Semantic Web 0 (2012) 1–0
IOS Press

An Architecture of a Distributed Semantic Social Network

Editor(s): Fabian Abel, Delft University of Technology, The Netherlands; Laura Hollink, Delft University of Technology, The Netherlands; Geert-Jan Houben, Delft University of Technology, The Netherlands
Solicited review(s): Fabian Abel, Delft University of Technology, The Netherlands; Laura Hollink, Delft University of Technology, The Netherlands; one anonymous reviewer

Sebastian Tramp *, Philipp Frischmuth, Timofey Ermilov, Saeedeh Shekarpour and Sören Auer
Universität Leipzig, Institut für Informatik, AKSW, Postfach 100920, D-04009 Leipzig, Germany
E-mail: {lastname}@informatik.uni-leipzig.de

Abstract. Online social networking has become one of the most popular services on the Web. However, current social networks are like walled gardens in which users do not have full control over their data, are bound to specific usage terms of the social network operator and suffer from a lock-in effect due to the lack of interoperability and standards compliance between social networks. In this paper we propose an architecture for an open, distributed social network, which is built solely on Semantic Web standards and emerging best practices. Our architecture combines vocabularies and protocols such as WebID, FOAF, Semantic Pingback and PubSubHubbub into a coherent distributed semantic social network, which is capable to provide all crucial functionalities known from centralized social networks. We present our reference implementation, which utilizes the OntoWiki application framework and take this framework as the basis for an extensive evaluation. Our results show that a distributed social network is feasible, while it also avoids the limitations of centralized solutions.

Keywords: Distributed Social Networks, Social Semantic Web, Architecture, Evaluation, WebID, Semantic Pingback

1. Introduction

Online social networking has become one of the most popular services on the Web. Especially Facebook with its 845M+ monthly active users and 100B+ 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 cur-

*Corresponding author.
WebID: http://sebastian.tramp.name
[1] http://www.sec.gov/Archives/edgar/data/1326801/000119312512034517/d287954ds1.htm
[2] http://allthingsd.com/20110623/the-web-is-shrinking-now-what/

rently 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, 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 easy 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 themselves, 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 paper 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 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.

This article is structured as follows: We present our reference architecture of a distributed social semantic network in Section 2. Our implementation based on the OntoWiki framework is demonstrated in Section 3, while the evaluation results of our approach are discussed in Section 4. Finally, we give an overview of related work in Section 5 and conclude with an outlook on future work in Section 6.

[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`

2. Architecture of a Distributed Semantic Social Network

In this section, 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.

2.1. 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 [4]. 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 [9]. 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.

2.2. 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.

2.2.1. 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 [20][4] 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[5]. 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 [7] as the fundamental 'industry standard' which can be extended[6]. An example WebID profile comprising some personal information (lines 8-12) and two `rel:worksWith`[7] links to co-workers (lines 6-7) is shown in Listing 1.

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 2.3.2.

In addition to this service, we introduce access delegation for the WebID protocol. WebID access delega-

[4] Formerly know as the FOAF+SSL best practice [21], the latest specification is available at `http://webid.info/spec/`.

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

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

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

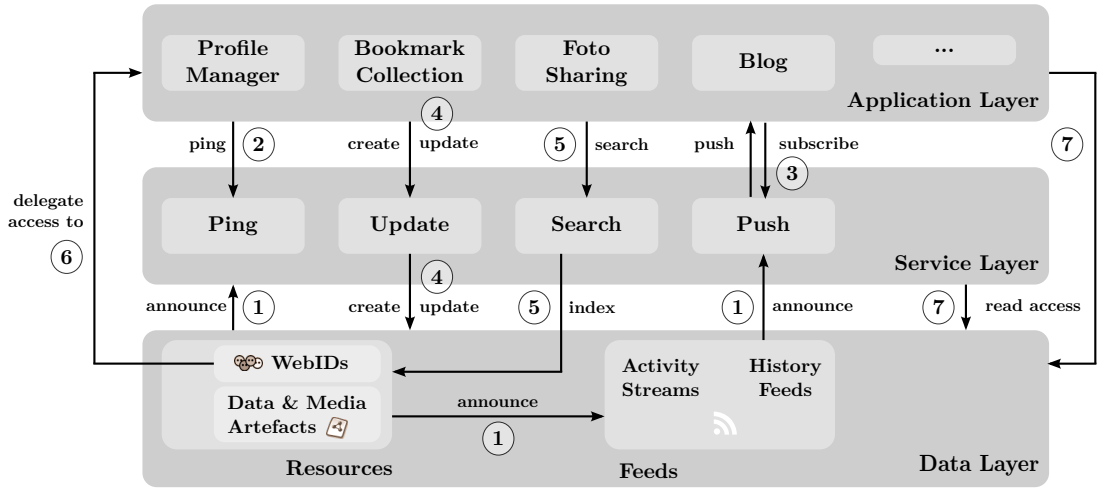

Fig. 1. Architecture of a Distributed Semantic Social Network (without protocol layer): (1) Resources announce services and feeds via links or header fields, feeds announce services – in particular a push service. (2) Applications initiate ping requests to spin the Linked Data Network. (3) Applications 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.

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

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

tion 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 WebID. We describe the WebID protocol as well as our access delegation extension in detail in Section 2.3.1.

Agents and Applications play an important role in today's social networks[8]. 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 2.3.1.

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 [6], Common Tag[9] or Activity Streams in RDF [11].

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

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

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

a meta-data part which describes this artefact[10]. 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 2.3.2, and a link to a push-enabled activity stream. An example photo-sharing application is described in Section 2.5.

2.2.2. 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 specific object)[11]. 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).

[10]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.

[11]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.

2.3. 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.

2.3.1. WebID (protocol)

From a more technical perspective, the WebID protocol [20] 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*[12] 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* [18]). An example of a WebID profile, which is annotated with a public key, is shown in Listing 2. This WebID profile contains 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

[12]`http://www.ietf.org/rfc/rfc2459.txt`

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[13]. The relying party then fetches the WebID and checks for a statement as shown in Listing 3[14].

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[15], where an API provider needs to serve and handle access tokens, a user only has to maintain one single central resource, her WebID.

[13] 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.

[14] 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/`.

[15] `http://oauth.net/`

```
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.

2.3.2. Semantic Pingback

The purpose of *Semantic Pingback* [23] 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 [10], 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 bidirectional links between WebIDs, RDF resources as well as weblogs and websites in general (cf. Figure 1). 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[16] 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 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

[16] In fact, we experimented with different service endpoints. Based on the results, which are described in more detail in [22], we now prefer simple HTTP post requests which are not compatible with standard XML-RPC pingbacks.

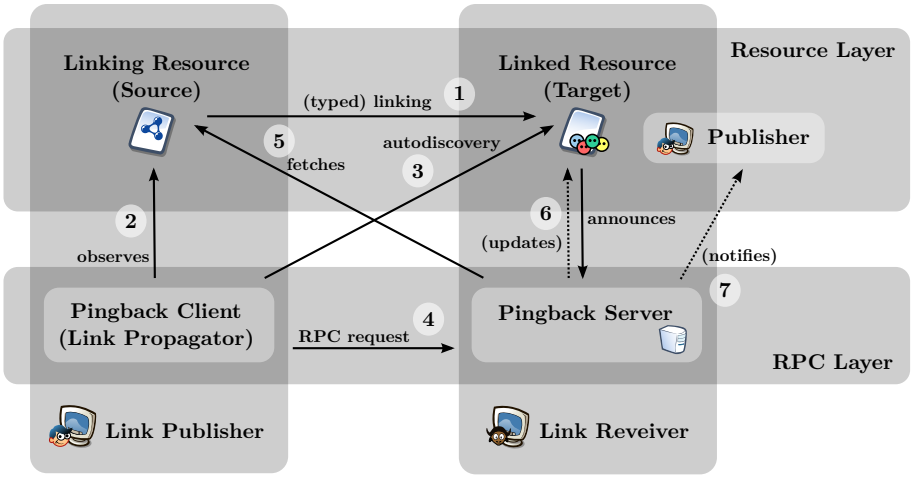

Fig. 2. Architecture of the Semantic Pingback approach: (1) A *linking resource* links to another (Data) Web resource, here called *linked resource*. (2) The *Pingback client* is either integrated into the data/content management system or realized as a separate service, which observes changes of the 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 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[17].

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[18]. 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 3 for more

```
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.

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

[18] 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#`).

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 any other resource)
sioc:Post	sioc:reply_of	sioc:Post	replying to a (friends) post
ctag:Tag	ctag:tagged	*	any resource is tagged by a user
aair:Activity	aair:activityObject	*	any resource is object of an activity

details). However, a specific reaction is not enforced by the protocol.

2.3.3. PubSubHubbub

PubSubHubbub[19] 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[20]. 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 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 [2] based on the work previously published by the Triplify project[21].

2.4. 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 1, we have defined four essential services for a DSSN.

[19] http://code.google.com/p/pubsubhubbub/

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

[21] http://triplify.org

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.

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 subproperties of the more generic `sioc:feed` relation from the SIOC project [6]: `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 [8] or Sindice [25]. 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[22].
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. 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[23]

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[24].

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.

[22] 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).

[23] 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.

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

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

2.5. 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[25]. To illustrate how DSSN applications work with a WebID and its services, we will describe a simple photo-sharing application:

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 metadata 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[26]. 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.

[25] 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.

[26] 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.

3. DSSN Implementation for OntoWiki

We provide a prototypical implementation of our approach which utilizes the OntoWiki application framework [3]. OntoWiki is a Semantic data wiki as well as Linked Data publishing engine. The prototypical implementation is also the basis for the evaluation in Section 4. All features were realized by employing the extension mechanisms offered by OntoWiki. A main feature of OntoWiki is its storage layer independence. This means that an OntoWiki setup can use a high-performance RDF triple store (e.g. Openlink Virtuoso[27]) for knowledge bases up to the size of the DBpedia project [12] as well as a MySQL backend with a SPARQL2SQL query rewriter for small and mid-size knowledge bases. We tested and used the DSSN implementation with both backends.

The prototype described here, can be summarized as a WebID provider with an integrated communication hub. The following features are implemented so far:

- Users can create and manage their WebID profiles and any other Linked Data enabled resource (see Section 3.1).
- Users can make friends, subscribe to their activities and profile updates and receive changes instantly on change (see Section 3.2).
- Users can search and browse for friends and activities inside their social network as well as filter these resources by facets based on object and datatype properties (see Section 3.3).
- Users can comment on and subscribe to any DSSN resource which is equipped with a Semantic Pingback service or a PubSubHubbub-enabled activity stream (see Section 3.4).
- Users receive notifications if someone comments or links her WebID and send a pingback notification (see Section 3.5).

We describe the implementation of these features and provide insights into our rationale for choosing certain technologies. An commented screenshot of the central activity stream interface can be seen on Figure 3.

3.1. Creating and Updating Data Artefacts

Creating and managing Linked Data enabled RDF resources can be achieved without modifying the OntoWiki basic functionality. We employ the RDFau-

[27] http://virtuoso.openlinksw.com/

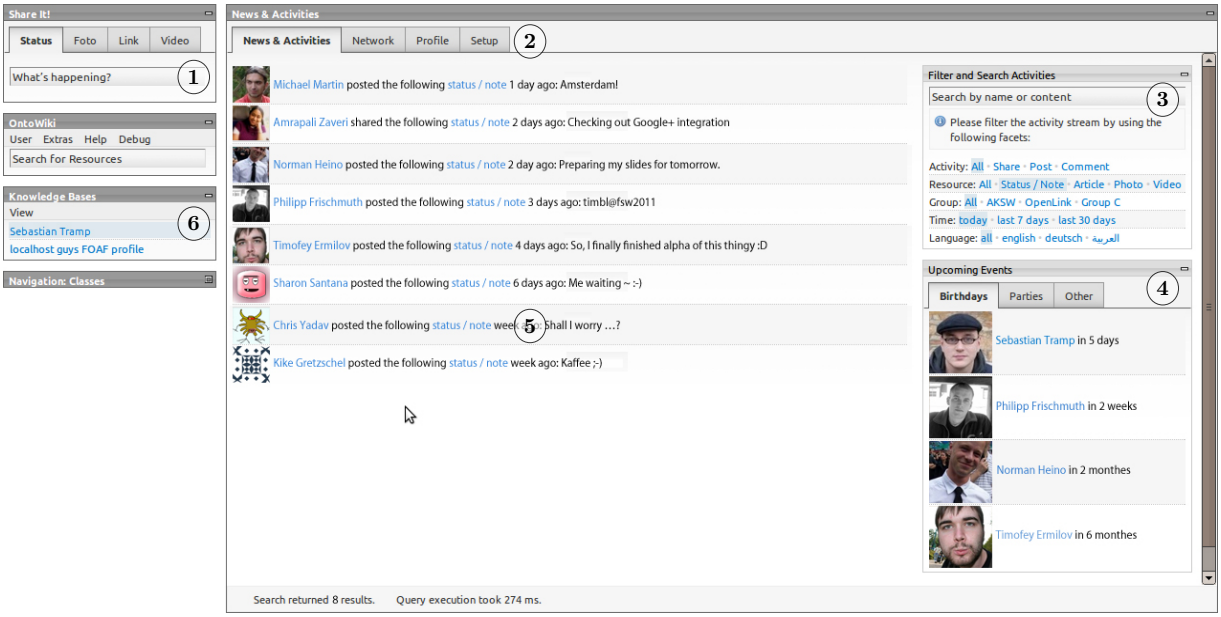

Fig. 3. Screenshot of the OntoWiki DSSN activity stream view. The interface elements are: (1) The *Share it!* activity creation module where users can post status notes and share links and media artefacts. (2) The main interface view tab to switch between configuration screen, profile manager, friending interface and the activity stream. (3) The activity filter and search module, to allow a facet-based browsing of activities. (4) The events module, which queries the cache social network data for birthdays and other events. (5) The activity stream, which is the result of the SPARQL query modified by the filter module. (6) The generic wiki interfaces to create any type of Linked Data resource (e.g. comments).

thor [24] JavaScript widget library in order to automatically create forms out of RDFa annotated HTML documents[28]. Without any user effort, a newly created resource will be Linked Data enabled if it shares the namespace of the OntoWiki installation. In addition to that, we added support for WebID authentication as well as for certificate creation by implementing a dedicated WebID extension. Since there is quite some cryptographic processes involved, this extension needs some prior configuration steps, e.g. the OntoWiki instance needs to be configured as a SSL/TLS enabled Web application.

3.2. Maintaining Social Network Connections

In order to maintain friend connections and other social network connections we execute the following steps in our implementation:

1. When a user enters a WebID inside the friending module, OntoWiki add a new statement (`foaf:knows`) into the RDF graph containing the users profile data. This automatically generates a Pingback request so the new friend will be notified.
2. The corresponding OntoWiki creates a new RDF graph, which will act as a cache for the WebID profile data about that particular friend. This new Knowledge Base is configured in such a way, that it is imported into the users graph automatically.
3. The wiki fetches the data from the friends WebID by employing the Linked Data principles. The newly generated graph caches that data in order to enhance the performance. Since those information is stored in a separate graph, a synchronization is trivial.
4. Finally, the wiki is subscribed to these feeds if they are published within the users profile: (1) The users activity feed, which the wiki uses to create the network activity timeline for the user. The incoming atom activity entries are transformed to AAIR resources and imported to an additional activity RDF graph. (2) The history feed for the friends' profile. This feed is employed in order to keep the cached WebID profile data up-to-date.

[28] Each output page, which is created by OntoWiki, is RDFa enhanced.

3.3. Ignoring Activities and WebIDs

Since a user may not be interested in all activities of her friends (e.g. gaming activities), we offer functionality to hide certain activities in the timelines that visualize the activity streams to which a user is subscribed. We therefore employ EvoPat [15], a pattern-based approach for the evolution and refactoring of RDF knowledge bases. EvoPat is integrated also as an OntoWiki extension and in conjunction with our DSSN implementation we apply this component to allow users to clean up their timelines.

Each time a user selects a *Hide all activities ...* button next to an activity, we create a new pattern, which subsequently matches such statements in the graph and removes them. Possible hide patterns are generated from the activity data itself, for instance *... from this user*, *... of this language* or *... about this object*. Once new data for a given user is added, we re-apply the pattern. In this way it also applies for future changes of the knowledge base.

3.4. Generating and Distributing Activities

A user is able to generate different kinds of activities in our implementation. In the current state we support three types of activities: status updates, photo sharing as well as link recommendations. We implemented a small extension, which displays a *Share It!* module inside the OntoWiki user interface. As a result the user can quickly access this functionality so that sharing is facilitated for the user. Once the user generates an activity, her activity feed is updated and subscribers to that feed are delivered with the new content by the push service.

3.5. Pingback Integration

All activities are represented as Linked Data resources that refer to a Pingback service and a corresponding activity feed. Thus, users can comment on any resource in their own social application and additionally subscribe to changes to that resource (e.g. comments by others). Each time someone comments on a resource (or otherwise links to it on the Linked Data Web), a Pingback request is sent to the owning OntoWiki instance. Consequently the publisher gets notified and is able to react again on that new activity, which facilitates conversations in a distributed manner. If a user is subscribed to a resource activity feed (which is automatically done, once she comments on a particular resource), she gets notified about other comments, even if she is not the commenting person or the owner of the resource.

4. Evaluation

We divided our evaluation process into two independent parts: Firstly, the qualitative evaluation part aims to prove the functionality of the DSSN architecture by assessing use cases from the Social Web acid test (Section 4.1). Secondly, the quantitative evaluation part aims to prove the real-world usefulness of our prototypical implementation by testing the performance and distribution of data in the social network (Section 4.2). This evaluation is carried out by using a social network simulation approach (Section 4.2.1).

4.1. Qualitative Evaluation: Social Web Acid Test

The Social Web Acid Test (SWAT) is an integration use case test conceived by the Federated Social Web Incubator Group of the W3C. Currently, only the first and very basic level of the test (SWAT0[29]) has been developed and described completely. Nevertheless, those parts of the next level (SWAT1) which are currently published are discussed here too.

SWAT0: The objectives of the first SWAT level have been specified in the following use case[30]:

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

Listing 5: Social Web Acid Test - Level 0

Utilizing all technologies described before, our DSSN architecture passes the SWAT0 without any

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

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

problems. The following enumeration describes the corresponding steps:

1. *User A takes a photo of user B and uploads it* (e.g. with the example application from Section 2.5): The Web space returns a link to the user's pingback server in the HTTP header of the uploaded image.
2. *User A explicitly tags the photo with user B:* This is done by creating a tag resource (`ctag:Tag`) which links both to the image and to the WebID of user B. A pingback client sends a ping request to all of these resources after publishing the tag on the Web.
3. *User B is notified that she is on a photo:* The notification is created by the pingback service of user B who has received a request from the tagging application which was used by user A.
4. *User C, who follows user A, receives the photo:* User C is instantly provided with an update in her activity stream, informing her about the new image.
5. *User C leaves a comment on the photo:* This is done in the same way as publishing the tag.
6. *User A and user B are notified about the comment:* User A will be notified because her pingback service informs her about this ping. User B will be notified only if she has subscribed to the activity feed of the photo provided that it exists. However, both notifications are optional and are not sent automatically.

SWAT1 is currently not finally defined[31], so an evaluation can be a rough sketch only. The next SWAT level will require a few different use cases which introduce some new Social Web concepts. However, most of the *user stories* are satisfied already as a consequence of the fully distributed nature of the DSSN architecture (e.g. data portability and social discovery). The more interesting user stories are: (1) The *Private content* and *Groups* use cases will require a distributed ACL management. Some ideas on using WebIDs for group ACL management are already published with dgFOAF [18] and we think that this is a good starting point for further research. (2) The *Social News* use case introduces a new vote activity. Since our architecture applies schema agnostic social network protocols, this new type of activity can be communicated as any other activity.

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

4.2. Quantitative Evaluation: DSSN Performance

Our next aim was to evaluate the architecture in quantitative terms. Based on the proposed architecture, a DSSN will be distributed over hundreds of servers. These social network nodes will have hardware specifications which can range from very light-weight (e.g. plug computers as proposed by the FreedomBox[32] project, smartphones and small virtual hosts) to medium and heavy class systems (e.g. cloud instances, hosted services and full root servers). Each of these nodes accesses only a small part of the complete social network graph since the information is shared only with the connected nodes.

Consequently, we need to pose the following questions in our quantitative evaluation:

1. How many incoming triples need to be cached from an averagely connected node in a week of average social network activity?
2. If a DSSN node queries these incoming triples with SPARQL, are the queries fast enough to provide the data for the user interface on such weak hardware?

To answer these research questions, we created an evaluation framework that allows for simulating the traffic within a DSSN. We apply this framework to measure the performance of our DSSN in a testbed using a large social network dataset.

4.2.1. Evaluation Framework Architecture

We decided to use a simulation approach in which activities are created artificially (but based on real data, see Section 4.2.2) on the social network rather than arranging a user evaluation, which strongly depends on the graphical user interface. In a first step, we added an additional service for remote procedure calls to the OntoWiki DSSN node as well as an execution client (replay agent) which can inject activities remotely controlled and based on input activity data in RDF. Then we used the public Twitter dataset described in [1] and transformed it to an RDF graph as a base for a replay of activities of type status note.

The workflow of the evaluation framework is depicted in Figure 4. It has a pipeline architecture which is built by using three additional tools that help us to

[32] `http://www.freedomboxfoundation.org/`

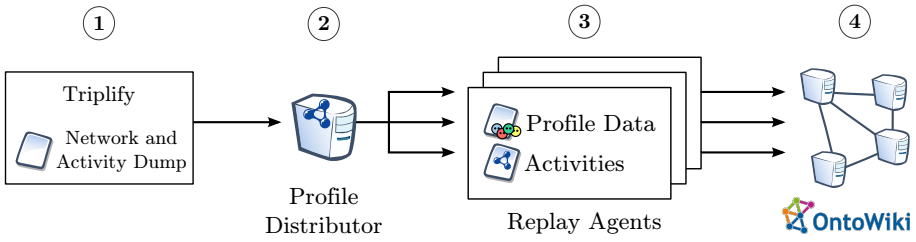

Fig. 4. Workflow of the social network evaluation framework: (1) An activity and profile graph is preprocessed and dumped as RDF. (2) The knowledge base is split up into a part for every single profile and can be distributed to a list of server machines. (3) For each profile, a *replay agent* is initialized which will create and update a specific profile as well as activities on a single social network node. (4) The social network node (in our case an OntoWiki instance) connects to other nodes as well as creates activities for the social network in the same way as it would be achieved under control of a human user.

generate and manage the generated replay agent test data. The generated data was processed by the profile distributor which splits the data into separate user profiles, each with personal data and activities. After splitting the data into single parts, the profile distributor passes each part on to a corresponding replay agent. Upon call the replay agent can instantiate a new OntoWiki-based DSSN node with the given user profile and activities.

4.2.2. Data Generation and Testbed Configuration

In a first step, we used Triplify [2] to generate the activity data from the relational database. The database consists of 2.3M public tweets fetched from 1701 Twitter accounts over a time frame of two months. For each posted tweet, we created a status note resource description and added two SIOC properties to link the creator account and publish the creation timestamp. In addition to that, we linked each status note to the Twitter terms of services to demonstrate the usage of licensing in our architecture (the data ownership issue from the Introduction)[33]. Listing 6 shows an example status note resource taken from the database[34].

For the creation of the corresponding activities we intepreted re-tweets as sharing and (original) tweets as posting activities and assigned different activity verbs based on the data. Each activity is linked to a FOAF person resource (without WebID specific enhancements) which we additionally enriched with a random `foaf:birthday`.

```
swj:o5516682621620224 a aair:Note;
  rdfs:seeAlso <http://twitter.com/#!/youngglobal/
       status/5516682621620224>;
  sioc:created_at "2010-11-19T08:04:14"^^xsd:
       dateTime;
  sioc:has_creator swj:youngglobal;
  dct:license <http://twitter.com/tos>;
  aair:content "I'm at Norwood (241 W 14th St, btw
       7th & 8th, New York) w/ 5 others. http://4sq
       .com/66OndN".

swj:a5516682621620224 a aair:Activity;
  atom:published "2010-11-19T08:04:14"^^xsd:
       dateTime;
  aair:activityActor swj:youngglobalPerson;
  aair:activityVerb aair:Post;
  aair:activityObject swj:o5516682621620224.
```

Listing 6: Example status note resource and corresponding activity.

```
swj:youngglobal a sioc:UserAccount;
  sioc:name "youngglobal";
  sioc:account_of swj:youngglobalPerson;
  rdfs:seeAlso <http://twitter.com/youngglobal>.

swj:youngglobalPerson a foaf:Person;
  foaf:name "youngglobal";
  foaf:knows swj:RdubuchePerson;
  foaf:birthday "01-31";
  foaf:depiction <http://a2.twimg.com/
       profile_images/1152004614/
       pip_2825_0370_normal.jpg>;
  foaf:account swj:youngglobal.
```

Listing 7: Example user account and FOAF person resource.

[33] Licensing statements can be easily added to any resource in the DSSN, e.g. by using the `dct:license` property.

[34] The Listings in this section use the prefixes `aair`, `dct`, `sioc`, `rdfs`, `xsd`, `atom` and `foaf` which are well known or already refered in this paper. The base prefix `swj` refers to the namespace `http://dssn.lod2.eu/SWJ2012/`, where the generated data set is available for download.

Listing 7 shows the user account and FOAF person resource which is linked from the activity in Listing 6.

In order to evaluate the query performance on different types of DSSN nodes in our architecture, we extracted four exemplary queries, which are crucial for rendering the user interface depicted in Figure 3.

Query Q1 asks for an ordered list of the last ten status posts of a given time frame. The query is exemplary for fetching a resource list based on a given user defined configuration. The query is used for area 2 on Figure 3 and is typically followed by a query which fetches the needed data of exactly these ten activities (rather than doing both in one single query).

```
1  SELECT DISTINCT ?r
2  WHERE {
3    ?r a aair:Activity.
4    ?r atom:published ?pub.
5    ?r aair:activityObject ?aairObject.
6    ?aairObject a aair:Note.
7    FILTER
8      (?pub >= "2010-12-01T00:00:00"^^xsd:dateTime)
9    FILTER
10     (?pub <= "2010-12-01T23:59:59"^^xsd:dateTime)
11 }
12 ORDER BY ?pub
13 LIMIT 10
```

Listing 8: Ordered list of the last ten status posts (Q1).

Query Q2 is used to build the facet-based exploration module depicted in area 3 on Figure 3. It asks for all values of a specific exploration facet of the activities of a given time frame. The query that is depicted in Listing 9 asks for the used verbs in the selected set of activities (possible verbs are post, share, comment etc.).

```
1  SELECT DISTINCT ?verb
2  WHERE {
3    ?r a aair:Activity.
4    ?r atom:published ?pub.
5    ?r aair:activityObject ?aairObject.
6    ?aairObject a aair:Note.
7    ?r aair:activityVerb ?verb.
8    FILTER
9      (?pub >= "2010-12-01T00:00:00"^^xsd:dateTime)
10   FILTER
11     (?pub <= "2010-12-01T23:59:59"^^xsd:dateTime)
12 }}
```

Listing 9: A list of verbs connected to a list of activities (Q2).

Query Q3 is used to fetch the list of the next five upcoming birthdays together with the associated person. Since `foaf:birthday` values are of datatype `xsd:string`, a string comparison has to be executed. Query 3 (see Listing 10) is used for area 4 on Figure 3.

```
1  SELECT DISTINCT ?person ?bday
2  WHERE {
3    ?person a foaf:Person.
4    ?person foaf:birthday ?bday.
5    FILTER (xsd:string(?bday) >= xsd:string("01-29"))
6  }
7  ORDER BY ASC(?bday)
8  LIMIT 5
```

Listing 10: List the next five upcoming birthdays (Q3).

Query Q4 is applied to prepare a human readable label for all the resources which are currently visible in the interface. This includes schema resource as well as instance data. The query uses a list of resources (line 5) and a list of possible label attributes (line 6) and fetches them in a vertical result set (e.g. with a minimal amount of projection variables). The client receives the data and has to select a value based on an ordered internal list (e.g. a `foaf:name` value is preferred over an `rdfs:label` value, because the latter is more general). This query strategy is especially useful in combination with incomplete data (e.g. use the `foaf:nick` if you do not have a `foaf:name`).

```
1  SELECT DISTINCT ?s ?p ?o
2  FROM <http://aksw.org/>
3  WHERE {
4  OPTIONAL {?s ?p ?o.}
5  FILTER(sameTerm(?s, <...>) || sameTerm(?s, <...>)
       || ... )
6  FILTER(sameTerm(?p, skos:prefLabel) || sameTerm(?p,
       dc:title) || sameTerm(?p, dct:title) ||
     sameTerm(?p, foaf:name) || sameTerm(?p, aair:
     name) || sameTerm(?p, sioc:name) || sameTerm(?
     p, rdfs:label) || sameTerm(?p, foaf:
     accountName) || sameTerm(?p, foaf:nick) ||
     sameTerm(?p, foaf:surname) || sameTerm(?p,
     skos:altLabel))}
```

Listing 11: Ask for all known title attributes for a given list of resources (Q4).

Since one of the ideas of a distributed social network is the usage of low-end hardware, which everyone can afford or which already exists in most house-

holds (e.g. DSL router or WLAN access points), we defined three prototypical categories of DSSN nodes where for which we would like to test the query performance:

A *server* is a typical host in a computing center which can be used for a rental fee per month. Privacy is moderatly preserved on such a system since the computing center staff can access the system. This category of DSSN node is used mostly by people with a strong technical background. In this category, we tested a Virtuoso 6.1.4 backed OntoWiki DSSN node on a dual core 2.4GHz system, with 4GB of RAM and an SSD.

A *FreedomBox* is a personal server running on a low-end system in an area where the users privacy can be preserved (e.g. as a DSL router in his household). No-one else has access to the system which runs 24 hours a day in the same way a server does. In this category, we used a virtual machine with 1GB of memory, one core and a 25% CPU limitation from the server system above. In addition to that, we limited the triple store process to 300MB of RAM.

Smartphones are very important for social network activities today, but they are used mostly as a thin client without a backend. We argue that connection stability and battery issues will be solved in the near future and smartphones can be used as first class DSSN nodes. In this category, we used an in-browser JavaScript API store based on rdfQuery[35] and deployed the data and the store without a frontend on an iPhone 4S.

4.2.3. Results and Discussion

As described in Section 4.2.2, the testbed consists of 1701 DSSN node profiles with 2.3M activities. We first looked, how this data was shared over these DSSN nodes to overview what amount of data these nodes have to store. Regarding this, two characteristic indices are important: (1) the number of activities of an account and (2) the number of related accounts which will receive these activities.

Figure 5 shows a scatter plot where each account corresponds to one point. The x axis represents the number of `foaf:knows` relations to other persons from the testbed network and the y axis depicts the amount of triples which are produced with the node's frontend (profile triples and activity triples).

Given this plot, we assume that the amount of friendship relations of an account and the amount of activities of an account do not correlate with each other. Since the given raw data included extreme values not suitable for our approach (see below), we cleaned the data by eliminating outliers in both dimensions. The average size of outgoing triples for all profiles after one week of activity is 1589 triples. The average amount of related contacts in the cleaned data is 225. We used these values as an artificial point in the graph and identified one single account which had the smallest distance to this point. This account was used for the query evaluation.

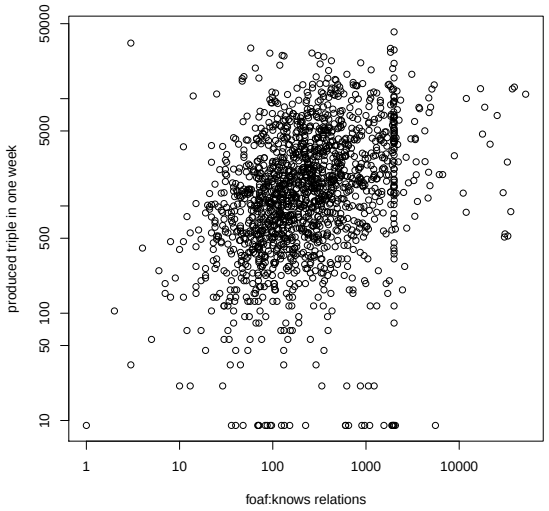

Fig. 5. Scatter plot with logarithmic axes of related accounts vs. the number of created outgoing triples after one week of social network activity (taken from [1], uncleaned).

The motivation behind this approach is to evaluate the average performance of a DSSN node. In [12] we analyzed the correlation between knowledge base size and query performance. This setup simplifies these results and assumes a linear correlation which is acceptable in the context of this evaluation.

The evaluated account has 212 `foaf:knows` relations with other accounts, shared 113 foreign notes and posted 5 original notes in one week. With this activities, the owner produced 1425 triples (incl. his profile). In addition to that, he received 396346 triples from his friends over Linked Data and PubSubHubbub.

We used this graph and executed the introduced queries on the three different systems, described in Section 4.2.2. Table 2 shows the average execution time in milliseconds after 5 runs for each query.

The results show, that querying this data is possible at least on real triple stores and with a moderate time frame. Using a DSSN node on a dedicated server

[35] `https://github.com/alohaeditor/rdfQuery`

Table 2
Runtime (in ms) of different evaluation queries

Query	Server	FreedomBox	Smartphone
Q1 (posts)	35	71	41325
Q2 (facets)	150	312	37558
Q3 (birthdays)	29	36	11324
Q4 (titles)	522	2205	n/a

should even work with much more data. The results from the smartphone demonstrate a gap of working triple store implementations for HTML5 applications.

Query 1 - 3 are similar in its structure since they combine at least two graph patterns with one or two filter. Query 2 is the slowest here since it uses graph pattern most which results in a query plan with many joins. Query 1 uses an `ORDER` clause on a big result set (all status notes timestamps). This may be the reason for being the slowest query on the smartphone.

Query 4 is hardest query in our experiment, since it does not use any graph pattern which can be used for filtering on index level. Instead it uses two complex filter which have to be evaluated on a lot of triple. Given these results, we will consider a different query strategy for querying the title attributes. One option is to send a query for each resource thus allow to restrict the index based on the subject.

The huge amount of data which is received by DSSN nodes in a realistic environment should be controlled not only by faster triple stores and more hardware, but also by implementing smart interfaces which cache at the right places and pre-calculate many interface elements fostering a faster user experience.

As a nice spin-off, this evaluation demonstrates how social network federation can be achieved if semantic interoperability is guaranteed. If a user wants to federate her DSSN node with other social networks, a semantic agent can easily fetch and write data from social network APIs, translate it to RDF and publish it as Linked Data including the discoverable services described in Section 2. The RDF translation can moreover be realized by the user's own DSSN node so that the user can be in full control of her data.

5. Related Work

Most Social Web applications operate as silos of information. This model poses some drawbacks such as the lack of interoperability between applications, having a one-single ownership model, the inability in fully exporting data as well as the opacity in using or transmitting private data. These challenges are the reasons for addressing a distributed model. Various architectures for achieving a federated social network have been proposed and numerous projects based on those architectures have been developed to provide convenient functionality to the users. With the advent of the Semantic Web, research in this field was adapted to taking advantage of machine-readable data and ontologies. We can roughly divide the related work into distributed social networks on the Web 2.0 and distributed social networks on the Semantic Web.

Distributed social networks on the Web 2.0. The distributed social network model emerged to overcome shortcomings attributed to centralized models. The foundation of the distributed model lies on a set of standards and technologies. This set of standards and protocols, which together are referred to as *Open Stack*, contains *Rss*, *PubSubHubbub*, *Webfinger*, *ActivityStreams*, *Salmon*, *OAuth authorization*, *OpenID authentication*, *Portable Contacts*, *Wave Federation Protocol*, *OpenSocial widget*, *XRD*, *OStatus* and *DSNP*[36]. For instance, the *Webfinger* protocol enables users to make email addresses valuable by adding metadata. Some of the projects which were developed using these technologies are: StatusNet[37], DiSo[38], GNU Social[39], The Mine Project[40], Appleseed[41], OneSocialWeb[42], BuddyPress[43], Cliqset[44], Posterous[45] and Diaspora[46]. These projects differ in the employed protocols and federation policy. The focus in projects such as *DiSo*, *The Mine Project*, *Appleseed* and *BuddyPress* is on equipping people with tools and functionalities. They allows users to build her own networks which enable them to manage and share data and relations. For instance, *DiSo* builds WordPress plugins by building up on OpenID, microformats and OAuth. It creates open and interoperable building blocks for launching decentralized social networks. In contrast, another strategy is to bridge between social networks to make a joint network. In a short overview, *StatusNet* is a mi-

[36] http://www.complang.org/dsnp/
[37] http://status.net/
[38] http://diso-project.org/
[39] http://foocorp.org/projects/social/
[40] http://themineproject.org/
[41] http://opensource.appleseedproject.org/
[42] http://onesocialweb.org/
[43] http://buddypress.org/
[44] http://cliqset.com/
[45] https://posterous.com/
[46] https://www.joindiaspora.com/

croblogging platform which extends OStatus for providing federated status updates; *GNU Social* is designed above *StatusNet* as a decentralized social network; *OneSocialWeb* aims at connecting social networks by employing XMPP [17] for instant messaging. *Diaspora* also uses XMPP, buddycloud channels and Activity Streams for federation.

Centralized models benefit from short development time, no required routing, central control and storage as well as data mining capabilities. In addition to that, using distributed models is solely subject to disadvantages e.g. reachability of interlinked data or the maintenance of many peers. While a federated model as a hybrid model improves some of those disadvantages (e.g. reachability maybe higher, since number of peers may be smaller), some of them still remain (e.g. full control over your owned data). There are many different views for tackling distributed social network challenges on the way to a federation in the Web 2.0. A well-known view is the *network of networks*, which employs existing protocols and standards for providing foundations on the basis of which networks can easily communicate with each other.

Another view is using mashups. Mashups are Web applications that combine data from more than one service provider to create new services. An example is the buddycloud project[47], running an inbox server as a centralized manager over federated social networks. This server aggregates all the posts and updates from social networks to which a user has subscribed. Here again, the XMPP protocol [17] is used for messaging and federation.

A user-centric architecture, used in Danube[48], employs individuals for maintaining personal data and relations. It allows them to manage their relationships with each other and with vendors. Another analogous view has been proposed by PrPl [19] as a person-centric, social networking infrastructure, where a person's data is logically collected in one place, and social networking applications can be executed in a distributed manner without a central service.

Distributed Social Networks on Semantic Web. Primarily, attempts have concentrated on the conformation of traditional technologies such as *microblogging*, *instant messaging* or *pingback* for Semantic Web. Thus, these adapted technologies can form the basis for a new generation of social networks. SMOB is a semantic and distributed microblogging framework introduced in [13]. It presents some main requirements for using microblogging at a large scale, i.e. machine-readable metadata, a decentralized architecture, open data as well as re-usability and interlinking of data. These challenges have been addressed in SMOB by using RDF(a)/OWL data, distributed Hubs for exchanging information and a sync protocol (based on SPARQL/Update over HTTP) and interlinking components. sparqlPuSH [14] utilizes the PubSubHubbub protocol to broadcast RDF query result updates. In this project, a SPARQL query which is associated with the agent is at first created and monitored in an RDF triple store. The registered user is notified whenever the result set changes.

Two important prerequisites for establishing a distributed social network on the Semantic Web is firstly to transform social network data into RDF and secondly, to aggregate the exported datasets by linking between person instances in different datasets. For the former prerequisite, an appropriate ontology is essential for representing social data. *FOAF* (Friend of a Friend) [7], which specifies how to describe personal information and relationships with other people in a social network, is well-suited for this purpose. The *SIOC* project [6], also extends FOAF in order to describe rich social data. The work presented in [5] uses *SIOC* regarding the representation of blog and bookmark content. For the second prerequisite, a graph matching model is needed for providing linkages. The work which has been carried out by [16] describes a method for interlinking user profiles from different social networks such as *Facebook*, *MySpace* and *Twitter*. The core idea is to generate RDF graphs, which can then be interlinked based on the corresponding user identifiers in each graph.

Beyond these activities, an infrastructure is necessary to which these technologies can be employed. Our current work is a pioneer effort in integrating current technologies into a coherent architecture to form a distributed social network in a semantic-based context.

6. Conclusions and Future Work

In this article we described our reference architecture and proof-of-concept implementation of a distributed social network based on semantic technologies. Compared with the currently prevalent centralized social networks, this approach has a number of advantages regarding privacy, data security, data own-

[47] http://buddycloud.com/
[48] http://projectdanube.org

ership, extensibility, reliability and freedom of communication. However, the work presented in this article can only be a first step of a larger research and development agenda aiming at realizing a truly distributed social network based on semantic technologies. Our implementation, for example, based on the OntoWiki technology platform is currently only a proof-of-concept implementation. As shown in the evaluation section, RDF triple stores, which are the foundation of any DSSN node, need to be enhanced, in particular from the mobile computing point of view. For a widespread use, the usability, scalability and the multi-client capabilities have to be improved. Likewise, the distributed realization of social networking applications (*apps*) as implemented in centralized social networks through APIs (e.g. Open Social) has to be investigated. We also expect that the combination of standards and protocols as described in this article will be implemented in a number of additional platforms (e.g. Wordpress and Drupal), thus making them first-class nodes of the DSSN.

Acknowledgments

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).

References

[1] Fabian Abel, Ilknur Celik, Geert-Jan Houben, and Patrick Siehndel. Leveraging the Semantics of Tweets for Adaptive Faceted Search on Twitter. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2011.

[2] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: light-weight linked data publication from relational databases. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 621–630. ACM, 2009.

[3] Sören Auer, Sebastian Dietzold, and Thomas Riechert. OntoWiki - A Tool for Social, Semantic Collaboration. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer, 2006.

[4] Tim Berners-Lee. Linked Data - Design Issues. website. last change: 2009/06/18; retrieved: 2011/07/25.

[5] Uldis Bojars, Alexandre Passant, Richard Cyganiak, and John Breslin. Weaving SIOC into the Web of Linked Data. In Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee, editors, *Proceedings of the Linked Data on the Web Workshop*, volume 369 of *CEUR Workshop Proceedings*, Beijing, China, April 2008. CEUR-WS.org.

[6] John G. Breslin, Stefan Decker, Andreas Harth, and Uldis Bojars. SIOC: an approach to connect web-based communities. *International Journal of Web Based Communities*, 2(2):133–142, 2006.

[7] Dan Brickley and Libby Miller. FOAF Vocabulary Specification. Namespace Document 2 Sept 2004, FOAF Project, 2004. http://xmlns.com/foaf/0.1/.

[8] Li Ding, Timothy W. Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 652–659. ACM, 2004.

[9] Maxwell Krohn, Alex Yip, Micah Brodsky, Robert Morris, and Michael Walfish. A World Wide Web Without Walls. In *6th ACM Workshop on Hot Topics in Networking (Hotnets)*, Atlanta, GA, USA, November 2007.

[10] Stuart Langridge and Ian Hickson. Pingback 1.0. Technical report, `http://hixie.ch/specs/pingback/pingback`, 2002.

[11] Michele Minno and Davide Palmisano. *Atom Activity Streams RDF mapping*. NoTube Project, 2010. `http://xmlns.notu.be/aair/`.

[12] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*, pages 454–469. Springer, 2011.

[13] Alexandre Passant, John G. Breslin, and Stefan Decker. Rethinking Microblogging: Open, Distributed, Semantics. In Boualem Benatallah, Fabio Casati, Gerti Kappel, and Gustavo Rossi, editors, *Web Engineering, 10th International Conference, ICWE 2010, Vienna, Austria, July 5-9, 2010. Proceedings*, volume 6189 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2010.

[14] Alexandre Passant and Pablo N. Mendes. sparqlPuSH: Proactive Notification of Data Updates in RDF Stores Using PubSubHubbub. In Gunnar Aastrand Grimnes, Sören Auer, and Gregory Todd Williams, editors, *Proceedings of the Sixth*

Workshop on Scripting and Development for the Semantic Web, Crete, Greece, May 31, 2010, volume 699 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[15] Christoph Rieß, Norman Heino, Sebastian Tramp, and Sören Auer. EvoPat - Pattern-Based Evolution and Refactoring of RDF Knowledge Bases. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes in Computer Science*, pages 647–662. Springer, 2010.

[16] Matthew Rowe. Interlinking Distributed Social Graphs. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Kingsley Idehen, editors, *Proceedings of the Linked Data on the Web Workshop (LDOW2009)*, volume 538 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[17] Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. Request for Comments 3920, IETF, October 2004. http://www.ietf.org/rfc/rfc3920.txt.

[18] Felix Schwagereit, Ansgar Scherp, and Steffen Staab. Representing distributed groups with $_{dg}$foaf. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II*, volume 6089 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2010.

[19] Seok-Won Seong, Jiwon Seo, Matthew Nasielski, Debangsu Sengupta, Sudheendra Hangal, Seng Keat Teh, Ruven Chu, Ben Dodson, and Monica S. Lam. PrPl: a decentralized social networking infrastructure. In Rick Han and Li Erran Li, editors, *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, pages 8:1–8:8, 2010.

[20] Manu Sporny, Toby Inkster, Henry Story, Bruno Harbulot, and Reto Bachmann-Gmür. WebID 1.0: Web identification and Discovery. Unofficial draft, August 2010. `http://payswarm.com/webid/`.

[21] Henry Story, Bruno Harbulot, Ian Jacobi, and Mike Jones. FOAF+TLS: RESTful Authentication for the Social Web. In Michael Hausenblas, Philipp Kärger, Daniel Olmedilla, Alexandre Passant, and Axel Polleres, editors, *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*, volume 447 of *CEUR Workshop Proceedings*, Heraklion, Greece, Jun 2009. CEUR-WS.org.

[22] Henry Story, Andrei Sambra, and Sebastian Tramp. Friending On The Social Web. In Evan Prodromou and Jan Schallaböck, editors, *Proceedings of Federated Social Web Europe 2011*, 2011.

[23] Sebastian Tramp, Philipp Frischmuth, Timofey Ermilov, and Sören Auer. Weaving a Social Data Web with Semantic Pingback. In Philipp Cimiano and Helena Sofia Pinto, editors, *Knowledge Engineering and Management by the Masses - 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11-15, 2010. Proceedings*, volume 6317 of *Lecture Notes in Computer Science*, pages 135–149. Springer, 2010.

[24] Sebastian Tramp, Norman Heino, Sören Auer, and Philipp Frischmuth. RDFauthor: Employing RDFa for Collaborative Knowledge Engineering. In Philipp Cimiano and Helena Sofia Pinto, editors, *Knowledge Engineering and Management by the Masses - 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11-15, 2010. Proceedings*, volume 6317 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2010.

[25] Giovanni Tummarello, Renaud Delbru, and Eyal Oren. Sindice.com: Weaving the Open Linked Data. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2007.