

# SWRL-based Access Policies for Linked Data

Hannes Mühleisen, Martin Kost and Johann-Christoph Freytag

Humboldt-Universität zu Berlin  
Department of Computer Science  
hannes@muehleisen.org, {kost | freytag}@informatik.hu-berlin.de

**Abstract.** Social applications are one of the fastest growing areas in the Web. However, privacy issues ensue if all information of all users of these applications is stored on a single computer system. With small extensions to Semantic Web technologies and Linked Data concepts, a distributed approach to the social web is possible, where users retain fine-grained control over their data and are still able to combine their data with users on different systems. We describe our concept of a Policy-enabled Linked Data Server (PeLDS) obeying user-defined access policies for the stored information. PeLDS also supports configuration-free distributed authentication. Access policies are expressed in a newly developed compact notation for the Semantic Web Rule Language. Authentication is performed using SSL certificates and the FOAF+SSL verification approach. We evaluate our concept using a prototype implementation and a distributed address book application.

## 1 Introduction

The Semantic Web as a new generation 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 far, no access control mechanism supporting fine-grained access policies is available for Linked Data, although a number of fitting scenarios are conceivable.

Previous web-based systems for the controlled distribution of sensitive 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 set their telephone number only to be visible to a particular group of users. However, system operators always have access to all of the stored information. This is an unsatisfactory situation, as the operator's behavior cannot be foreseen. Moreover, if a malicious user manages to circumvent the access restrictions, the information stored by all users is exposed and each user's privacy is endangered.

We will outline an alternative approach, where users store their information on a system under their control. Integration with users on other systems is supported, and the

access to the stored information can be controlled by the users themselves. To implement our approach, we use and combine Semantic Web technologies. We also slightly extended some technologies to enable users publishing Linked Data content to specify who is allowed to retrieve their information. This way, distributed social web applications with support for sophisticated access policies can be developed.

The rest of this paper is structured as follows: in Section 2 we describe the related work in the field of access policies for Semantic Web data, Section 3 describes the requirements to formulate access policies for RDF data. Section 4 describes our concept design of the Policy-enabled Linked Data Server (PeLDS). Section 5 shows our results in evaluating a PeLDS prototype and our demonstration application, called the “Distributed Address Book”. Finally, Section 6 concludes this paper.

## 2 Related Work

Tootoonchian et al. describe a privacy management system named “Lockr” especially designed for social networks [18], thus their policy format is limited to describe required social relationships for data access. However, by using a general-purpose format such as W3C’s Resource Description Format (RDF) as data format, this domain-specific problem can be handled by a more general approach. Hollenbach, Presbrey and Berners-Lee present an approach where RDF metadata is used to describe general-purpose access policies to RDF files stored on a web server [9]. While their approach mentions the possibility of extending access control to the data model level, they developed and evaluated access policies only for atomic RDF files. Research in the area of access policy languages for RDF data is exhaustively described in an article by Duma, Herzog and Shahmehri [6]. They especially conclude on the need for fine-grained access policy languages for RDF graphs and their elements. Reddivari, Finin and Joshi developed such a language in [16] as well as an implementation of a system evaluating these access policies based on the Jena inferencing engine. Jain and Farkas follow a similar approach [11]. They also show why access policies developed for XML data representations are not applicable to RDF data. Once access policies are defined, their enforcement presents another challenging task. Abel et al. developed such a mechanism [1] using access policies and query expansion. Neither of the mentioned solutions were available as an implementation ready for usage or evaluation at the time of this writing. Additionally, Web Ontology Language (OWL) reasoning and the handling of the inferred information is not supported by these approaches. OWL and its evaluation make powerful access policies possible, as we will show later.

The language Rei [12] was considered most suitable for the task of enabling users to express their access policies for semantic web data, but focuses more on developing an ontology for policy expression than to their actual application. We therefore chose to work on a more abstract level, that is, create a policy language and evaluation methods suitable for any RDF-based access policy expression language such as Rei. At the same time, we stress compatibility to the Linked Data principles, and are thus unable to build upon concepts that require a special form of protocol or trust negotiation, for example Protune [5].

The challenge of authenticating users in the Semantic Web environment is not as straightforward as in the conventional WWW, as browsing activities can require requests to multiple systems. Story et al. [17] have presented the FOAF+SSL concept for distributed authentication designed to make a distributed social web feasible.

### 3 Access Policies

The enforcement of explicit access policies can be used to limit access to data. In simple cases, these access policies consist of lists of users with access privileges. This eliminates the need for complex access policy evaluation, but only enables very crude access control. A more thorough approach allows users to specify custom access policies on various levels of expressiveness. These access policies are then used to determine which data the current user is allowed to retrieve or manipulate. Most users have an intuitive notion which information should be made publicly available, and which information should only be released to a limited group of people. Expressing intuitive access policies in a formal way rises the challenge of bridging the gap between human intuition and machine-readable definition. We begin our discussion of access policies with the definition of the term “access policy”:

**Definition 1.** *An Access Policy is a set of rules. These rules are evaluated in order to decide whether a user is allowed to access a data object [13].*

#### 3.1 Types of Access Policies

Whereas one could always write a custom program to decide which information should be communicated, a declarative expression of access policies is commonly preferred to keep policy expression and policy evaluation apart. In general, we distinguish three different types of access policy patterns:

*Discretionary Access Control (DAC)* distinguishes between named users and named objects such as files. A mechanism allows users with access rights to a specific object to award their access rights to other users or groups of users. There is a way to limit the propagation of access rights to sub-objects. The granularity of access rights can be refined to the level of individual users and objects [13]. A well-known example for DAC is the UNIX file systems file permissions model.

*Mandatory Access Control (MAC)* requires all objects and users to be described by a global access policy. Every object and every user is annotated with a security classification level. These classification levels are organized in a hierarchical way and provide the basis to decide whether an object can be accessed. Users may only access an object, if they possess a security classification equal or higher in the security hierarchy. Users cannot award their access rights to other users [13]. Systems supporting MAC are frequently used by public authorities, for example, to control access to confidential (“classified”) information.

*Role-based Access Control (RbAC)* does not distinguish between single users regarding access rights. Users are simply given a “role” according to their assignments, all access rights are bound to that role. A user can possess multiple roles. RbAC is a simplified form of MAC lacking security classification hierarchies, but derivation and composition of roles are still possible [7].

### 3.2 Data Classification

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 is referred to as “classification” here. Data classifications can be defined on multiple abstraction levels of an RDF graph. We distinguish three different levels with increasing abstraction: syntactical level, data model level and semantic level. An RDF graph in its serialized form can be assigned to the syntactical level. 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.

Data classification for RDF graphs on the *syntactical level* has been shown to be ineffective [11], mainly because serialization formats permit multiple ways of representing an identical graph. This classification level is therefore not pursued further.

Reasonable data classification for RDF graphs can be performed on the *data model level*. Triples are the smallest units to be classified. They are logically independent of any syntactical representation and can be classified easily through the usage of triple patterns. Triple patterns describe matching conditions for each triple element and can be used to select graph elements. On each request a system is able to classify (and hence control access to) every triple by evaluating all triple patterns currently present, an approach also followed in [11]. Wildcards can be used to classify a set of triples or triples with unknown values. An example for this classification is contained in Listing 1.1 within the following section.

Finally, the *semantic level* allows classification of data based on concepts defined with schema languages such as RDF Schema or OWL. This classification allows for a set of related triples to be classified by a single pattern. As the classification references the concept definition, updates to the concepts are automatically considered for classification. Resources can be classified indirectly by assigning them to a classified concept with OWL statements. One approach allowing data classification on the basis of RDFS concepts is described in [16]. However, OWL support is desirable due to its more powerful expressions for concept and property relationships, for instance transitive properties.

### 3.3 Semantic Web Rule Language

The Semantic Web Rule Language (SWRL) is a generic rule language for Semantic Web data. SWRL rules can be evaluated by a reasoning program such as Pellet [4], 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 rules describe implications and consist of two lists of predicates, the antecedent and the consequent part. If all predicates of the antecedent take the Boolean value *true*, all predicates in the consequent part are evaluated. The usable predicates are given by SWRL’s language specification. The predicates are listed below with their conditions under which they will be evaluated to *true*:

- $C(x)$  - A resource  $x$  is an instance of the concept  $C$ .

- $D(z)$  - The value  $z$  is of data type  $D$ .
  - $P(x, y)$  - The resource  $x$  has a property  $P$  with a reference to the object  $y$ .
  - $Q(x, z)$  - The resource  $x$  has a property  $Q$  with the literal value  $z$ .
  - $sameAs(x, y)$  -  $x$  and  $y$  identify identical resources.
  - $differentFrom(x, y)$  -  $x$  and  $y$  identify distinct resources.
  - $builtin(r, z_1, \dots, z_n)$  - The built-in function  $r$  with the parameters  $z_*$  returns *true*.
- A number of functions providing standard comparisons are defined by the language specification. Additional functions can be added by the user if required.

The RDF or XML representation of SWRL rules is not designed to be human-readable, thus it is usually displayed in a Prolog-like notation [10]. However, this notation is not intended to be interpreted by a computer, thus SWRL rules are usually written using specialized programs. Our PeLDS concept uses SWRL for the expression of access policies, see Section 4.

## 4 PeLDS System Concept

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 creating a temporary view on the stored graphs that contain only those elements the querying user has been authorized to retrieve by the publishing user. The access policy is expressed in a custom policy language. This language can be used to implement all types of access policies described in Section 3. The concept can be compared to views on relations in relational databases.

The entire data stored is partitioned into *datasets* using named graphs to support multiple users. To achieve this, every triple stored is assigned to a graph identifier. This way, all triples belonging to a specific graph can be retrieved from the storage component. This mechanism is used here to achieve multi-user capabilities: storage operations require a graph identifier to be specified, and access policies as well as ownership information are bound to each single named graph.

As access policies contain rules, we have decided to use a general-purpose rule language to express our access policies. We start by introducing our descriptive access policy language PsSF based on SWRL, then we describe the algorithms for policy evaluation, and we finish with a description of the various operations provided by PeLDS.

### 4.1 Policy Language PsSF

Access policies are described as a set of rules defining access conditions for each dataset stored on the server. 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. To facilitate the easy description of access policies, we have developed a short notation for policies and rules we call “Prolog-style SWRL Format (PsSF)”. Each rule consists of a label, a rule antecedent describing the condition under which the rule is satisfied and a consequent. Both the antecedent

and the consequent contain a collection of predicates joined by the logical *AND* condition. In addition to all of SWRL's predicates described in Section 3.3, PsSF supports three predicates enabling data classification on the identified levels. These classification predicates can only be used in rule consequents.

- *permit\_triple(subject,predicate,object)* - Access to all triples matching the parameters *subject*, *predicate* and *object* is permitted. All parameters may be replaced with the wildcard character *\**.
- *permit\_resource(resourceUri)* - Access to the resource with the identifier *resourceUri* is permitted. The wildcard *\** can be used to enable access to arbitrary resources (e.g. the complete dataset). This predicate is merely a special case of the first one.
- *permit\_instance(conceptUri)* - Access to all instances of the concept identified by *conceptUri* as well as all instances of derived concepts is permitted.

Each access rule can be defined according to different types of access. Currently, we only distinguish query and update actions. Conditions have to be expressed in a positive fashion, negation is currently not supported for decidability reasons [14]. The rule syntax is described in detail with examples in [15]. Using this syntax, users can specify their access policies.

Listing 1.1 gives an example of a PsSF rule. The rule expresses the following notion: the user Horst is permitted to access Anna's phone number. 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/horst` as the value for its *actor* property. The consequent consists of a data classification predicate covering all triples with resource `http://example.com/anna`, property `ex:phone`, and arbitrary values.

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

**Listing 1.1.** Example PsSF rule

## 4.2 Policy Schema and Evaluation

We have developed a simple OWL schema to describe the actions performed on the stored datasets and make query meta data accessible for PsSF rules. Three main concepts are defined: *Action* for query-related meta data, *Rule* to model single rules as a part of access policies, and *TriplePattern* for defined data classifications. The concepts *UpdateAction* and *QueryAction* are derived concepts to model the different interaction types. Each action holds a user identifier and a one-to-many relationship to the rules defined by the access policy. Each rule contains a reference to its data classifications within the *TriplePattern* instances.

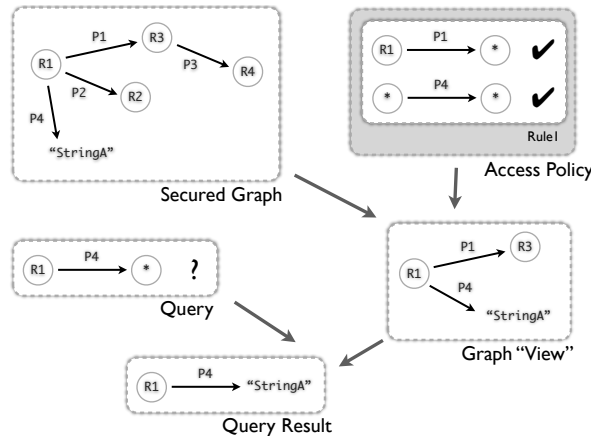


Fig. 1. Query data flow for PeLDS

An RDF graph containing the appropriate instances of the described schema is created for every request and merged with the affected dataset according to the defined access policy. Each rule from the access policy is attributed with an additional consequence to add the rule identifier to a global list of matched rules. If such a rule matches due to sufficient access rights for the current user, it will be added to this list. A reasoner performs rule evaluation by reasoning on the defined OWL and SWRL rules. PeLDS is then able to determine the data classifications defining the graph elements the current user is authorized to retrieve.

Based on an empty result graph, the requested dataset is loaded into memory and the access policy is translated into instances of the policy schema and is added to the dataset. The list of rules is evaluated for the information present in the dataset together with the user identity given in the Action instance. If a rule matches, every triple matching the data classifications contained in the rules consequence predicate list is copied from the dataset to the result graph. The user’s query is now executed on the result graph, and the query results are sent back to the user. This process is depicted in Figure 1: a secured graph containing various resources is queried. The specified access policy allows access to all triples with resource *R1* and property *P1* and all triples with the property *P4*. Hence, the triples (*R1*,*P1*,*R3*) and (*R1*,*P4*,"StringA") form the temporary graph view authorized for the current user. The user’s query for the value of the property *P4* on the resource *R1* can then be answered with the corresponding element of the graph view.

### 4.3 Encrypted Communication and Authentication

The Linked Data principles include the principle of dereferencing: resolving a URL found as an identifier within an RDF graph yields another RDF graph describing the resource identified by this URL. To implement arbitrary dereferencing, authentication cannot be based on shared secrets, as any URL may appear within an RDF graph. As

a consequence, user name/password or local trust settings for certificates are neither an elegant nor a scalable solution.

We use the HTTP Secure Protocol (HTTPS) for communication as well as certificate exchange and the approach presented by Story et al. [17] to validate SSL client certificates: The URL describing the user making a request is included within the SSL client certificate used to sign the HTTP request to an HTTPS server. Dereferencing this URL yields an RDF graph containing RDF triples defined by the Friend-of-a-Friend (FOAF) vocabulary. This graph also contains meta information about the cryptographic key used to sign the request, which is only available to the owner of the specific key. For RSA keys, this is the modulus and exponent of the private key. The server receiving the request is now able to verify whether this request was issued by the person controlling the URL included in the certificate, which is sufficient to identify the requesting user. This authentication mechanism does not rely on a global trust system or local settings. The concept can be used to authenticate Linked Data clients in a safe manner. Even though we have chosen FOAF+SSL due to its use of Linked Data, other authentication solutions such as OpenID could also be used and integrated.

#### 4.4 Interface and Operations

To maximize compatibility to existing software components, the PeLDS API was designed to be as consistent as possible regarding existing standards for handling RDF graphs. Additional API operations were added to enable policy management. In total, four main operations were identified: policy update, data update, data query, and dereferencing. Users are assumed to be authenticated and identified through their URLs. Datasets are generally created if the update operation is given an unknown graph identifier, they are then annotated with the URL identifying the user issuing the corresponding operation and thus “owned” by this user.

**Policy Update** - *reponseCode = updatePolicy(datasetUri, policy)*

Only the dataset owner may specify the access policy. The dataset URI has to be specified along with an access policy detailing which parts of the dataset should be disclosed. The new access policy is stored if it is syntactically correct according to the PsSF language specification [15], and the corresponding response code is returned to the user. Any existing access policy is overwritten.

**Data Update** - *reponseCode = updateData(datasetUri, update)*

RDF graphs can be uploaded, changed and deleted. This is facilitated using an update statement describing which graph elements to change. This is preferable to sending the entire graph for each update operation, as less graph elements have to be communicated. If the dataset owner issues an update, it is approved without further action. If another user tries to update the dataset, all changed graph elements have to be approved by the corresponding access policy. An update is only stored to persistent storage if no error has occurred.

**Data Query** - *result = queryData(datasetUri, query)*

An RDF graph (or parts of it) stored on a PeLDS instance are retrieved using a query language. The user has to specify a dataset identifier and a query. The query is executed on the elements the authenticated user is authorized to see by the access policy present for the requested dataset. This process was described in Section 4.2. The query

is expressed in a query language suitable for querying RDF graphs. The result value contains authorized graph elements from the specified dataset matching the query. If the specified dataset does not exist, no error is returned, because the awareness of the existence of a dataset can already be sensitive information. If no access policy is set for the dataset, only the dataset owner may issue queries to it. To achieve compatibility with existing software packages, authentication is optional for this operation.

**Dereferencing** -  $result = dereference(resourceUrl)$

In order to fulfill the Linked Data requirements, PeLDS must be able to deliver an RDF graph further describing a resource with only a given URL which may be described in any dataset. This operation takes a resource identifier in URL form as argument. The operation looks up all datasets containing graph elements describing the resource and evaluates their access policies. It then delivers a result containing graph elements describing this resource, if a) elements describing this resource are stored and b) the authenticated user is authorized to view a subset of these elements. Similar to the query operation, authentication is optional here.

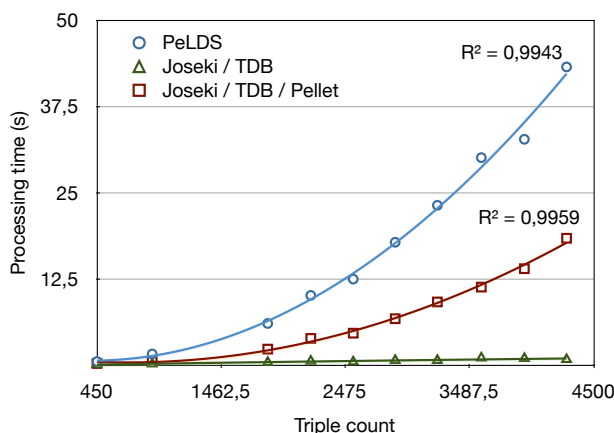
## 5 Evaluation

We have implemented a prototype of the PeLDS system described in Section 4 as a Java application. All operations are performed using the HTTP protocol. This prototype supports all specified API operations and is able to evaluate the PsSF access policy format for each stored dataset, thus satisfying the identified functional requirements. Policy evaluation and OWL reasoning is performed by the Pellet reasoning program [4]. W3C standards are obeyed and supported where applicable, for example the SPARQL query language, the SPARQL results format, the various RDF serializations like RDF/XML and N3, the SPARQL HTTP protocol, and the SPARQL/Update update language. In this section, we evaluate system security, system performance, and describe the distributed address book we have implemented as a demonstration application on the basis of the PeLDS prototype.

### 5.1 System Performance

Determining the system performance for our prototype is not an obvious task, as the only comparable system mentioned in [16] is not available for testing. However, systems supporting a subset of PeLDS' features are available, hence we were able to test such a system for comparison. The SPARQL server Joseki [8] supports querying and modification of RDF storage systems over an HTTP interface and data separation in multiple datasets, but not evaluation of access policies. Joseki was backed by the triple store Jena TDB and - optionally - the reasoning program Pellet [4] for OWL inference. The test was intended to show the additional effort required for the evaluation of our access policies.

To make query results comparable for all test runs, a special access policy was installed in the PeLDS prototype allowing read and write access to the stored data without any authentication. The test data generator included in the Berlin SPARQL Benchmark



**Fig. 2.** Query performance

[3] was used to generate a sufficient amount of triples for testing in datasets of different sizes. Each dataset was imported into the respective system and a simple SPARQL query returning all stored and inferred triples was executed at least three times for each dataset. The shortest time required to complete the query was taken as a test result, in order to acknowledge caching strategies. Test results are given as a scatter plot in Figure 2 with the x-axis describing the amount of triples and the y-axis the time required to complete the test query. As the result of the  $R^2$  least squares fitting test, an approximation to a polynomial of second degree is also plotted.

The difference in results of the Joseki instance with and without reasoning support illustrates the amount of time required for reasoning in general. The PeLDS prototype requires additional time for access policy evaluation, however, this effort only increases in a linear fashion as the dataset grows. The approximation tests yielded polynomial complexity for both Joseki and PeLDS which is mainly attributed to reasoning activities.

## 5.2 Security Considerations

PeLDS's security directly depends on secure authentication. If an attacker is able to circumvent the FOAF+SSL authentication scheme, unauthorized access is possible. FOAF+SSL relies on dereferencing the URL identifying a user, if an attacker gains control over that URL, for example by manipulating DNS entries, he is able to take that identity. However, recent improvements to the Internet architecture such as DNSSEC aim at impeding such attacks. Another attack vector are the queries specified. They do not pose a security threat by themselves, as they are not evaluated over the global database. Queries can only "see" a temporary graph containing only the elements of the specified datasets authorized for the current user.

### 5.3 Demo Application: Distributed Address Book

In order to demonstrate the capabilities of the PeLDS system, we have implemented a demo application with a storage layer solely based on PeLDS. This application implements a distributed address book as a web application. Users can manage their contact profiles and contacts within this address book. Users are identified by a URL and all information about the contacts is retrieved in real-time from the server their corresponding profile is stored on. Users can organize their contacts in groups and assign visibilities to each data item stored in their profile. For example, a user may define her telephone number to be private and only visible to her family.

All user data is stored within a PeLDS instance. Storage communication is handled via SPARQL and SPARQL/Update, respectively. The privacy settings the users define are translated into PeLDS access policies and activated for their personal data. Access to user data is controlled by PeLDS, thus only clients properly identifying themselves and authorized can retrieve protected information. Data integration and user identification is performed using Linked Data principles, all a user has to know to add another user to his address book is the URL describing her.

In contrast to popular systems, our Distributed Address Book leaves all personal data under the control of each user, a central instance is not required. Also, an arbitrary client program capable of displaying RDF information can be used to view and manage address book entries, given that this client supports the usage of SSL certificates.

## 6 Conclusion and Outlook

Following a survey of existing work in the area of access control for Semantic Web storage systems, we commenced on detailing the different types of access policies and the methods for data classification within the contexts of RDF graphs used to represent Semantic Web content. We then explained the Semantic Web Rule Language as a possible candidate for a rule format in access policies. Our concept of a Policy-enabled Linked Data server was described. PeLDS consists of our access policy language PsSF, which extends SWRL by adding custom predicates for data classification, an OWL policy schema, the FOAF+SSL authentication mechanism and a high-level API definition of the different operations provided. This concept was implemented as a prototype and evaluated for system performance in comparison with an established solution, Joseki. Our Distributed Address Book based on PeLDS was introduced to show the kind of both distributed and privacy aware applications now possible. The PeLDS prototype and the Address Book are available as open source software and can be downloaded at <http://www.pelds.org>.

We would like to extend both the PeLDS concept and prototype with more features such as negation support within our policy language. Performance optimizations are another area of future work, as scalability was not the main goal for the implementation of the prototype. API operations for detailed modifications of single rules instead of whole access policies may also be desirable.

**Acknowledgments** We would like to thank the anonymous reviewers, Andy Seaborne for his answers, and Benjamin Nowack for the approval of our patches.

## References

1. Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, and Daniel Olmedilla. Enabling advanced and context-dependent access control in RDF stores. In *ISWC2007/ASWC2007*, pages 1–14, 2007.
2. Tim Berners-Lee. Linked data, 2006. <http://www.w3.org/DesignIssues/LinkedData.html> accessed on 2010-04-20.
3. Christian Bizer and Andreas Schultz. The Berlin SPARQL benchmark. *International Journal On Semantic Web and Information Systems - Special Issue on Scalability and Performance of Semantic Web Systems*, 2009.
4. Clark and Parsia. Pellet: The open source OWL reasoner, 2009. <http://clarkparsia.com/pellet> accessed on 2010-04-20.
5. Juri Luca De Coi, Daniel Olmedilla, Piero A. Bonatti, and Luigi Sauro. Protune: A framework for semantic web policies. In Christian Bizer and Anupam Joshi, editors, *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
6. Claudiu Duma, Almut Herzog, and Nahid Shahmehri. Privacy in the semantic web: What policy languages have to offer. *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:109–118, 2007.
7. David Ferraiolo and Richard Kuhn. Role-based access control. In *Proceedings of 15th National Computer Security Conference*, 1992.
8. Hewlett-Packard Development Company. Joseki - a SPARQL server for jena, 2009. <http://www.joseki.org> accessed on 2010-04-20.
9. James Hollenbach, Joe Presbrey, and Tim Berners-Lee. Using RDF metadata to enable access control on the social semantic web. In *Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK2009)*, October 2009.
10. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosf, and Mike Dean. SWRL: A semantic web rule language, 2004. <http://www.w3.org/Submission/SWRL> accessed on 2010-04-20.
11. Amit Jain and Csilla Farkas. Secure resource description framework: an access control model. In *SACMAT'06*, 2006.
12. Lalana Kagal, Timothy Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–74, 2003.
13. Donald C. Latham. *Trusted Computer System Evaluation Criteria (Orange Book)*. Department of Defense, 1985.
14. Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2005.
15. Hannes Mühleisen. Zugriffsrichtlinien und Authentifizierung für Linked-Data-Systeme. Master's thesis, Humboldt-Universität zu Berlin, <http://muehleisen.org/da-hm-ld.pdf>, 2009.
16. Pavan Reddivari, Tim Finin, and Anupam Joshi. Policy-based access control for an RDF store. In *Proceedings of the IJCAI-07 Workshop on Semantic Web for Collaborative Knowledge Acquisition*, January 2007.
17. Henry Story, Bruno Harbulot, Ian Jacobi, and Mike Jones. FOAF+SSL: RESTful authentication for the social web. Submitted to Semantic Web Conference 2009, 2009.
18. Amin Tootoonchian, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: better privacy for social networks. In *CoNEXT '09: Proceedings*, pages 169–180, New York, NY, USA, 2009. ACM.