User-Defined Rules in a Distributed Address Book

Hannes Mühleisen and Adrian Paschke

Freie Universität Berlin
Department of Computer Science
Königin-Luise-Str. 24/26, 14195 Berlin, Germany
{muehleis,paschke}@inf.fu-berlin.de

Abstract. The Semantic Web as an evolution of the World Wide Web allows its users to share content over the boundaries of applications and web sites. However, current web-based systems for the controlled distribution of private information require the presence of information on centralized systems. A potential risk for the user is the central system operator which has full access to all stored information. Moreover, if a malicious user gets unauthorized access to the operator's management rights, the information stored by all users is exposed. In this paper, we demonstrate our alternative approach of a distributed address book, where users can store their information on a system under their own control. A special focus in our implementation was on the usability, in particular for the complex rule generation and processing chain, so that any user familiar with social network software is able to formulate access policies.

1 Introduction and Motivation

The Semantic Web as an evolution of the World Wide Web allows its users to share content over the boundaries of applications and web sites. To achieve this goal, the resources of the WWW are annotated using machine-readable meta data. The principles of Linked Data [2] describe a set of conventions how this meta data should be structured and published, so information can be re-combined and processed at an arbitrary location.

Conventional web-based systems for the controlled distribution of private information require the presence of information on centralized systems. In order to control the access to the managed information, these systems usually support secured data storage, access policies for the stored data and user authentication. Examples for such systems include various social networks: all users sign in on a central website to store their information there. Users configure their privacy settings on that website, for instance to reveal their telephone number only to a particular group of authorized users. However, a potential risk for the user is the central system operator which has full access to all stored user information. The user cannot necessarily trust the operator. Moreover, if a malicious user gets unauthorized access to the operator's management rights, the information stored by all users is exposed and each user's privacy is endangered.

We have developed a decentralized method for enforcing distributed user-defined access policies on Semantic Web data encoded in the Resource Description Format (RDF) [4] in a concept called the "Policy enabled Linked Data Server" PeLDS [6,5]. In

this paper, we demonstrate our alternative approach of a distributed address book, where users can store their information on a system under their own control. Integration with other users on other systems is easily possible, and the access to the stored information can be controlled by the users themselves.

The remainder of this paper is structured as follows: Section 2 describes the basic concepts for protecting information published as linked data with the definition and enforcement of access policies. Our back-end system, the Policy-enabled Linked Data Server is introduced shortly. Subsequently, Section 3 describes our Distributed Address Book application, its system architecture and user interaction concept. Finally, Section 4 concludes this paper.

2 Linked Data with Access Policies

Access policies are rules which describe the situations in which different pieces of information are made available to others. In order to express access policies for an RDF graph, the user has to describe those parts of the graph to be affected by a particular rule. This process can be compared to a "classification" of a document with sensitive contents. We have identified two levels for feasible data classification in Semantic Web data: the data model level and the semantic level. RDF Graphs can be decomposed into triples containing a subject, a predicate and an object element. These triples belong to the data model level. Resource descriptions, their affiliation with concepts and relationships to other resources are on the semantic level.

The Semantic Web Rule Language (SWRL) is a W3C member submission of the RuleML Datalog rule interchange language with additional OWL literals for Semantic Web ontology data [3]. SWRL rules can be evaluated by a reasoning program such as Pellet[1], KAON2 or RacerPro. SWRL rules can be represented using an RDF graph, and thus allow easy rule handling along with the RDF graphs containing the information to be protected. SWRL supports a number of predicates designed to enable access to the information encoded in RDF. Our PeLDS concept uses SWRL for the expression of access policies.

2.1 The Policy enabled Linked Data Server

The main feature for our concept of a Policy-enabled Linked Data Server is to provide a semantic storage system which allows its users to specify which elements of their RDF graphs are published to which user. This is achieved by calculating a temporary constructive view on the stored graphs that contains only those elements the querying user has been authorized to retrieve by the publishing user. To facilitate the easy description of access policies, we have developed a rule-based access policy format based on SWRL [6, Sec. 4.1].

Users publishing data on the system can define an access policy for each dataset they have created. The system guarantees the enforcement of a valid policy during each operation involving this dataset. Each rule consists of a label, a rule antecedent describing the condition under which the rule is satisfied and a consequent. Both the antecedent

[1] `http://clarkparsia.com/pellet`

and the consequent contain a collection of logical predicates joined by the logical *AND* condition. In addition to all of SWRL's predicates, our policy language supports custom predicates enabling data classification for single triples and resources as well as instances of specific classes.

Listing 1.1 gives an example of an access policy containing a single rule. The rule expresses the following notion: the user "Bob" is permitted to access the phone number of the user "Alice"'. The rule is labeled *phoneRule* and contains two antecedent predicates. The first predicate specifies the *?action* resource to be an instance of the concept *QueryAction*, the second predicate requires the *actor* resource to have `http://example.com/bob` as the value for its *actor* property. The consequent consists of a data classification predicate covering all triples with resource `http://example.com/alice`, property `ex:phone`, and arbitrary values.

```
phoneRule:
QueryAction(?action) && actor(?action, http://example.com/bob)
 => permit_triple(http://example.com/alice, ex:phone, *);
```

Listing 1.1. Example access policy

3 The Distributed Address Book

Our demonstration application is a distributed address book designed as a web-based application which is run from a standard web browser. As one would expect from an address book application, users can add, view and remove contacts, and also organize their contacts into groups. Users are able to change their own address book entry. In contrast to most previous systems, users are able to securely integrate their contact information with other users using different installations of the application without *any* form of previous setup. A special focus in our implementation was on the usability, in particular for the complex rule generation and processing chain, so that any user familiar with social network software is able to formulate access policies. The address book is using the PeLDS system only as a storage back-end, and all data concerning other users is retrieved on-the-fly using Linked Data concepts. Users define access policies using a set of pre-defined building blocks to secure their data.

3.1 System Architecture

Fig. 1 shows the two main parts of the Distributed Address Book: *Addressbook Client* is a JavaScript application running inside the user's web browser. The common combination of HTML and CSS is used to present information to the user. The client is sending HTTP requests to the corresponding *Addressbook Server*. This server is a PHP application which uses the ARC RDF library[2] for dereferencing and the access to the PeLDS

[2] http://arc.semsol.org

back-end storage. The PHP interpreter is run in an instance of the Apache HTTP server. All persistent user data as well as access policies are stored inside a PeLDS instance. The access to this data is performed using the RDF query language SPARQL [7] for query operations and SPARQL/Update [8] for update operations.

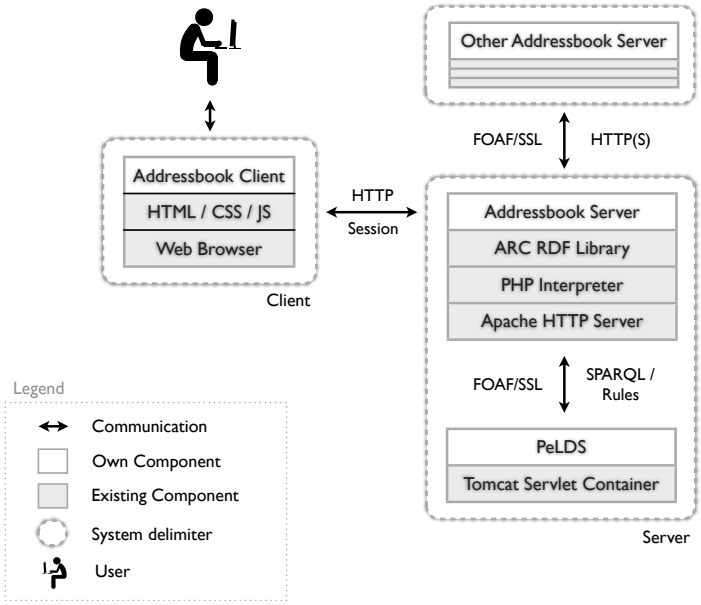

Fig. 1. Distributed Address Book - Components

Only authorized users are able to view an entry according to the defined authentication and authorization mechanisms in every communication layer. Between *Addressbook Client* and *Addressbook Server* HTTP sessions are maintained which are established after a successful user/password authentication by the user. *Addressbook Server* and local or remote *PeLDS* instances use the FOAF+SSL authentication scheme which employs SSL client certificates and a fully distributed certificate validation mechanism [9].

3.2 Contact Information and Access Policy Input

User profiles consist of various fields, and the application is generating the input form using a configuration file. This configuration defines the following properties for each field: (1) the form elements the user inputs his data in, (2) a method to check whether these inputs are valid, (3) a method to convert the data given by the user into RDF, (4) a method to load the form data from RDF, and (5) a method to generate a rule consequence fitting to the RDF snippet of this field. This way, the entire process of loading the current values into the generated profile form, validating all values, and creating

the RDF graph corresponding to the submitted values is very flexible and customizable. The application itself on the other hand defines various visibility levels, which then form the rule antecedents. At the moment, five visibility levels are implemented and selectable for each field:

1. *Public* - Field entries are made available to all users and clients, even without authentication.
2. *All contacts* - Field entries are only visible to users, who have been added as a contact by the current user.
3. *Group* (requires group selection) - Field entries are only visible to users which have been added to the selected contacts group by the current user.
4. *Contact* (requires contact selection) - Field entries are only visible to the selected user.
5. *hidden* - Field entries are only visible to the user that has created them.

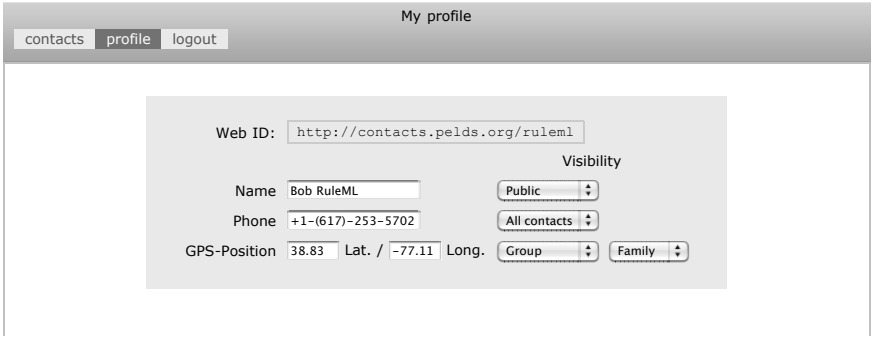

Fig. 2. Bob's Account Settings

Fig. 2 shows an example for the profile view for the user "Bob". Three fields are defined, "Name", "Phone", and "GPS-Position". Bob has chosen for his name to be visible to everybody, his phone number to be visible to all his contacts, and his GPS Position only to be visible to the members of the contacts group "Family" he has defined. From this input, the address book generates an RDF graph shown in Listing 1.2 encoded in the N3 RDF format [1] and an access policy consisting of three rules shown in Listing 1.3 in the PsSF language that can be evaluated by the PeLDS system.

```
<http://contacts.pelds.org/ruleml> foaf:name "Bob RuleML" .
<http://contacts.pelds.org/ruleml> foaf:phone
        "+1-(617)-253-5702" .
<http://contacts.pelds.org/ruleml> foaf:based_near _:pos .
_:pos rdf:type geo:Point .
_:pos geo:lat "38.83" .
_:pos geo:long "-77.11" .
```

```
<http://contacts.pelds.org/ruleml> foaf:knows
        <http://contacts.pelds.org/alice2> .
```

Listing 1.2. RDF graph created for Bob

The first rule in the access policy, *AddressbookDocument_[...]_name*, defines Bob's phone number to be publicly available. The rule condition sets no constraint on the client, it just specifies the request (*?action*) to be a query operation. The consequence consists of a data classification predicate *permit_triple*, this way the triple `(http://[...]/ruleml,foaf:name,"Bob RuleML")` containing Bob's full name is marked to be affected by this rule using the wild card "*" and is thus made available to all other users.

The second rule, *AddressbookDocument_[...]_phone* makes Bob's phone available to contacts. The rule conditions therefore contain not only the restriction for the current request to be a query in the first predicate (as before), but also puts constraints on the querying user (*actor*), who has to be the entity responsible for the current action according to the second predicate. The third condition predicate then requires a triple defining the querying user to be known by Bob. The consequence is defined analog to the first rule.

Finally, the third rule *AddressbookDocument_[...]_pos* describes the access conditions for Bob's GPS position, which only members of the "Family" contact group are able to see. As in the second rule, the first three predicates of the rule condition only match if the querying user is in the contact list of Bob, the fourth predicate then checks for the membership of the contact in the "Family" group. The consequence contains a more complex data classification: Since the position is a complex object consisting of latitude and longitude, it is more efficient to perform a instance-classification on the class *geo:Point*. The reference to this instance is classified using a simple triple pattern with the *foaf:based_near* predicate.

```
AddressbookDocument_RdfInputText_RdfVisibilityWorld_name:
pelds:QueryAction(?action) =>
permit_triple(http://contacts.pelds.org/ruleml,foaf:name,*);

AddressbookDocument_RdfInputText_RdfVisibilityAllContacts_phone:
pelds:QueryAction(?action) &&
pelds:actor(?action,?actor) &&
foaf:knows(http://contacts.pelds.org/ruleml,?actor) =>
permit_triple(http://contacts.pelds.org/ruleml,foaf:phone,*);

AddressbookDocument_RdfInputPos_RdfVisibilityGroup_pos:
pelds:QueryAction(?action) &&
pelds:actor(?action,?actor) &&
foaf:knows(http://contacts.pelds.org/ruleml,?actor) &&
foaf:member(http://contacts.pelds.org/ruleml/group/family,
        ?actor) =>
permit_triple(http://contacts.pelds.org/ruleml,
        foaf:based_near,*) &&
permit_instance(geo:Point);
```

Listing 1.3. Access Policy for Bob

3.3 Access to Contact Information

Users can add contacts to their address book using their URL as an identifier and link to the remote contact data. Contrary to current systems, our Distributed Address Book allows the addition of users on different Addressbook servers. Data retrieval is then performed on the fly using Linked Data dereferencing. The information retrieved is controlled by the respective PeLDS instances, and the Addressbook Client component authenticates itself against every remote node it retrieves data from using the FOAF+SSL verification method.

As shown in Fig. 3, the Addressbook client displays a contact and a group list to the user, which can then choose a contact to display his or her details. Continuing our example from the previous section, Alice has successfully logged into the Distributed Address Book and added Bob as a contact using his URL. Also, she has created a group "Family", and added Bob to it. Now, she selected the group "Family" in the left column to list all contacts, and then chose the contact "Bob RuleML" in the center colum to display Bob's contact details in the right column. Bob's name, phone number, and current positions are displayed. Alice is able to see all this, because Bob's access policy permits her to see this information.

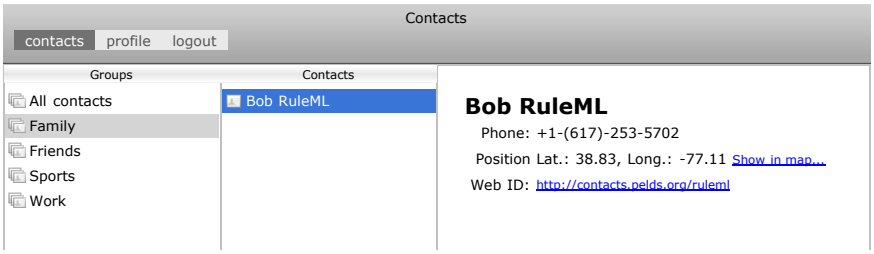

Fig. 3. Alice's Contacts View

4 Conclusion

We have motivated the need for an architectural change in web-based computer systems storing private information. We have introduced our concept of protecting information published as Linked Data using access policies consisting of declarative access rules. Our concept of the Policy enabled Linked Data Server was described shortly. The main topic of this paper was the architecture and user interaction of our prototype of a Distributed Address Book. The Distributed Address Book enables its users to store contact information on a system they trust, and also makes secure integration with users of other systems possible. Users are able to create access policies themselves by following the template structure offered to them in the application. From the user's choices, an individual access policy is compiled, stored, and enforced. The described system is

available on-line at http://contacts.pelds.org, its source code can be downloaded at http://www.pelds.org.

Acknowledgments We would like to acknowledge the contributions of Martin Kost and Johann-Christoph Freytag. This work has been partially supported by the "Inno-Profile Corporate Semantic Web" project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Laender - Entrepreneurial Regions.

References

1. Berners-Lee, T.: Notation 3 (1998), http://www.w3.org/DesignIssues/Notation3.html, http://www.w3.org/DesignIssues/Notation3.html accessed on 2010-04-20
2. Berners-Lee, T.: Linked data (2006), http://www.w3.org/DesignIssues/LinkedData.html, http://www.w3.org/DesignIssues/LinkedData.html accessed on 2010-04-20
3. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language (2004), http://www.w3.org/Submission/SWRL/, http://www.w3.org/Submission/SWRL accessed on 2010-04-20
4. Manola, F., Miller, E., McBride, B.: RDF primer (2004), http://www.w3.org/TR/rdf-primer/
5. Mühleisen, H.: Zugriffsrichtlinien und Authentifizierung für Linked-Data-Systeme. Master's thesis, Humboldt-Universität zu Berlin, http://muehleisen.org/da-hm-ld.pdf (2009), http://muehleisen.org/da-hm-ld.pdf
6. Mühleisen, H., Kost, M., Freytag, J.C.: SWRL-based Access Policies for Linked Data. In: Kärger, P., Olmedilla, D., Passant, A., Polleres, A. (eds.) Proceedings of the Second Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2010). vol. 576 (2010), http://CEUR-WS.org/Vol-576/paper1.pdf
7. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF (2008), http://www.w3.org/TR/rdf-sparql-query/
8. Seaborne, A., Manjunath, G., Bizer, C., Breslin, J., Das, S., Davis, I., Harris, S., Idehen, K., Corby, O., Kjernsmo, K., Nowack, B.: SPARQL Update (2008), http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/
9. Story, H., Harbulot, B., Jacobi, I., Jones, M.: FOAF+SSL: RESTful authentication for the social web (2009), submitted to Semantic Web Conference 2009