the source and the target resource. This relation can be directed either from the source to the target or in the opposite direction.
- A (possible RDFa-encoded) RDF triple where either the subject or the object of the triple is the target resource. This category represents *additional information* about the target resource including textual information (e.g. an additional description) as well as assertions about relations between the target resource and a third resource. This last category will most likely appear only in RDFa enhanced web pages since Linked Data endpoints are less likely to return triples describing foreign resources.

Depending on these categories, a Semantic Pingback server will handle the Pingback request in different ways. We describe this in more detail later in section 8.5.

Figure 3 illustrates the complete life-cycle sequence of a (Semantic) Pingback. Firstly, the source publisher updates the source resource,

which is observed by a Pingback client. The Pingback client then scans the source resource for links (typed or untyped) to other resources. Each time the client detects a suitable link, it tries to determine a Pingback server by means of an autodiscovery process. Once a Pingback server was determined, the client pings that server via an XML-RPC request. Section 8.4 contains a more detailed description of these steps. Since the requested Pingback server only receives the source and target URIs as input, it tries to gather additional information. At least the source document is fetched and (possibly typed) links are extracted. Furthermore the target resource is updated and the publisher of the target resource is notified about the changes. In section 8.5 the server behavior is described in more detail. Finally, the Pingback server responds with an XML result.

8.4 CLIENT BEHAVIOR

One basic design principle of the original Pingback specification is to keep the implementation requirements of a Pingback client as simple as possible. Consequently, Pingback clients do not even need an XML/HTML parser for basic functionality. There are three simple actions to be followed by a Pingback client: (1) Determine suitable links to external target resources, (2) detect the Pingback server for a certain target resource and (3) send an XML-RPC post request via HTTP to that server. Conventional Pingback clients would naturally detect (untyped) links by scanning HTML documents for `<a>`-elements and use the `href`-attribute to determine the target. Semantic Pingback clients will furthermore derive suitable links by examining RDFa annotated HTML or RDF documents. Both conventional and Semantic Pingback clients are able to communicate with a Semantic Pingback server, since the Semantic Pingback uses exactly the same communication interface. In particular, we did not change the remote procedure call, but we introduce a third possible autodiscovery mechanism for Semantic Pingback clients in order to allow the propagation of server information from within RDF documents. On the one hand, this enables the publisher of a resource to name a Pingback server, even if the HTTP header cannot be modified. On the other hand, this allows caching and indexing of Pingback server information in a Semantic Web application. Since a large number of Semantic Web applications store the data retrieved from other parties, they can take advantage of the embedded Pingback server information without requesting the data again, thus accelerating the discovery process.

Server autodiscovery

The server autodiscovery is a protocol followed by a Pingback client to determine the Pingback server of a given target resource. The Pingback

mechanism supports two different autodiscovery mechanisms which can be used by the Pingback client:

- an HTTP header attribute `X-Pingback` and
- a `link`-element in the HTML head with a relation attribute `rel="pingback"`.

Both mechanisms interpret the respective attribute value as URL of a Pingback XML-RPC service, thus enabling the Pingback client to start the request.

The `X-Pingback` HTTP header is the preferred autodiscovery mechanism and all Semantic Pingback server must implement it in order to achieve the required downward compatibility. We define an additional autodiscovery method for Linked Data resources which is based on RDF and integrates better with Semantic Web technologies.

Therefore, we define an OWL object property `service`[4], which is part of the Pingback namespace and links a RDF resource with a Pingback XML-RPC server URL. The advantage compared to an HTTP header attribute is that this information can be stored along with a cached resource in an RDF knowledge base. Another benefit is, that different resources identified by hash URIs can be linked with different Pingback servers. However, a disadvantage (as for the HTML link element too) is that Pingback clients need to retrieve and parse the document instead of requesting the HTTP header only.

8.5 SERVER BEHAVIOR

While the communication behavior of the server is completely compatible with the conventional Pingback mechanism (as described in [**?**]), the manipulation of the target resource and other request handling functionality (e.g. sending email notifications) is implementation and configuration dependent. Consequently, in this section we focus on describing guidelines for the important server side manipulation and request handling issues spam prevention, backlinking and provenance tracking.

8.5.1 *Spam Prevention*

At some point every popular service on the Internet, be it Email, Weblogs, Wikis, Newsgroups or Instant Messaging, had to face increasing abuse of their communication service by sending unsolicited bulk messages indiscriminately. Each service dealt with the problem by implementing technical as well as organizational measures, such as black- and whitelists, spam filters, captchas etc.

4 http://purl.org/net/pingback/service

The Semantic Pingback mechanism prevents spamming by the following verification method. When the Pingback Server receives the notification signal, it automatically fetches the linking resource, checking for the existence of a valid incoming link or an admissible assertion about the target resource. The Pingback server defines, which types of links and information are admissible. This can be based on two general strategies:

- *Information analysis.* Regarding an analysis of the links or assertions, the Pingback server can, for example, dismiss assertions which have logical implications (such as domain, range or cardinality restrictions), but allow label and comment translations into other languages.
- *Publisher relationship analysis.* This can be based e.g. on the trust level of the publisher of the linking resource. A possibility to determine the trust level is to resolve `foaf:knows` relationships from the linked resource publisher to the linking resource publisher.

If admissible links or assertions exist, the Pingback is recorded successfully, e.g. by adding the additional information to the target resource and notifying its publisher. This makes Pingbacks less prone to spam than e.g. trackbacks[5].

In order to allow conventional Pingback servers (e.g. WordPress) to receive links from the Data Web, this link must be represented in a respective HTML representation of the linking resource (managed by the Pingback client) at least as an untyped X(HTML) link. This enables the server to verify the given source resource even without being aware of Linked Data and RDF.

8.5.2 *Backlinking*

The initial idea behind propagating links from the publisher of the source resource to the publisher of the target resource is to automate the creation of backlinks to the source resource. In typical Pingback enabled blogging systems, a backlink is rendered in the feedback area of a target post together with the title and a short text excerpt of the source resource.

To retrieve all required information from the source resource for verifying the link and gather additional data, a Semantic Pingback server will follow these three steps:

1. Try to catch an RDF representation (e.g. RDF/XML) of the source resource by requesting Linked Data with an HTTP `Accept` header.

5 http://en.wikipedia.org/wiki/Trackback

2. If this is not possible, the server should try to gather an RDF model from the source resource employing an RDFa parser.

3. If this fails, the server should at least verify the existence of an untyped (X)HTML link in the body of the source resource.

Depending on the category of data which was retrieved from the source resource, the server can react in different ways:

- If there is only an *untyped (X)HTML* link in the source resource, this link can be created as an RDF triple with a generic RDF property like `dc:references` or `sioc:links_to` in the servers knowledge base.

- If there is at least one *direct link* from the source resource to the target resource, this triple should be added to the servers knowledge base.

- If there is any other triple in the source resource where either the subject or the object of the triple corresponds to the target resource, the target resource can be linked using the `rdfs:seeAlso` property with the source resource.

In addition to the statements which link the source and the target resource, metadata about the source resource (e.g. a label and a description) can be stored as well.

8.5.3 *Provenance Tracking*

Provenance information can be recorded using the provenance vocabulary [**?**][6]. This vocabulary describes provenance information based on data access and data creation attributes as well as three basic provenance related types: executions, actors and artefacts. Following the specification in [**?**], we define a *creation guideline* for Pingback requests, which is described in this paper, and identified by the URI `http://purl.org/net/pingback/RequestGuideline`. A specific Pingback request *execution* is then performed by a Pingback *data creating service*, which uses the defined creation guideline.

The following listing shows an example provenance model represented in N3:

```
@prefix : <http://purl.org/net/provenance/ns#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .
@prefix pingback: <http://purl.org/net/pingback/> .

[a rdf:Statement;
 rdf:subject <http://example1.org/Source>;
 rdf:predicate sioc:links_to;
```

6 The Provenance Vocabulary Core Ontology Specification is available at `http://trdf.sourceforge.net/provenance/ns.html`.

```
10  rdf:object <http://example2.org/Target>;
11  :containedBy  [
12   a :DataItem;
13   :createdBy  [
14    a :DataCreation;
15    :performedAt "2010-02-12T12:00:00Z";
16    :performedBy  [
17     a :DataCreatingService;
18      rdfs:label "Semantic Pingback Service" ];
19      :usedData  [
20       a :DataItem;
21       :containedBy <http://example1.org/Source> ];
22     :usedGuideline [a pingback:RequestGuideline ]
23  ]];].
```

This provenance model describes a Pingback from `http://example1.org/Source` to `http://example2.org/Target`. The Pingback was performed Friday, 12 February at noon and resulted in a single statement, which links the source resource to the target resource using a `sioc:links_to` property.

8.6 RELATED WORK

Pingback [**?**] is one of three approaches which allow the automated generation of backlinks on the Social Web. We have chosen the Pingback mechanism as the foundation for this work, since it is widely used and less prone to spam than for example Trackbacks[7]. Pingback supports the propagation of untyped links only and is hence not directly applicable to the Data Web.

The PSI BackLinking Service for the Web of Data[8] supports the manual creation of backlinks on the Data Web by employing a number of large-scale knowledge bases, as for example, data of the UK Public Sector Information domain. Since it is based on crawling a fixed set of knowledge bases, it cannot be applied for the entire Data Web. Another service that amongst others is integrated with the PSI BackLinking Service is SameAs.org[9] [**?**]. Other than the Semantic Pingback it crawls the Web of Data in order to determine URIs describing the same resources. OKKAM [**?**] is a system that aims at unifying resource identifiers by employing metadata about resources in order to match them on entities.

In [**?**] the authors introduce SILK as a link discovery framework for the data web. It enables the publisher of a dataset to discover links to other datasets, by employing various similarity metrics. Since SILK adds links in the local dataset only and does not inform the publisher of the target dataset, it could be enhanced with a Semantic Pingback client.

The approaches above support interlinking of resources employing centralised hubs, but do not support decentralised, on-the-fly backlink-

7 http://www.sixapart.com/pronet/docs/trackback_spec
8 http://backlinks.psi.enakting.org
9 http://sameas.org

ing, since they are based on crawling the Data Web on a regular basis. Consequently the primary goal of these approaches is to reveal resource identifiers describing the same entities, rather than interlinking different resources - a key feature of the Semantic Pingback approach.

8.7 CONCLUSION AND FUTURE WORK

Although the Data Web is currently substantially growing, it still lacks a network effect as we could observe for example with the blogosphere in the Social Web. In particular coherence, information quality, and timeliness are still obstacles for the Data Web to become an Web-wide reality. With this work we aimed at extending and transferring the technological cornerstone of the Social Web the Pingback mechanism towards the Data Web. The resulting Semantic Pingback mechanism has the potential to significantly improve the coherence on the Data Web, since linking becomes bi-directional. With its integrated provenance and spam prevention measures it helps to increase the information quality. Notification services based on Semantic Pingback increase the timeliness of distributed data. In addition these different benefits will mutually strengthen each other. Due to its complete downwards compatibility our Semantic Pingback also bridges the gap between the Social and the Data Web. We also expect the Semantic Pingback mechanism to support the transition process from data silos to flexible, decentralised structured information assets.

8.7.0.1 *Future Work.*

Currently the Semantic Pingback mechanism is applicable to relatively static resources, i.e. RDF documents or RDFa annotated Web pages. We plan to extend the Semantic Pingback mechanism in such a way, that it is also usable in conjunction with dynamically generated views on the Data Web - i.e. SPARQL query results. This would allow end-users as well as applications using remote SPARQL endpoints to get notified once results of a query change.

Part III

APPLICATIONS

Part IV

XOPERATOR - AN INSTANT MESSAGING AGENT

Instant Messaging (IM) is in addition to Web and Email the most popular service on the Internet. With xOperator we present a strategy and implementation which deeply integrates Instant Messaging networks with the Semantic Web. The xOperator concept is based on the idea of creating an overlay network of collaborative information agents on top of social IM networks. It can be queried using a controlled and easily extensible language based on AIML templates. Such a deep integration of semantic technologies and Instant Messaging bears a number of advantages and benefits for users when compared to the separated use of Semantic Web technologies and IM, the most important ones being context awareness as well as provenance and trust. We showcase how the xOperator approach naturally facilitates contacts and calendar management as well as access to large scale heterogeneous information sources.

9 INTRODUCTION

With estimated more than 500 million users Instant Messaging (IM) is in addition to Web and Email the most popular service on the Internet. IM is used to maintain a list of close contacts (such as friends or co-workers), to synchronously communicate with those, exchange files or meet in groups for discussions. Examples of IM networks are ICQ, Skype, AIM or the Jabber protocol and network[1]. The latter is an open standard and the basis for many other IM networks such as Google Talk, Meebo and Gizmo.

While there were some proposals and first attempts to bring semantic technologies together with IM (e.g. [**? ? ?**]) in this paper we present a strategy and implementation called xOperator, which deeply integrates both realms in order to maximise benefits for prospective users. The xOperator concept is based on the idea of additionally equipping an users' IM identity with a number of information sources this user owns or trusts (e.g. his FOAF profile, iCal calendar etc.). Thus the social IM network is overlaid with a network of trusted knowledge sources. An IM user can query his local knowledge sources using a controlled (but easily extensible) language based on Artificial Intelligence Markup Language (AIML) templates[**?**]. In order to pass the generated machine interpretable queries to other xOperator agents of friends in the social IM network xOperator makes use of the standard message exchange mechanisms provided by the IM network. After evaluation of the query by the neighbouring xOperator agents results are transferred back, filtered, aggregated and presented to the querying user.

Such a deep integration of semantic technologies and IM bears a number of advantages and benefits for users when compared to the separated use of Semantic Web technologies and IM. From our point of view the two most crucial ones are:

- **Context awareness.** Users are not required to world wide uniquely identify entities, when it is clear what/who is meant from the context of their social network neighbourhood. When asked for the current whereabout of Sebastian, for example, xOperator can easily identify which person in my social network has the name Sebastian and can answer my query without the need for further clarification.
- **Provenance and trust.** IM networks represent carefully balanced networks of trust. People only admit friends and colleagues to

1 http://www.jabber.org/

their contact list, who they trust seeing their online presence, not being bothered by SPAM and sharing contact details with. Overlaying such a social network with a network for semantic knowledge sharing and querying naturally solves many issues of provenance and trust.

The paper is structured as follows: after presenting envisioned usage scenarios and requirements in Section 15.2 we exhibit the technical xOperator architecture in Section 11. We report about a first xOperator evaluation according to different use cases in Section 15.8, present related work in Section 15.9. We draw conclusions and suggest directions for future work in Section 14.

10

COMMUNICATION SCENARIOS AND REQUIREMENTS

This section describes the three envisioned agent communication scenarios for xOperator. We will introduce some real-world application scenarios also later in Section 15.8. Figure 4 shows a schematic depiction of the communication scenarios. The figure is divided vertically into four layers.

The first two layers represent the World Wide Web. Mutually interlinked RDF documents (such as FOAF documents) reference each other using relations such as `rdf:seeAlso`.[1] These RDF documents could have been generated manually, exported from databases or could be generated from other information sources. These can be, for example, mailing list archives which are represented as SIOC ontologies or personal calendars provided by public calendaring servers such as Google calendar. In order to make such information available to RDF aware tools (such as xOperator) a variety of transformation and mapping techniques can be applied. For the conversion of iCal calendar information for example we used Masahide Kanzaki's ical2rdf service[2].

The lower two layers in Figure 4 represent the Jabber Network. Here users are interacting synchronously with each other, as well as users with artificial agents (such as xOperator) and agents with each. A user can pose queries in natural language to an agent and the agent transforms the query into one or multiple SPARQL queries. Thus generated SPARQL queries can be forwarded either to a SPARQL endpoint or neighbouring agents via the IM networks transport protocol (XMPP in the case of Jabber). SPARQL endpoints evaluate the query using a local knowledge base, dynamically load RDF documents from the Web or convert Web accessible information sources into RDF. The results of SPARQL endpoints or other agents are collected, aggregated, filtered and presented to the user depending on the query as list, table or natural language response.

The different communication scenarios are described in the following subsections:

1 The prefix `rdf`, `rdfs`, `foaf` and `ical` used in this paper represent the well known namespaces (e.g. `http://xmlns.com/foaf/0.1/` for `foaf`).

2 `http://www.kanzaki.com/courier/ical2rdf`

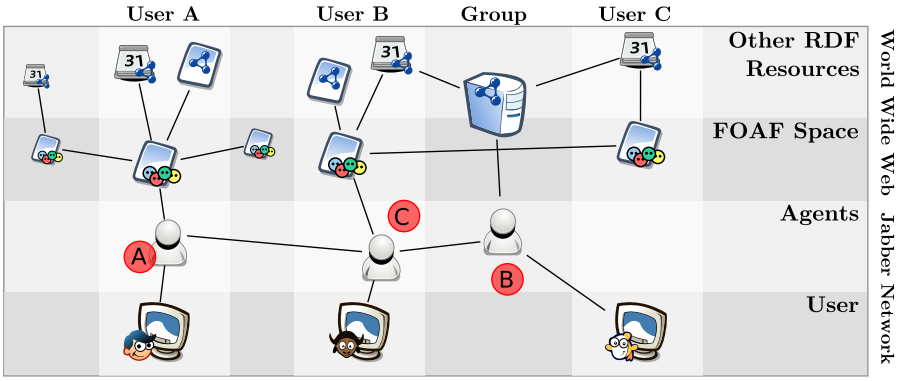

Figure 4: Agent communication scenarios: (a) personal agent, (b) group agent, (c) agent network.

10.1 PERSONAL AGENT (A)

This scenario is the most important one and also builds the foundation for the other two communication scenarios. A user of an Instant Messaging network installs his own personal agent and configures information sources he owns or trusts. For easy deployment the software representing the agent could be distributed together with (or as a plugin of) the IM network client (e.g. Pidgin). Information sources can be for example a FOAF profile of the user containing personal information about herself and about relationships to other people she knows and where to find further information about these. This information is represented in FOAF using the properties `foaf:knows` and `rdfs:seeAlso`. The following listing shows an excerpt from a FOAF profile.

```
:me a foaf:Person ;
  foaf:knows [
    a foaf:Person ;
    rdfs:seeAlso <http://eye48.com/foaf.rdf> ;
    foaf:name "Michael Haschke" ;
    foaf:nick "Haschek" ] .
```

Additionally this FOAF profile can link to other RDF documents which contain more information about the user and his activities. The RDF version of his calendar, for example, could be linked as follows:

```
:me rdfs:seeAlso <http://.../ical2rdf?u=http...> .
<http://.../ical2rdf?u=http...> a ical:Vcalendar;
  rdfs:label "Haschek's Calendar" .
```

Such links span a network of information sources as depicted in Figure 4. Each user maintains his own information and links to information sources of his acquaintances. Depending on the query, the agent will access the respective resources. The following example queries are possible, when FOAF profiles are known to the agent: Tell me the phone / homepage / ... of Frank! What is the birthday of Michael? Where is Dave now? Who knows Alex?

10.2 GROUP AGENT (B)

This communication scenario differs from the Personal Agent scenario in that multiple users get access to the same agent. The agent should be able to communicate with multiple persons at the same time and to answer queries in parallel. As is also depicted in Figure 4 the agent furthermore does not only access remote documents but can also use a triple store for answering queries. When used within a corporate setting this triple store can for example contain a directory with information about employees or customers. The triple store can be also used to cache information obtained from other sources and thus facilitates faster query answering. For agents themselves, however, the distinction between RDF sources on the Web and information contained in a local triple store is not relevant.

10.3 AGENT NETWORK (C)

This scenario extends the two previous ones by allowing communication and interaction between agents. The rationale is to exploit the trust and provenance characteristics of the Instant Messaging network: Questions about or related to acquaintances in my network of trust can best be answered by their respective agents. Hence, agents should be able to talk to other agents on the IM network. First of all, it is crucial that agents on the IM network recognize each other. A personal agent can use the IM account of its respective owner and can access the contact list (also called roster) and thus a part of its owner's social network. The agent should be able to recognise other personal agents of acquaintances in this contact list (auto discovery) and it should be possible for agents to communicate without interfering with the communication of their owners. After other agents are identified it should be possible to forward SPARQL queries (originating from a user question) to these agents, collect their answers and present them to the user.

11 TECHNICAL ARCHITECTURE

First of all, the xOperator agent is a mediator between the Jabber Instant Messaging network[1] on one side and the World Wide Web on the other side. xOperator is client in both networks. He communicates anonymously (or using configured authentication credentials) on the WWW by talking HTTP with Web servers. On the Jabber network xOperator utilizes the Extensible Messaging and Presence Protocol (XMPP, [**?**]) using the Jabber account information provided by its owner. Jabber clients only communicate with the XMPP server associated with the user account. Jabber user accounts are have the same syntax as email addresses (e.g. `soerenauer@jabber.ccc.de`). The respective Jabber server cares about routing messages to the server associated with the target account or temporarily stores the message in case the target account is not online or its server is not reachable. Since 2004 XMPP is a standard of the Internet Engineering Task Force and is widely used by various services (e.g. Google Talk). Figure 5 depicts the general technical architecture of xOperator.

The agent works essentially in two operational modi:

1. (Uninterrupted line) Answer natural language questions posed by a user using SPARQL queries and respond to the user in natural language according to a predefined template. Questions posed by a user (a) are either directly mapped to a SPARQL query template (b) or SPARQL queries are generated by a query script (c), which might obtain additional information by means of sub queries (d). The resulting SPARQL query will be evaluated on resources of the user (e), as well as passed on the Jabber network to neighbouring agents for evaluation (f). All returned results are collected and prepared by a result renderer for pre-

1 Our implementation currently supports the Jabber network, but can be easily extended to other IM networks such as Skype, ICQ or MSN Messenger

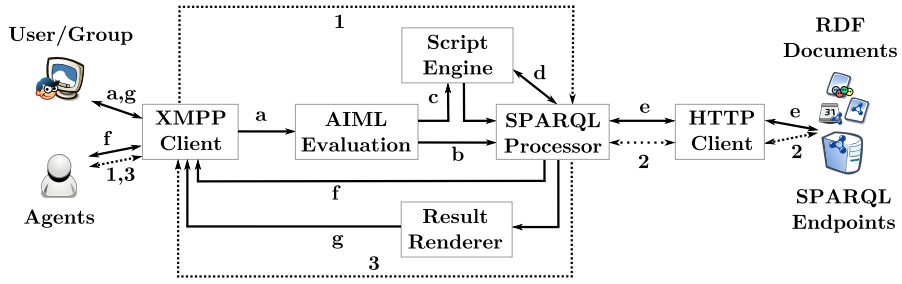

Figure 5: Technical architecture of xOperator.

sentation to the user (g). Algorithm 1 demonstrates the workings of xOperator.

Algorithm 1: Evaluation of XMPP user input.

Input: User input I from XMPP
Output: Sendable Agent response
Data: set $S = A \cup D \cup E$ of agents, documents and endpoints
Data: set C of AIML categories
Data: set $R = \emptyset$ of results

```
1  if I is an admin or extension command then return executeCommand (I)
2  else if I has no match in C then return defaultmsg
3  else if I has standard match in C then return aimlResult (I, C)
4  else
5      if I has SPARQL template match in C then
6          Query = fillPatterns (aimlResult (I, C) )
7      else if I has query script match in C then
8          Query = runScript (aimlResult (I, C) )
9  if Query then
10     foreach s ∈ S do
11         R = R ∪ executeQuery(Query, s)
12     return renderResults (R);
13 else
14     return error
```

2. (Dotted line) Receive SPARQL queries from neighbouring agents (1) on the IM network, evaluate these queries (2) on the basis of locally known RDF documents and SPARQL endpoints and send answers as XML SPARQL Result Set [**?**] back via XMPP (3).

In both cases the agent evaluates SPARQL queries by querying a remote SPARQL endpoint via HTTP `GET` Request according to the SPARQL HTTP Bindings [**?**] or by retrieving an RDF document as well via HTTP and evaluating the query by means of a local SPARQL query processor.

In the following we describe first the natural language component on the basis AIML templates and address thereafter the communication in the Jabber network.

11.1 EVALUATION OF AIML TEMPLATES

The Artificial Intelligence Markup Language (AIML, [**?**]) is an XML dialect for creating natural language software agents. In [**?**] the authors

describe AIML to enable pattern-based, stimulus-response knowledge content to be served, received and processed on the Web and offline in the manner that is presently possible with HTML and XML. AIML was designed for ease of implementation, ease of use by newcomers, and for interoperability with XML and XML derivatives such as XHTML. Software reads the AIML objects and provides application-level functionality based on their structure. The AIML interpreter is part of a larger application generically known as a bot, which carries the larger functional set of interaction based on AIML. A software module called a responder handles the human-to-bot or bot-to-bot interface work between an AIML interpreter and its object(s). In xOperator AIML is used for handling the user input received through the XMPP network and to translate it into either a query or a call to a script for more sophisticated evaluations.

The most important unit of knowledge in AIML is the category. A category consists of at least two elements, a pattern and a template element. The pattern is evaluated against the user input. If there is a match, the template is used to produce the response of the agent. It is possible to use the star ($*$) as a placeholder for any word in a pattern. We have extended this basic structure in two ways:

Simple Query Templates:

In order to enable users to create AIML categories on the fly we have created an extension of AIML. It allows to map natural language patterns to SPARQL query templates and to fill variables within those templates with parameters obtained from $*$-placeholders in the natural language patterns.

```
<category>
  <pattern>TELL ME THE PHONE OF *</pattern>
  <template>
    <external name="query"
      param="SELECT DISTINCT ?phone WHERE {...}" />
  </template>
</category>
```

Within the SPARQL template variables in the form of %%n%% refer to $*$-placeholder (n refers to the n^{th} $*$-placeholder in the category pattern). The question for the phone number of a person, for example, can be represented with the following AIML template:

```
TELL ME THE PHONE OF *
```

A possible (very simple) SPARQL template using the FOAF vocabulary could be stored within the AIML category as follows:

```
SELECT DISTINCT ?phone WHERE
    { ?s foaf:name "%%1%%". ?s foaf:phone ?phone. }
```

On activation of a natural language pattern by the AIML interpreter the corresponding SPARQL templates variables are bound to the values of the placeholders and the resulting query is send independently to all known SPARQL endpoints and neighbouring agents. These answer independently and deliver result sets, which can complement each other, contain the same or contradictory results. The agent renders results as they arrive to the user, but filters duplicates and marks contradictory information. The agent furthermore annotates results with regard to their provenance.

This adoption of AIML is easy to use and directly extensible via the Instant Messaging client (cf. Sec. 11.2). However, more complex queries, which for example join information from multiple sources are not possible. In order to enable such queries we developed another AIML extension, which allows the execution of query scripts.

Query Scripts:

Query scripts basically are small pieces of software, which run in a special environment where they have access to all relevant subsystems. They are given access to the list of known data sources and neighbouring agents. xOperator, for example, allows the execution of query scripts in the Groovy scripting language for Java.The execution of a query script results in the generation of a SPARQL query, which is evaluated against local information sources and passed to other agents as described in the previous section. We motivate and illustrate the workings of query scripts using an application scenario based on the FOAF space (cf. Figure 4). The FOAF space has the following characteristics:

- The `foaf:knows` relation points to other people known by this person.
- Other FOAF and RDF documents are linked through `rdfs:seeAlso`, allowing bots and agents to crawl through the FOAF space and to gather additional RDF documents like calendars or blog feeds.

To enable the agent to retrieve and evaluate additional information from sources which are referenced from the user's FOAF profile, a query script can contain subqueries, whose results are used within another query. Query scripts also enable the usage of special placeholders such as `now` or `tomorrow`, which can be populated for the querying of iCal calendars with the concrete values.

In order to extend the agent for other application domains or usage scenarios, xOperator allows to dynamically assign new query scripts to AIML categories. A query script is assigned to an AIML template by means of an `external` tag (as are also simple SPARQL templates). An example script implementing a subquery to retrieve relevant resources about a `foaf:person` is presented in Section 15.8.

11.2 ADMINISTRATION AND EXTENSION COMMANDS

Users can easily change the configuration of their agents by using a set of administration and extension commands. These commands have a fix syntax and are executed without the AIML engine:

- `list ds`, `add ds {name} {uri}`, `del ds {name}`: With these commands, users can manage their trusted data sources. Each source is locally identified by a name which is associated to an URI.
- `list templates`, `add template {pattern} {query}`, `del template {pattern}`: With these template commands, users can manage simple query templates which are associateded by its AIML pattern.
- `query {sparql query}`: This command is used to send on-the-fly SPARQL queries to the xOperator. The query will be evaluated on every datastore and routed to every agent in the neighbourhood. The query results will be rendered by a default renderer.
- `list ns`, `add ns {prefix} {uri}`, `del ns {prefix}`: To easlily create on-the-fly SPARQL queries, users can manage namespaces. The namespaces will be added to the namespace section in the on-the-fly query.
- `help`: This is an entry point for the help system.

11.3 XMPP COMMUNICATION AND BEHAVIOUR

While the HTTP client of the agent uses standard HTTP for querying SPARQL endpoints and the retrieval of RDF documents, we extended XMPP for the mutual communication between the agents. This extension complies with standard extension routines of XMPP will be ignored by other agents. With regard to the IM network the following functionality is required:

- The owner of the agent should be able to communicate easily with the agent. He should be able to manage the agent using the contact list (roster) and the agent should be easily recognizable.
- The agent has to have access to the roster of its owner in order to identify neighbouring agents.
- It should be possible for other agents to obtain information about the ownership of an agent. His requests will not be handled by other agents for security reasons if he can not be clearly assigned to an owner.
- The agent should be only visible for his owner and neighbouring agents (i.e. agents of contacts of his owner) and only accept queries from these.

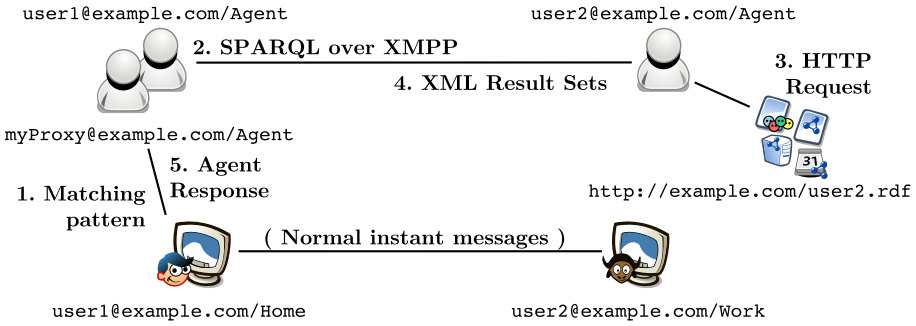

Figure 6: XMPP Communication example

As a consequence from those requirements it is reasonable that the agent acts using the account of its owner (main account) for the communication with other agents, as well as an additional account (proxy account) for the communication with its owner[2]. Due to the usage of the main account other agents can trust the agents answers and easily track the provenance of query results. Figure 6 depicts the concept of using two different accounts for the communication with the owner and other agents. For unique identification of senders and recipients so called resource names (in the figure `Home`, `Work` and `Agent`) are used and simply appended to the account name.

We demonstrate the agent communication with two XMPP messages:

Agent Autodiscovery:

Goal of the autodiscovery is the identification of agents among each other. For that purpose each agent sends a special message of type info/query (`iq`) to all known and currently active peers. Info/query messages are intended for internal communication and queries among IM clients without being displayed to the human users. An autodiscovery message between the two agents from Figure 6, for example, would look as follows:

```
<iq from="user1@example.com/Agent" type='get'
    to="user2@example.com/Agent" id='...'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

A positive response to this feature discovery message from an xOperator agent would contain a feature with resource ID `http://www.w3.org/2005/09/xmpp-sparql-binding`. This experimental identifier/namespace was created by Dan Brickley for SPARQL / XMPP experiments (cf. Section 15.9). The response message to the previous request would look as follows:

2 Technically, it is sufficient for the agent to use the owner's account which, however, could create confusing situations for the user when communicating with 'herself'.

```
<iq from='user2@example.com/Agent' type='result'
    to='user1@example.com/Agent' id='...' />
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
      category='client' name='xOperator' type='bot'/>
    <feature
      var='http://www.w3.org/2005/09/xmpp-sparql-binding'/>
    <!-- ... more here -->
  </query>
</iq>
```

Similar XMPP messages are used for sending SPARQL queries and retrieving results. The latter are embedded into a respective XMPP message according to the SPARQL Query Results XML Format[3].

Routing and Recall.

Queries are propagated to all neighbouring xOperator agents. As currently there is no way of anticipating which agent could answer a question, asking all directly connected agents offers the best compromise between load and recall. Flooding the network beyond adjacent nodes would cause excessive load. Especially in the domain of personal information, persons or their respective agents directly related to the querying person or agent should be most likely to answer the query.

3 http://www.w3.org/TR/rdf-sparql-XMLres/

12 EVALUATION

The xOperator concept was implemented in Java and is available as open-source software from: http://aksw.org/Projects/xOperator. The agent is able to log into existing accounts and can receive querying and configuration commands.

Figure 7: Communication with the xOperator agent by means of an ordinary Jabber client.

We evaluated our approach in a number of scenarios, which included various heterogeneous information sources and a different number of agents. As information sources we used FOAF profiles (20 documents, describing 50 people), the SPARQL endpoint of our semantic Wiki OntoWiki [**?**] (containing information about publications and projects), information stored in the LDAP directory service of our department, iCal calendars of group members from Google calendar (which are accessed using iCal2RDF) and publicly available SPARQL endpoints such as DBpedia [**?**]. Hence the resulting information space contains information about people, groups, organizations, relationships, events, locations and all information contained in the multidomain ontology DBpedia. We created a number of AIML categories, interacting with this information space. Some example patterns and corresponding timings for obtaining answers from the agent network in the three different network scenarios (personal agent, agent network and group agent) are summarized in Table 2.

The first three templates represent queries which are answered using simple SPARQL templates. Template 4 makes use of a reserved word (`now`), which is replaced for querying with an actual value. Template 5

	Template	Scenario:	1	2	3
1	What is / Tell me (the) * of *		2.3	3.9	1.5
2	Who is member of *		3.5	4.3	1.6
3	Tell me more about *		3.2	5.6	1.1
4	Where is * now:		5.1	6.7	4.2
5	Free dates * between * and *		5.1	6.8	4.7
6	Which airports are near *		–	–	3.4

Table 2: Average response time in seconds (client to client) of some AIML patterns used in three scenarios: (1) 20 documents linked from one FOAF profile, 1 personal agent with no neighbourhood (2) 20 documents linked from different FOAF profiles and spread over a neighbourhood of 5 agents (3) one SPARQL endpoint as an interface to a Semantic Wiki or DBpedia store with one group agent

is implemented by means of a query script which retrieves all available time slots from the calendars of a group of people and calculates the intersection thus offering suitable times to arrange meetings or events, where the attendance of all group members is required. Template 6 uses the DBpedia SPARQL endpoint in a group agent setting to answer questions about the geographic location of places (such as airports). These query templates are meant to give some insights in the wealth of opportunities for employing xOperator. Further, AIML templates can be created easily, even directly from within the IM client (using the administration and extension commands as presented in Section 11.2).

A typical user session showing the communication with the agent is depicted in Figure 7. The response timings indicate that the major factor are latency times for retrieving RDF documents or querying SPARQL endpoints. The impact of the number of agents in the agent network as well as the overhead required by the xOperator algorithm is rather small. The timings are furthermore upper bounds, since answers are presented to the user as they arrive. This results in intuitive perception that xOperator is a very responsive and efficient way for query answering.

Experiences during the evaluation have led to the following rules for creating patterns and queries in xOperator.

(1) Query as fuzzy as possible: Instant Messaging is a very quick means of communication. Users usually do not capitalize words and use many abbreviations. This should be considered, when designing suitable AIML patterns. If information about the person 'Sören Auer' should be retrieved, this can be achieved using the following graph pattern: `?subject foaf:name "Auer".` However, information can be represented in multiple ways and often we have to deal with minor

misrepresentations (such as trailing whitespace or wrong capitalizations), which would result for the above query to fail. Hence, less strict query clauses should be used instead. For the mentioned example the following relaxed SPARQL clause, which matches also substrings and is case insensitive, could be used:

```
?subject foaf:name ?name.
FILTER regex(?name,'.*Auer.*','i')
```

(2) Use patterns instead of qualified identifiers for properties: Similar, as for the identification of objects, properties should be matched flexible. When searching for the `homepage` of 'Sören Auer' we can add an additional property matching clause to the SPARQL query instead of directly using, for example, the property identifier `foaf:homepage`):

```
?subject ?slabel ?spattern.
?subject ?property ?value.
?property ?plabel ?ppattern.
FILTER regex(?spattern,'.*Auer.*','i')
FILTER regex(?ppattern,'.*homepage.*','i')
```

This also enables multilingual querying if the vocabulary contains the respective multilingual descriptions. Creating fuzzy queries, of course, significantly increases the complexity of queries and will result in slower query answering by the respective SPARQL endpoint. However, since we deal with a distributed network of endpoints, where each one only stores relatively small documents this effect is often negligible.

(3) Use sub queries for additional documents: In order to avoid situations where multiple agents retrieve the same documents (which is very probable in a small worlds scenario with a high degree of interconnectedness) it is reasonable to create query scripts, which only distribute certain tasks to the agent network (such as the retrieval of prospective information sources or document locations), but perform the actual querying just once locally.

13

RELATED WORK

Proposals and first prototypes which are closely related to xOperator and inspired its development are Dan Brickley's JQbus[1] and Chris Schmidt's SPARQL over XMPP[2]. However, both works are limited to the pure transportation of SPARQL queries over XMPP.

Quite different but the xOperator approach nicely complementing are works regarding the semantic annotation of IM messages. In [**?**] for example the authors present a semantic archive for XMPP instant messaging which facilitates search in IM message archives. [**?**] suggests ways to make IM more semantics aware by facilitating the classification of IM messages, the exploitation of semantically represented context information and adding of semantic meta-data to messages. Comprehensive collaboration frameworks which include semantic annotations of messages and people, topics are, for example, CoAKTinG [**?**] and Haystack [**?**]. The latter is a general purpose information management tool for end users and includes an instant messaging component, which allows to semantically annotate messages according to a unified abstraction for messaging on the Semantic Web[**?**].

In [**?**] natural language interfaces (NLIs) are used for querying semantic data. The NLI used in xOpertor employs only a few natural language processing techniques, like stop word removal for better template matching. Generic templates would be possible to define, but as [**?**] shows user interaction is necessary for clarifying ambiguities. For keeping IM conversation as simple as possible, domain specific templates using AIML were chosen. Finally, in [**?**] the author enhanced AIML bots by generating AIML categories from RDF models. Different to xOperator, these categories are static and represent only a fixed set of statements.

1 http://svn.foaf-project.org/foaftown/jqbus/intro.html
2 http://crschmidt.net/semweb/sparqlxmpp/

14

CONCLUSIONS AND FUTURE WORK

With the xOperator concept and its implementation, we have showed how a deeply and synergistic coupling of Semantic Web technology and Instant Messaging networks can be achieved. The approach naturally combines the well-balanced trust and provenance characteristics of IM networks with semantic representations and query answering of the Semantic Web. The xOperator approach goes significantly beyond existing work which mainly focused either on the semantic annotation of IM messages or on using IM networks solely as transport layers for SPARQL queries. xOperator on the other hand overlays the IM network with a network of personal (and group) agents, which have access to knowledge bases and Web resources of their respective owners. The neighbourhood of a user in the network can be easily queried by asking questions in a subset of natural language. By that xOperator resembles knowledge sharing and exchange in offline communities, such as a group of co-workers or friends. We have showcased how the xOperator approach naturally facilitates contacts and calendar management as well as access to large scale heterogeneous information sources. In addition to that, its extensible design allows a straightforward and effortless adoption to many other application scenarios such as, for example, sharing of experiment results in Biomedicine or sharing of account information in Customer Relationship Management.

In addition to adopting xOperator to new domain application we view the xOperator architecture as a solid basis for further technological integration of IM networks and the Semantic Web. This could include adding light-weight reasoning capabilities to xOperator or the automatic creation of AIML categories by applying NLP techniques. A more fine grained access control will be implemented in a future version. Instead of simply trusting all contacts on the roster, individual and group based policies can be created. An issue for further research is the implementation of a more sophisticated routing protocol, that allows query traversal beyond directly connected nodes without flooding the whole network.

15

MSSW – A MOBILE DSSN CLIENT

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 only for centralized proprietary platforms (such as Google mail, contacts & calendar) as well as data-silo-like social networks (e.g. Facebook). Within the Semantic Web initiative standards and best-practices for social, Semantic Web applications such as FOAF emerged. However, there is no comprehensive strategy, how these technologies can be used efficiently in a mobile environment. In this paper we present the architecture as well as the implementation of a mobile Social Semantic Web framework, which weaves a distributed social network based on semantic technologies.

15.1 INTRODUCTION

[ToDo: Acknowledge for seppl and micha]

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 [**?**].

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* (cf. Section 15.9). 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 paper 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 Section 15.2), 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 (Section 8.3),
- We develop an architecture for making mobile devices endpoints for the Social Semantic Web (Section 8.3),
- A comprehensive implementation of the architecture was performed for the Android platform (Section 15.3 and 15.8).

Furthermore, our paper contains an overview on related work in Section 15.9 and concludes in Section 15.10 with a discussion and outlook on future work.

15.2 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[1] 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[2]. where people want to permit and deny access to specific information in fine grained shades and based on groups, live contexts and individuals.

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

1 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.

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

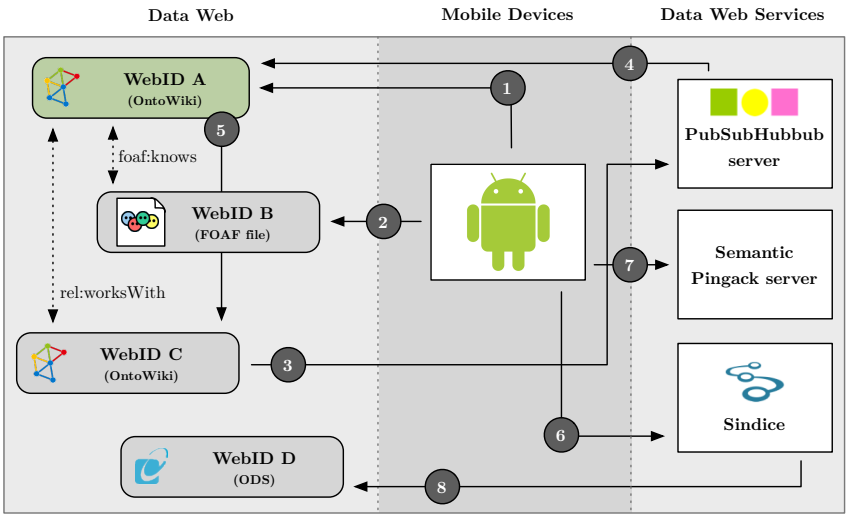

Figure 8: Mobile usecase centered DSSN architecture: (1) A mobile user may retrieve updates from his social network via his WebID provider, e.g. from OntoWiki. (2) He may also fetch updates directly from the sources of the connected WebIDs. (3) A WebID provider can notify a subscription service, e.g. 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.

low end profiles, there should be no other required feature on a WebID than the availability as Linked Data[3]. 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.

3 In the meaning of Linked RDF Data defined at http://www.w3.org/DesignIssues/LinkedData.

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.

15.3 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.

15.4 ANDROID SYSTEM INTEGRATION

Figure 9 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*[4] is one of those libraries, which itself is a partial port of the popular Jena framework[5] 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`).

15.5 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 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 15.6).

4 http://code.google.com/p/androjena/
5 http://jena.sourceforge.net/

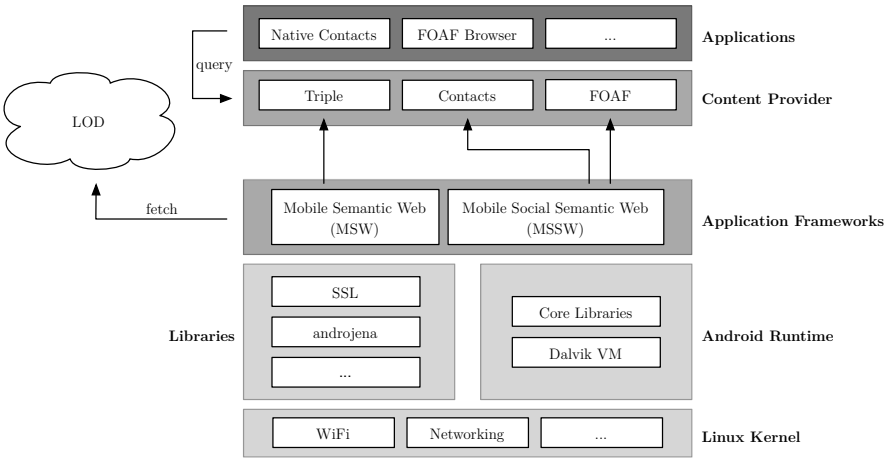

Figure 9: Android Integration Layer Cake

- `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.

15.6 RULES AND DATA PROCESSING

One of our initial requirements from Section 15.2 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[6].

Since we also did not want our implementation to depend on the FOAF vocabulary (alternative solutions include RDF vCards [**?**]),

6 http://jena.sourceforge.net/inference/#RULEsyntax

```
1  @prefix foaf: <http://xmlns.com/foaf/0.1/>.
2  @prefix android: <http://ns.aksw.org/Android/>.
3  @prefix acontacts: <http://ns.aksw.org/Android/ContactsContract.CommonDataKinds.>.
4  @prefix im: <http://ns.aksw.org/Android/ContactsContract.CommonDataKinds.Im.>.
5
6  [jabber:
7    (?s foaf:jabberID ?o), makeTemp(?d) ->
8      (?s android:hasData ?d),
9      (?d rdf:type acontacts:Im),
10     (?d im:DATA ?o),
11     (?d im:TYPE im:TYPE_HOME),
12     (?d im:PROTOCOL im:PROTOCOL_JABBER)
13 ]
```

Listing 5: 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).

we decided to create a native Android system vocabulary which represents the Android contacts database defined by the Android APIṪhis 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[7].

Based on this vocabulary, the given rules transform the downloaded WebID statements into Android-specific structures which are well suited for a straightforward 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 WebIDȦn example rule which creates an instant messaging account for the contact is presented in Listing 5.

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.

15.7 ONTOWIKI

The mobile Semantic Web client supports arbitrary WebIDs, even those backed by plain RDF files. Nevertheless, some features require special

7 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/`).

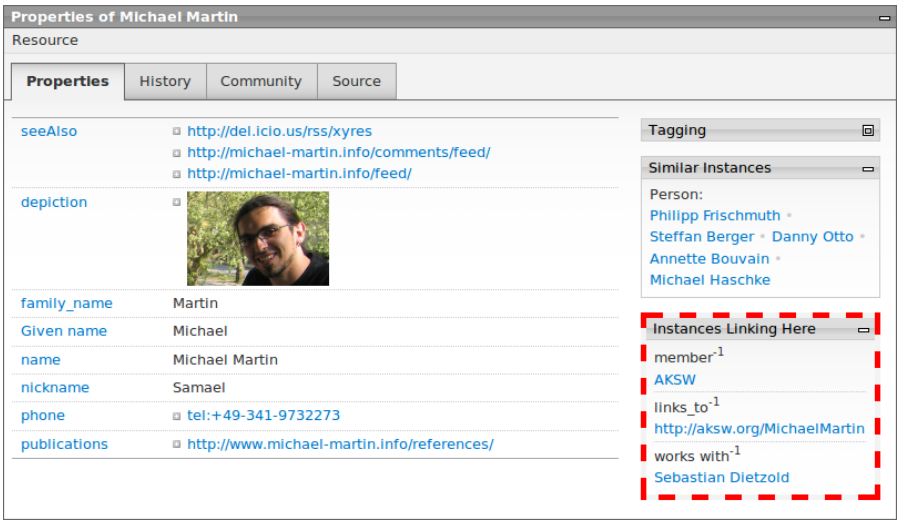

Figure 10: Visualization of a WebID in OntoWiki: incoming backlinks (via Semantic Pingback) are rendered in the "Instances Linking Here" side box.

support on the server-side. For our semantic data wiki OntoWiki [**?**] 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[8]. From the 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 10). 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.

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

Hence, editing functionality is another important aspect and OntoWiki supports editing via SPARQL/Update.

15.8 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 11). 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 11 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 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.

15.9 RELATED WORK

Related work can be roughly divided into semantics-based (but centralized) social network services, distributed social network projects, mobile Semantic Web projects and mobile social network clients. A comprehensive overview is contained in the final report of the W3C Social Web Incubator Group [**?**]. In the sequel we present some related approaches along the four dimensions in more detail.

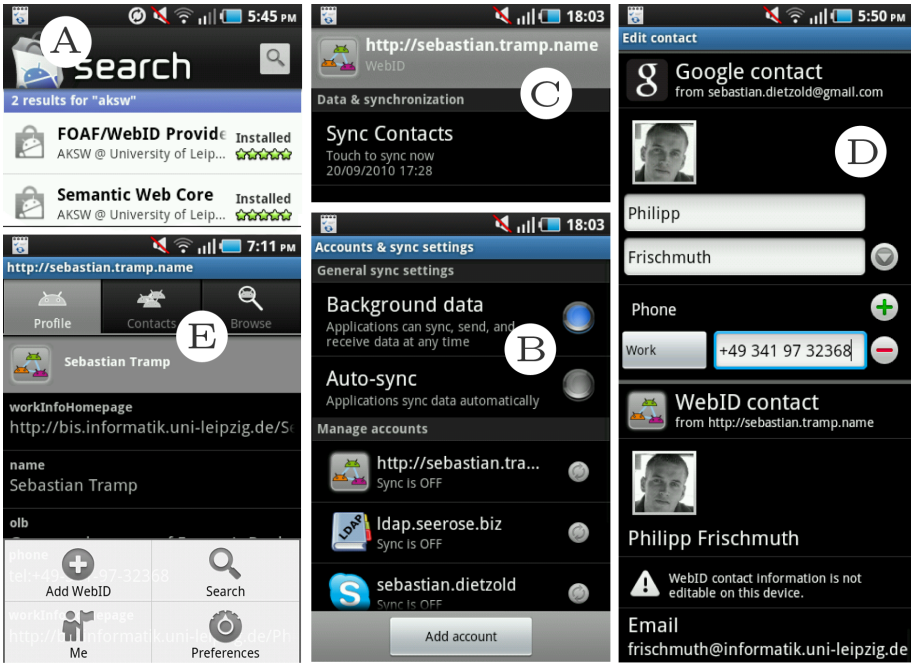

Figure 11: Screenshots of the Mobile Social Semantic Web Client, the FOAF Browser and the Android components which integrate the WebID account: (A) The client as well as the triple store can be found in the official Google application market. (B) After installation, users can add a WebID account the 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.

SEMANTICS-BASED (BUT CENTRALIZED) SOCIAL NETWORK SERVICES. Evri[9] is a centralized social network based on RDF and mainly used for collaborative information storage. Evri also maintains mobile applications for iOS and Android to give their users access to these information.

DISTRIBUTED SOCIAL NETWORK. The idea of distributed social networks appeared quickly after social networks became popular. A distributed social network traditionally refers to an Internet social network service that is decentralized and distributed across different providers, with emphasis on portability and interoperability. The most prominent representatives are Diaspora[10], NoseRub[11] and GNU social[12]. Although some of these networks make use of vocabularies (e.g. FOAF in the case of GNU Social) or certain elements of semantic

9 http://www.evri.com (formally know as Twine)
10 https://joindiaspora.com
11 http://noserub.com
12 http://www.gnu.org/software/social/

technologies, their use of Semantic Web technologies and best-practices is rather limited.

MOBILE SEMANTIC WEB APPLICATIONS. The application of Semantic Web technologies on mobile devices is not new - one of the earlier works dates back to 2003 [**?**]. However, this research area was not pursued very actively due to the large number of mobile device limitations common in that era. In the light of increasing processing power and data connectivity of modern mobile devices, the use of mobile Semantic Web technologies is becoming more feasible. There are a few publications which review the state of the mobile Semantic Web and the possibilities thereof to improve mobile Web (e.g. [**?**]). Also, there are publications on more complex systems, such as *SmartWeb* [**?**] which uses semantic technologies to enhance the backends of mobile web service. On the frontend side there are semantic mobile applications like *DBpedia Mobile*[**?**] or *mSpace Mobile* [**?**] that implement or utilize Semantic Web technologies directly on mobile devices. However, these frontend applications focus on very specific use cases - information about points of interest in the case of DBpedia Mobile and information for university students in the case of mSpace. Finally, there are publications which describe proof of concept applications as well as algorithms for consuming and replicating Linked Data and RDF in general [**? ? ?**].

MOBILE SOCIAL NETWORKING CLIENTS. All major social networking services (such as Facebook, Myspace, LinkedIn etc.[13]) have meanwhile clients for different mobile platforms, which are more or less integrated with the mobile phone platform itself. In addition to this, the Android market place lists more than 3500 applications in the category social networks with our implementation being one of them. However, up to our knowledge our MSSW client is the first to consequently employ W3C standards as well as Social Semantic Web best practices with regard to all aspects of data representation and integration.

15.10 CONCLUSION AND FUTURE WORK

We see the work described in this article 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

13 An ordered list can obtained from Wikipedia: http://en.wikipedia.org/wiki/List_of_social_networking_websites

Web standards, technologies and best practices can be integrated into a comprehensive architecture for social networking (on mobile devices).

With regard to future work we plan to further decrease 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 WebIDİn 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. 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.

Part V

APPENDIX

A

CURRICULUM VITÆ

PERSONAL DATA

Sebastian Tramp, geb. Dietzold

Born on 29.09.1977

http://sebastian.tramp.name[1]

April 15, 2013

A.1 COMMUNITY SERVICES

A.1.1 *Organizing Committee*

- IMC-SSW2008: International Workshop on Interacting with Multimedia Content in the Social Semantic Web; in conjunction with the 3rd International Conference on Semantic and Digital Media Technologies (SAMT 2008); December 03, 2008; Koblenz, Germany

A.1.2 *Research Program Committee*

- SDoW2009: 2nd Social Data on the Web workshop; co-located with the 8th International Semantic Web Conference (ISWC2009); Washington DC (USA); October 25, 2009
- ESIW 2009: Workshop on Exploiting Structured Information on the Web; in conjunction with 10th International Conference on Web Information Systems Engineering (WISE 2009); Poznań, Poland; October 5–7, 2009
- SFSW2009: 5th Workshop on Scripting and Development for the Semantic Web; Colocated with ESWC 2009; May/June, 2009; Crete, Greece
- SemWiki2009: Fourth Workshop on Semantic Wikis, co-located with the 6th Annual European Semantic Web Conference (ESWC2009) in Crete, Greece
- SAW2009: 3rd Workshop on Social Aspects of the Web; in conjunction with the 12th International Conference on Business

1 Usable as WebID

Information Systems (BIS 2009); April 27,28 or 29, 2009; Poznan, Poland

- SDoW2008: 1st Social Data on the Web Workshop; co-located with the 7th International Semantic Web Conference (ISWC2008); October 26/27, 2008; Karlsruhe, Germany
- SemWiki2008: 3rd Semantic Wiki Workshop; co-located with the 5th European Semantic Web Conference (ESWC2008); June 02, 2008; Tenerife, Spain
- SFSW2008: 4th Workshop on Scripting for the Semantic Web; co-located with the 5th European Semantic Web Conference (ESWC2008); June 01, 2008; Tenerife, Spain

A.1.3 *Reviewing*

- Springer Science+Business Media, Publishing Group, Heidelberg; Book in Computer Science
- IJSWIS: International Journal On Semantic Web and Information Systems, Special Issue on Linked Data
- ESWC2009: The 6th Annual European Semantic Web Conference – Demonstrations Track; 31 May – 4 June 2009, Heraklion, Greece
- LDOW2008: Linked Data on the Web; Workshop at the 17th International World Wide Web Conference (WWW2008); April 22, 2008; Beijing, China
- CSSW2007: Conference on Social Semantic Web; September 26–28, 2007; Leipzig, Germany

A.2 SEMINARS AND TEACHING

- Softwaretechnik-Praktikum (Sommersemester 2009)
- Semantik Web Praktikum (Sommersemester 2009)
- Schwerpunktpraktikum Semantische Technologien / Social Semantic Web (Wintersemester 2008/2009)
- Semantische Unterstützung von Software Entwicklungsprozessen (Sommersemester 2008)
- Softwaretechnik-Praktikum (Sommersemester 2008)
- Entwicklung von Semantischen Webanwendungen (Sommersemester 2007)
- Semantic Web Services und Interfaces (Wintersemester 2006/2007)

- Agiles Software and Knowledge Engineering (Sommersemester 2006)

A.3 SUPERVISION

Bachelor

- Jonas Brekle: Ein SPARQL Query Komponente für die API-basierte Manipulation von SPARQL Abfragen
- Leszek Kotas: Evaluation und Erweiterung des OntoWiki Plugin-Systems
- Maria Moritz: Integration des DL-Learners in OntoWiki
- Thanh Nghia Lam: SPARQL Templates für die Darstellung von RDF Inhalten
- Thomas Sobik: Management von Standard Operation Procedures
- Atanas Alexandrov: Implementierung eines Oracle basiertem RDF Store Backends
- Martin Peklo: Semantische Supportdatenbank

Master

- Rolland Brunec: Ein modularer und backend-unabhängiger SPARQL / SPARUL Parser für PHP
- Atanas Alexandrov: Tagging in Semantischen Daten-Wikis
- Martin Peklo: JavaScript API zur Manipulation von RDF Modellen
- Jörg Unbehauen: Konzept und Implementierung eines Semantischen Agenten
- Thomas Kappel: Integration von Sozialen Netzwerken mit dem Sozializr
- Christian Lehmann: Implementierung eines D2RQ Mapping Backends für RAP

Diploma

- Elena Kop: Eine Semantic Web Ontologie für den Gasmarkt
- Christoph Riess: Evaluations-Pattern in Semantic Web Wissensbasen

- Philipp Frischmuth: Interwiki-Kommunikation in semantischen Daten-Wikis
- Nomunbilegt Batsukh: Unterstützung von Online Shop Systemen mit Semantischen Technologien
- Feng Qiu: Semantisches Plugin Repository
- Norman Heino: Intelligente RDFa-Widgets
- Michael Haschke: Semantic Personal Knowledge Management mit OntoWiki
- Stefan Berger: Access Control on Triple Stores
- Enrico Popp: WebDAV Dateisystem für RDF Modelle
- Denis Gärtner: LDAP Query to SPARQL Transformation

COLOPHON

This thesis was typeset with LaTeX 2ε using Hermann Zapf's *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL* were used). The listings are typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.)

The typographic style was inspired by **?** 's genius as presented in *The Elements of Typographic Style* [**?**]. It is available for LaTeX via CTAN as "`classicthesis`".

NOTE: The custom size of the textblock was calculated using the directions given by Mr. Bringhurst (pages 26–29 and 175/176). 10 pt Palatino needs 133.21 pt for the string "abcdefghijklmnopqrstuvwxyz". This yields a good line length between 24–26 pc (288–312 pt). Using a "*double square textblock*" with a 1:2 ratio this results in a textblock of 312:624 pt (which includes the headline in this design). A good alternative would be the "*golden section textblock*" with a ratio of 1:1.62, here 312:505.44 pt. For comparison, `DIV9` of the `typearea` package results in a line length of 389 pt (32.4 pc), which is by far too long. However, this information will only be of interest for hardcore pseudo-typographers like me.

To make your own calculations, use the following commands and look up the corresponding lengths in the book:

```
\settowidth{\abcd}{abcdefghijklmnopqrstuvwxyz}
\the\abcd\ % prints the value of the length
```

Please see the file `classicthesis.sty` for some precalculated values for Palatino and Minion.

145.86469pt

DECLARATION

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, Oktober 2011

Sebastian Tramp