OAuth+UAO: A Distributed Identification Mechanism for Triplestores

Dominik Tomaszuk[1] and Henryk Rybiński[2]

[1] Institute of Computer Science,
University of Bialystok, Poland
dtomaszuk@ii.uwb.edu.pl
[2] Institute of Computer Science,
Warsaw University of Technology, Poland
h.rybinski@ii.pw.edu.pl

Abstract. The Semantic Web gives users and applications the ability to access and retrieve decentralized resources which may be stored in triplestores. This paper describes a simple identification protocol dedicated to triplestores which is universal and appropriate for the distributed environment. We propose a mechanism based on the HTTP standard, extended with OAuth Protocol and Semantic Web ontology. One can optionally adopt Transport Layer Security protocol. We present a scalable method that allows user authentication and authorization to triplestores with data integrity and confidentiality. The identification mechanism enables users to access triplestore data without disclosing authentication and authorization data.

Keywords: Semantic Web, triplestore, ontology, Resource Description Framework, authentication, authorization, access control list.

1 Introduction

A triplestore is a purpose-built database for the storage and retrieval of Resource Description Framework (RDF) data. Much like in the case of other databases, one can find and modify data in triplestores via a query language, such as SPARQL Protocol and RDF Query Language [1,2].

An RDF triple consists of a subject, a predicate, and an object. In [3] the meaning of subject, predicate and object is explained. The subject denotes the resource, the predicate means traits or aspects of the resource, and expresses a relationship between the subject and the object. A collection of RDF statements intrinsically represent a labeled, directed multigraph. The nodes are the subjects and objects of their triples.

Following [3], let U be the set of all URI references, B an infinite set of blank nodes, L the set RDF plain literals, and D the set of all RDF typed literals. U, B, L and D are pairwise disjoint. Let $O = U \cup B \cup L \cup D$ and $S = U \cup B$, then $T \subset S \times U \times O$ is set of all RDF triples.

The RDF syntax and semantics could be extended to *named graphs* [4]. The *named graphs* data model is a simple variation of the RDF data model. The

P. Jędrzejowicz et al. (Eds.): ICCCI 2011, Part I, LNCS 6922, pp. 275–284, 2011.

basic idea of the model consists in introducing a graph naming mechanism, with the note by G, the set of graphs which can be seen as $G = 2^T$. A named graph is a pair $ng = (n, g)$ with $n \in U$ (called name) and $g \in G$.

RDF information providers do not have any explicit way to express any intention concerning information security. In the paper we attempt to define the proper functions to equip the RDF information in triplestore with security means, such as authentication, authorization, data integrity and confidentiality.

In the context of triplestores there is a need to provide security means similar to those specific to classical databases. In this paper a new distributed identification mechanism based on OAuth [5] protocol and access control ontology for triplestores is presented. OAuth allows users to share private resources stored on one site with another site without having to hand out their credentials, typically login and password. An access control ontology can define a list of permissions attached to a role.

The paper is constructed as follows. Section 2 is devoted to related work. In Section 3, we propose a solution for RDF information security with the use of the following standards and tools: (1) authentication and data integrity with the OAuth protocol and optionally confidentiality over Transport Layer Security; and (2) authorization with the User Access Ontology approach. In Section 4 the implementation of the proposal is presented. The paper ends with conclusions.

2 Related Works

In the classical databases one can distinguish three main categories of security: user authentication for querying the database results [6,7], an access control approach [8,9], and encrypting datasets [10]. The authentication of query results is divided into subcategories: cryptographic primitives, based on digital signature [6] and Merkle Hash Trees [7]. The access control approach consists of two subcategories: content-based access control [8] and rule-based access control [9].

For semi-structured databases, which are more suited to web applications, the main research concentrates on access control [11,12,13]. In [11] are defined access restrictions directly on the structure and content. Other ones [12] present varying protection granularity levels. The access control policy is proposed by using a 5-tuple of subject, object, privilege, propagation option and signed access decision. In [13] an access control model provides provisional authorization.

On the other hand, in web applications there are database independent and resource oriented solutions, which focus in decentralized manner on the authentication and/or authorization procedures. Good examples are OpenID [14,15] and OAuth Protocol [15,5]. The idea of OpenID consists in assigning a unique ID to a resource, which takes the form of a unique URL, and is managed by an OpenID provider, which handles the authentication procedure. While OpenID is all about using a single identity to sign into many sites, OAuth is about giving access to a resource without sharing identity at all. The OAuth protocol provides authentication, authorization and data integrity. Yet another proposal in this area is provided in [16] by means of Security Assertion Markup Language

(SAML), which is a standard for exchanging authentication and authorization data between the identity providers and service providers. Another proposal, which is extended to access control support, is XACML [17]. It is a language based on XML for security policies and access decisions. XACML provides security administrators to describe an access control policy once, without having to rewrite it numerous times in different application-specific languages. While this approach may be sufficient for triplestores using XML syntax to store, it is not satisfactory for other triplestores.

In the context of Semantic Web, there are also new proposals for authorization solutions. Some of the papers [18,19] define policy-based access control. [18] defines policies that describe subgraphs on which various operations, such as *insert*, *remove*, *update* and *read*, are identified by specifying RDF patterns. The authors define a set of policy rules, enforced by a policy engine to reach the authorization decisions. Unfortunately, they do not discuss the semantics in a formal way. In [19] triples are not annotated with accessibility information, but the enforcement mechanism is query-based. The policy permissions are injected into the query in order to ensure that the triples obtained are only the accessible ones. Unfortunately, the semantics of the policy are not formally defined in [19]. Yet another approach is provided in [20,21], which respect RDF Schema (RDFS) entailments. In [20], the authors discuss how conflicts can be resolved using the RDFS subsumption hierarchies. This proposal, just like in [18], requires instantiating the RDF patterns. In [21] inferences are computed for an RDF dataset without revealing information that might have been explicitly not permitted.

In the context of Semantic Web, the first attempts towards solving security issues were presented in [22,23]. They are concentrate on assuming explicit and domain-specific trust ratings. An extension of these ideas is presented in [24,25], where the secure authentication protocol WebID[1] was proposed. The solution enables the building of distributed social networks. WebID uses the Web of Trust mechanism [26], but this does not require signing. The friendship relations are not embedded in the signature. Unfortunately, it is not well suited to triplestores. In particular, it cannot be used for larger triplestore resources. Furthermore, WebID only supports authentication. In contrast, here we do not use Web of Trust. We concentrate on defining mechanisms strictly dedicated to triplestores, preserving the feature of using linked data [27] to publish, share, and connect users' and groups' data stored in triplestores. What is important is that, our approach differs from the idea presented in [24,25] in that it does not depend on other users for trust. In addition, WebID uses Transport Layer Security with X.509 certificates which make this form of communication slower and more complicated than our proposal.

3 Distributed Identification Mechanism for Triplestores

In this Section we discuss the idea of using the identification mechanism, as provided by OAuth. Additionally we define a new ontology, devoted to the spec-

[1] WebID is also known as FOAF+SSL.

ification of access control by means of authorization. For the cases where confidentiality is needed, optional, encryption based on Transport Layer Security (TLS) can be used.

3.1 Authentication and Data Integrity over OAuth

In this Section we suggest using OAuth to access a triplestore. It is token-based authentication. That means that a logged-in user has a unique token used to access data from the triplestore. Users access to triplestore data with sharing tokens and without disclosing any identity data. This approach presents a triplestore as a server. It could be, optionally, an authorization endpoint and/or an access control lists repository. A client is an application that uses OAuth to access the triplestore on behalf of the user.

We propose an authorization algorithm using OAuth over Hypertext Transfer Protocol. It consists of the following seven steps:

1. Obtain request token from the triplestore[2] to the client.
2. Redirect client to the authorization endpoint.
3. The server requests user to sign in by using login and password. It is important that in this step login, and password should be encrypted.
4. If login and password are correct, the server associates the user with the role and asks for approval granting to triplestore.
5. Redirect from the authorization endpoint to the client.
6. Exchange request token for access token.
7. The client is ready to request the private data to triplestore.

OAuth take care of the data integrity by signing HTTP requests. The OAuth protocol is secure, because it has tokens (and the fields: *timestamp* and *nonce*) that do not pass login and password, for verifying unique requests. Each token grants access for a specific triplestore resources and for a defined duration.

The main disadvantage of OAuth is that login, password and other settings, such as email, cannot be changed via this protocol.

OAuth over HTTP does not provide confidentiality. The solution to this problem is to use Hypertext Transfer Protocol Secure (HTTPS). This proposal does not force the use of HTTPS and allows using HTTP, when confidentiality is not needed.

The main disadvantage is that HTTPS requires both parties to the communication to do extra work in exchanging handshakes and encrypting and decrypting the messages, making this form of communication slower than it would be without it.

3.2 Authorization: User Access Ontology

In this Section the proposed User Access Ontology is presented (in the sequel denoted by UAO). UAO is an ontology describing roles, their permissions, and

[2] In OAuth it is called service provider.

allowed or permitted actions on triplestores. It allows the description of access control lists for users, without assigning them to a single triplestore and/or other databases. UAO is a descriptive vocabulary expressed in Resource Description Framework (RDF), RDF Schema and Web Ontology Language (OWL). The presented ontology is written in the RDF/XML and Terse RDF Triple Language syntaxes [29].

We define an authorization $a \in AuthZ$ as a tuple of the form $< role, action >$ where $role \in R$ and $action \in ACT$. The user description (`User` class) consists of first name (`firstName` property), last name (`lastName` property) and user name (`userName` property). The most important is user name, because it identifies and associate the authenticated user with access control lists. The user characteristics can be extended to other ontologies, such as FOAF [28]. Users are assigned (`hasRole` property) to roles (`Role` class). Users should have a minumum of one role. Roles may have names (`roleName` property). There is also default policy for the role (`DefaultPolicy` class), which could deny (`Deny` class) or permit (`Permit` class) access to data. It should have exactly one default policy. Roles are assigned (`hasPermission` property) to their permissions (`Permission` class). Let P be the permissions, USR be the user and DP be the default policy, then role $R \in usr, p, dp$ with $usr \in USR$, $p \in P$ and $dp \in DP$. Permissions may have numeric priorities (`priority` property), which prevents conflicts. It should also have filters (`filter` property), which are sets of triple pattern TP or URI references U (see Section 1). Let V be the set of all variables, then $TP \subset (S \cup V) \times (U \cup V) \times (O \cup V)$ and V is infinite and disjoint from U, B, L and D (see Section 1). Permissions may have named graph declaration (`graph` property). When this value is not set, the permisions refer to the default graph.

The permissions are assigned (`hasAction` property) to actions (`Action` class). The actions ACT specify roles which are granted to access the triplestore, as well as what operations are allowed or forbidden on the triplestore. We propose nineteen types of actions, which are based on the SPARQL clauses [1,2] and that can be combined with each other. These types of permission are divided into three groups: graph management (`GraphManage` class), graph modification (`GraphModify` class) and query forms (`QueryFrom` class). The graph management permissions allow the execution of the clauses: CREATE and DROP. The graph management permissions allow the execution of the clauses: INSERT DATA, INSERT, LOAD, DELETE DATA, DELETE, DELETE WHERE, CLEAR and DELETE/INSERT. The query form permissions allow the execution of the read-only query forms: SELECT, CONSTRUCT, ASK and DESCRIBE. All SPARQL clauses are reflected in classes and presented in Table 1.

Prohibit or permit action names are identical with classes (see Table 1). All actions have own parameters, which are identical to filter values. All actions with parameters are presented in Table 2.

Listing presents the example of access control list in Terse RDF Triple Language. This information is stored in an ACL repository and it could be part of the triplestore. The User Access Ontology is presented in Fig. 1.

Table 1. classes reflected in SPARQL

Class	Superclass	SPARQL clauses
GraphManage	Action	CREATE, DROP
Create	GraphManage	CREATE
Drop	GraphManage	DROP
GraphModify	Action	INSERT [DATA], LOAD, CLEAR, DELETE [DATA] [WHERE], DELETE/INSERT
DeleteInsert	GraphModify	DELETE/INSERT
Load	GraphModify	LOAD
Clear	GraphModify	CLEAR
Add	GraphModify	INSERT DATA. INSERT
Remove	GraphModify	DELETE DATA, DELETE, DELETE WHERE
InsertData	Add	INSERT DATA
Insert	Add	INSERT
DeleteData	Remove	DELETE DATA
Delete	Remove	DELETE
DeleteWhere	Remove	DELETE WHERE
QueryFrom	Action	ASK, DESCRIBE, SELECT, CONSTRUCT
Ask	QueryFrom	ASK
Describe	QueryFrom	DESCRIBE
Select	QueryFrom	SELECT
Construct	QueryFrom	CONSTRUCT

Table 2. Actions with parameters

Class	Action with parameters
GraphManage	GraphManage(ng) with $ng \in U$
Create	Create(ng) with $ng \in U$
Drop	Drop(ng) with $ng \in U$
GraphModify	GraphModify(ng, u, tp_1, tp_2) with $ng \in U$, $u \in U$, $\{tp_1 : tp_1 \in TP\}$, $\{tp_2 : tp_2 \in TP\}$
DeleteInsert	DeleteInsert(ng, tp_1, tp_2) with $ng \in U$, $\{tp_1, tp_2 : tp_1, tp_2 \in TP\}$
Load	Load(ng, u) with $ng \in U$, u ∈ U
Clear	Clear(ng) with $ng \in U$
Add	Add(ng, tp) with $ng \in U$, $\{tp : tp \in TP\}$
Remove	Remove(ng, tp) with $ng \in U$, $\{tp : tp \in TP\}$
InsertData	InsertData(ng, tp) with $ng \in U$, $\{tp : tp \in TP\}$
Insert	Insert(ng, tp) with $ng \in U$, $\{tp : tp \in TP\}$
DeleteData	DeleteData(ng, tp) with $ng \in U$, $\{tp : tp \in TP\}$
Delete	Delete(ng, tp) with $ng \in U$, $\{tp : tp \in TP\}$
DeleteWhere	DeleteWhere(ng, tp) with $ng \in U$, $\{tp : tp \in TP\}$
QueryFrom	QueryFrom(ng, tp) with $\{ng : ng \in U\}$, $\{tp : tp \in TP\}$
Ask	Ask(ng, tp) with $\{ng : ng \in U\}$, $\{tp : tp \in TP\}$
Describe	Describe(ng, tp) with $\{ng : ng \in U\}$, $\{tp : tp \in TP\}$
Select	Select(ng, tp) with $\{ng : ng \in U\}$, $\{tp : tp \in TP\}$
Construct	Construct(ng, tp) with $\{ng : ng \in U\}$, $\{tp : tp \in TP\}$

Example of access control list

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix uao: <http://example.org/uao#> .

_:u01 a uao:User ;
      uao:firstName "John" ;
      uao:lastName "Smith" ;
      uao:userName <http://example.org/card#me> ;
      uao:hasRole _:r01 .

_:r01 a uao:Role ;
      uao:roleName "teachers" ;
      uao:hasDefaultPolicy uao:Permit ;
      uao:hasPermission _:p01 .

_:p01 a uao:Permission ;
      uao:priority "10"^^xsd:int ;
      uao:graph "$g" ;
      uao:filter "($s $p $o)" ;
      uao:hasAction uao:Select .
```

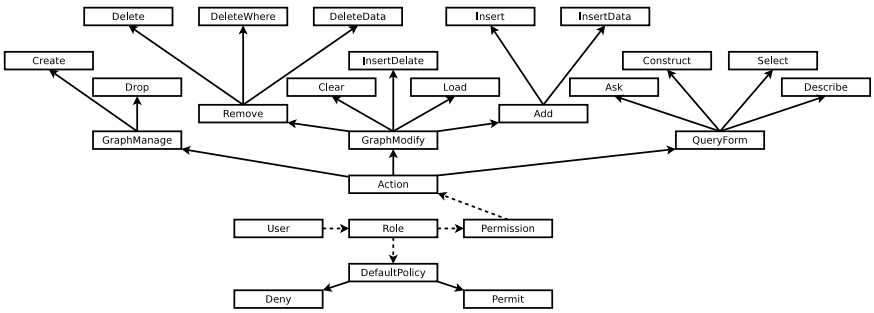

Fig. 1. User Access Ontology

4 Implementation

Now we present the implementation of our approach. We used PHP5 as the development platform. The system consists of the following five parts: query engine, access control lists repository (applying User Access Ontology), protected resources (a part of triplestore), authorization endpoint and HTTP client. Our implementation of the proposed prototype is made of three internal modules, namely query engine, access control lists repository and protected resources. Fig. 2 presents the system architecture.

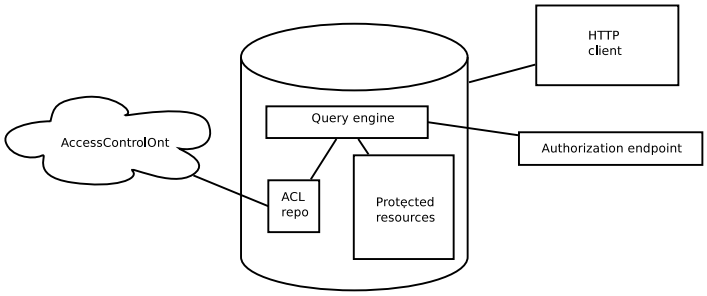

Fig. 2. System architecture

Our modules are additional layers on top of a document-oriented database (MongoDB), which substitutes the triplestore. The access control lists and protected resources are stored in triplestore. We use external, third-party OAuth authorization endpoint and a web browser as an HTTP client to test the prototype. The UML sequence diagram (Fig. 3) presents the workflow of our implementation. The diagram shows basic stages of interaction between actors:

1. The web browser obtains an unauthorized request token.
2. The authorization point and access control list repository authorizes the request token.
3. The web browser exchanges the request token for an access token.
4. The user executes SPARQL queries.

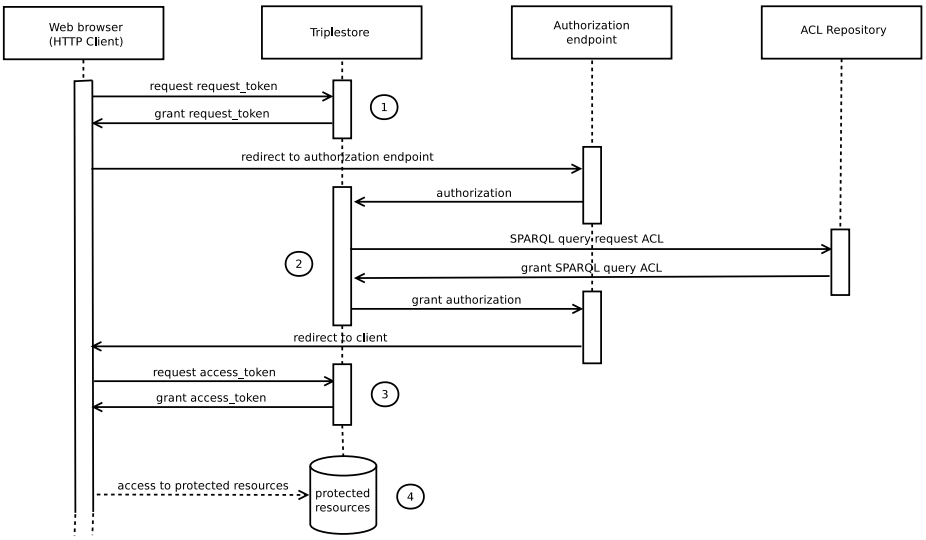

Fig. 3. UML sequence diagram

To enforce an access control, we analyze SPARQL queries and compare to permit or deny actions as exceptions to default policy. Next, the triple patterns from `filter` and/or `graph` properties are mapped to the relevant clauses of SPARQL queries. If this wildcard is true for user permissions, the query is executed.

5 Conclusions

The problem of how to adjust access control to a triplestore has produced many proposals. Most of them are hard to use without dedicated tools, hence making the problem seem difficult and slow. We assume that the triplestores, to be more functional, should provide a mechanism to confirm the identity of a user. This mechanis also confirm restrictions that operate on the RDF graphs. The main motivation for this paper is a lack of such requirements.

We have produced a simple, thought-out RDF standard based and closed triplestores proposal. We believe that our idea is an interesting approach, because it is triplestore independent. We have proposed an identification protocol dedicated to triplestores that is universal and distributed. It uses HTTP, OAuth and ontology. Our proposal can work either with mobile and other devices or a web browser and other software as a triplestore client and server. Another advantage is that users do not have to give a password to third parties. A crucial advantage of the proposal is that the identification mechanism is distributed and adopts linked data. The implementation shows its great potential.

We realize that some further work on this issue is still necessary. We are currently investigating our proposal to support web SPARQL endpoint and RESTful managing RDF graphs.

References

1. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF, World Wide Web Consortium (2008)
2. Schenk, S., Gearon P., Passant A.: SPARQL 1.1 Update, World Wide Web Consortium (2010)
3. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium (2004)
4. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs, provenance and trust. In: 14th International Conference on World Wide Web. ACM, New York (2005)
5. Hammer-Lahav, E.: The OAuth 1.0 Protocol. Internet Engineering Task Force (2010)
6. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. ACM Transactions on Storage (2006)
7. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Special Interest Group on Management Of Data. ACM, New York (2006)
8. Bertino, E., Haas, L.M.: Views and security in distributed database management systems. In: Schmidt, J.W., Missikoff, M., Ceri, S. (eds.) EDBT 1988. LNCS, vol. 303, Springer, Heidelberg (1988)
9. Ahn, G., Sandhu, R.: Role-based authorization constraints specification. ACM Transactions on Information and System Security, TISSEC (2000)

10. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. Database systems for Advanced Applications (2004)
11. Paraboschi, S., Samarati, P.: Regarding access to semistructured information on the web. In: 16th IFIP TC11 Annual Working Conference on Information Security: Information Security for Global Information Infrastrukture (2000)
12. Bertino, E., Castano, S., Ferrari, E., Mesiti, M.: Controlled access and dissemination of XML documents. In: WIDM 1999 Proceedings of the 2nd International Workshop on Web Information and Data Management. ACM, New York (1999)
13. Jajodia, S., Kudo, M., Subrahmanian, V.S.: Provisional authorizations. E-commerce Security and Privacy (2001)
14. Recordon, D., Reed, D.: OpenID 2.0: a platform for user-centric identity management. In: The Second ACM Workshop on Digital Identity Management. ACM, New York (2006)
15. Kaila, P.: OAuth and OpenID 2.0. The Seminar on network security (2008)
16. Cantor, S., Kemp, J., Philpott, R., Maler, E.: Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Organization for the Advancement of Structured Information Standard (2005)
17. Moses, T.: eXtensible Access Control Markup Language (XACML) Version 2.0, Organization for the Advancement of Structured Information Standard (2005)
18. Reddivari, P., Finin, T., Joshi, A.: Policy based Access Control for a RDF Store. In: Proceedings of the Policy Management for the Web Workshop (2005)
19. Abel, F., Luca De Coi, J., Henze, N., Koesling, A.W., Krause, D., Olmedilla, D.: Enabling advanced and context-dependent access control in RDF stores. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 1–14. Springer, Heidelberg (2007)
20. Kim, J., Jung, K., Park, S.: An introduction to authorization conflict problem in RDF access control. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part II. LNCS (LNAI), vol. 5178, pp. 583–592. Springer, Heidelberg (2008)
21. Jain, A., Farkas, C.: Secure resource description framework: an access control model. In: 11th ACM Symposium on Access Control Models and Technologies. ACM, New York (2006)
22. Golbeck, J., Parsia, B., Hendler, J.: Trust networks on the semantic web. In: Klusch, M., Omicini, A., Ossowski, S., Laamanen, H. (eds.) CIA 2003. LNCS(LNAI), vol. 2782, pp. 238–249. Springer, Heidelberg (2003)
23. Richardson, M., Agrawal, R., Domingos, P.: Trust management for the semantic web. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 351–368. Springer, Heidelberg (2003)
24. Story, H., Harbulot, B., Jacobi, I., Jones, M.: FOAF+SSL: RESTful Authentication for the Social Web. In: European Semantic Web Conference (2009)
25. Gamble, M., Goble, C.: Standing on the Shoulders of the Trusted Web: Trust, Scholarship and Linked Data. In: Web Science Conference (2010)
26. Khare, R., Rifkin, A.: Weaving a Web of trust. World Wide Web Journal - Special issue: Web security: a matter of trust (1997)
27. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems (2009)
28. Brickley, D., Miller, L.: FOAF Vocabulary Specification 0.98. FOAF Project (2010)
29. Beckett, D., Berners-Lee, T.: Turtle - Terse RDF Triple Language. In: World Wide Web Consortium (2008)