An Authorisation and Access Control Framework for Information Sharing on the Semantic Web

Owen Sacco
DERI, National University of Ireland, Galway
owensacco@deri.org

John G. Breslin
National University of Ireland, Galway
john.breslin@nuigalway.ie

***Abstract*—The Semantic Web brought about open data formats which give rise to an increase in the creation and consumption of structured data. This structured data is easily accessible from SPARQL endpoints, which are considered as the main Web Services in the Semantic Web. Most SPARQL endpoints are publicly available and do not provide fine-grained authorisation and access control enforcement to protect user's personal information. Although a substantial amount of work exists on access control and authorisation on the Web, these cannot be applied directly to structured data due to the different nature of how the data is formatted. In this paper, we present our authorisation and access control framework for Web Services in the Semantic Web. We present several vocabularies that model the different aspects of the authorisation sequence. We also extend our Privacy Preference Manager (PPM) that handles the authorisation sequence for clients accessing the resource owner's personal information in RDF stores.**

Keywords–Access Control, Authorisation, Privacy, Semantic Web.

I. INTRODUCTION

The Semantic Web [13] provides formats to enrich information on the Web by annotating Web data with additional meaning that can be processed by machines to offer enhanced services for data sharing and interoperability amongst different data sources. This enables Web agents or Web enabled devices to process this meaning to carry out complex tasks automatically on behalf of users. Moreover, by means of Semantic data, information can be merged easily from heterogeneous sources based on the relationships amongst the data, even if the underlying data schemas differ. The most commonly used meta-formats for the Semantic Web is the Resource Description Framework (RDF) [11]. RDF describes resources on the Web and the relationships between them in the form of *graph models*. RDF uniquely identifies resources on the Web, such as people, events, blog posts, reviews and tags by means of Uniform Resource Identifiers (URIs). Each resource can link to other resources by referring to the URI of the specific resource. The advantage of linking resources is that different datasets can be linked and hence create the *Web of Data*.

RDF data can be queried using SPARQL [9]. SPARQL queries take the form of a set of triple patterns called a basic graph pattern. Most RDF data stores contain a SPARQL Endpoint which accept incoming SPARQL queries over HTTP and return back SPARQL query results. SPARQL Endpoints can be considered as the main Web Services in the Semantic Web. The majority of SPARQL Endpoints follow the RESTful architecture and are publicly accessible.

Web Services allow third-party applications to access and use personal information of end-users. This brought about a need for authorisation mechanisms that allow users to authorise applications to use their information on their behalf. The most common authorisation method is OAuth [3] that provides resource owners to authorise clients to use their protected resources on their behalf without sharing their credentials with the client.

Most SPARQL Endpoints do not use authorisation mechanisms such as OAuth to protect sensitive resources since most SPARQL Endpoints are publicly available. Although the idea behind the Semantic Web is to publish open datasets, this causes a risk to sensitive and personal resources. Several access control models have been proposed, however, most of them do not provide fine-grained authorisation mechanisms for RDF graphs.

In this work, we present our Authorisation and Access Control framework that builds upon our previous work [24]. This framework provides an OAuth architecture for personal information that can be accessed from Web Services in the Semantic Web. Therefore, our framework provides a fine-grained authorisation and access control platform for RDF graphs. Our work is based on the attribute-based access control (ABAC) model since the nature of the Semantic Web is to provide an open environment without knowing *a priori* who will access the data. In this work, we propose several vocabularies to model each authorisation process including credentials and the scope of what a client can access. We also present how this framework is implemented on top of SPARQL Endpoints. Although our main focus for this research paper is on personal data in the Semantic Web, our work can be applied to any use case that requires authorisation for data from Web Services in the Semantic Web.

This paper is structured as follows: Section II provides an overview of some related work. Section III provides an overview of our authorisation and access control framework. In Section IV, we explain how the authentication in this framework functions. Section V presents our vocabularies for describing both the client and the Web Service authorisation component preferences. In Section VI, we explain our vocabularies that model the scope, which provides the necessary permissions for the client (authorised by the user) to access the user's personal information. In Section VII, we explain in detail the authorisation process using the vocabularies described in this paper. Section VIII concludes the paper.

 ISBN: 978-1-61208-432-9

II. Related Work

The authors in [26] propose a method that uses OAuth to authorise clients to access data from triple stores. This work uses usernames and passwords for authentication rather than leveraging the benefits of WebID. Moreover, the OAuth protocol implemented in this work does not use client credentials. Furthermore, the access control ontology presented in this work does not provide fine-grained access control on protected resources but restricts SPARQL clauses. This ontology is also a role-based model that relies on pre-defined access control policies bound to the user's roles.

Similarly, the OpenLink Software Virtuoso [12] RDF store provides OAuth for its SPARQL endpoint. Although this store uses WebID for authentication, the user can authorise or decline the requested SPARQL query rather than fine-grained authorisation for specific resources.

The eXtensible Access Control Markup Language [20] is an XML based language for expressing a large variety of access control policies. Although the XACML is widely used [16], [18], it does not provide the necessary elements to define fine-grained access control statements for structured data. It also does not contain enough semantics to describe what the actual access restriction is about and also does not semantically define which attributes a client or requester must satisfy. Moreover, this method does not provide an authorisation method to authorise third-party applications to use resource owner's data on their behalf.

The Protocol for Web Description Resources (POWDER) [2] is designed to express statements that describe what a collection of RDF statements are about. The descriptions expressed using this protocol are text based and therefore do not contain any semantics that can define what the description states.

The authors in [19] propose a privacy preference formal model consisting of relationships between objects and subjects. The proposed formal model however does not provide any authorisation mechanism for third-party applications to access RDF stores. The authors in [15] propose an access control framework by specifying privacy rules using the Semantic Web Rule Language (SWRL) [1]. This approach also does not provide any authorisation mechanisms for third-party applications to access SPARQL Endpoints. Moreover, this approach relies that the system contains a SWRL reasoner. In [17] the authors propose a relational based access control model called `RelBac` which provides a formal model based on relationships amongst communities and resources. This approach requires to specifically define who can access the resource(s) but does not provide any authorisation mechanism for third-party applications to access SPARQL endpoints.

The authors in [14] propose an expressive logic-based technique for the specification of security properties. However, this approach requires another language and framework in order to process security policies, unlike our work that uses RDF to express the polices that can be processed by the SPARQL engine.

III. Authorisation and Access Control Framework – Overview

The Authorisation and Access Control Framework, illustrated in Figure 1, provides authentication and authorisation

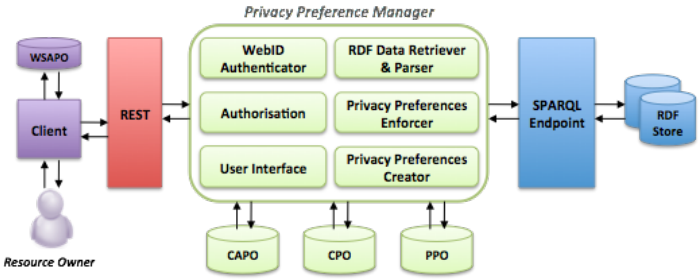

Figure 1. The Authorisation and Access Control Framework

mechanisms for RDF data. It is designed to be deployed over SPARQL Endpoints as a Web Service to control and filter data accessed by third-party clients. The authorisation flow in this framework follows the OAuth 2.0 [3] sequence. This Authorisation and Access Control framework is developed within the Privacy Preference Manager (PPM).

The PPM [22], [24] is an access control manager that provides users to create fine-grained access control policies, known as privacy preferences, for RDF data. The manager also filters requested data by returning back only a subset of the data which is granted access as specified by the privacy preferences. The PPM was developed as a Web application either as a centralised Web application hosted on a centralised server or in a Federated Web environment where users could host their own manager on servers where they desire.

The PPM provides users to login to their manager and create their privacy preferences for their RDF data. Moreover, they could also login to other user's manager and request data. The PPM would return back only that data which the user is granted access – based on the privacy preferences.

The PPM also offers an API which could be integrated within other applications. However, the PPM does not provide any Web Service API whereby third-party applications could call the PPM over HTTPS to benefit from the access control features which it offers. Moreover, it does not provide any authorisation methods to enable users to authorise third-party applications which information they could access and consume. Therefore, we have extended the PPM to provide RESTful methods where third-party applications could send their SPARQL query to the manager and receive back the filtered RDF data. Furthermore, we have extended the PPM with an authorisation component to handle the authorisation process of third-party applications.

The PPM is designed to handle the requests sent to SPARQL Endpoints from client applications using the RESTful architecture. The PPM handles the requests by (1) requesting the resource owner (i.e. user) to authenticate with the PPM; (2) requesting the resource owner to authorise the client which data it can consume; and (3) sends back a filtered subset of the data which the client is authorised to access. The SPARQL Endpoints should be configured to accept requests sent only from the PPM.

The authentication and authorisation sequence in our framework, as illustrated in Figure 2, consists of: (1) the resource owner (i.e. user) requests a service from the client; (2) the client sends a request for temporary credentials to the PPM – the request includes the client credentials that identify

 ISBN: 978-1-61208-432-9

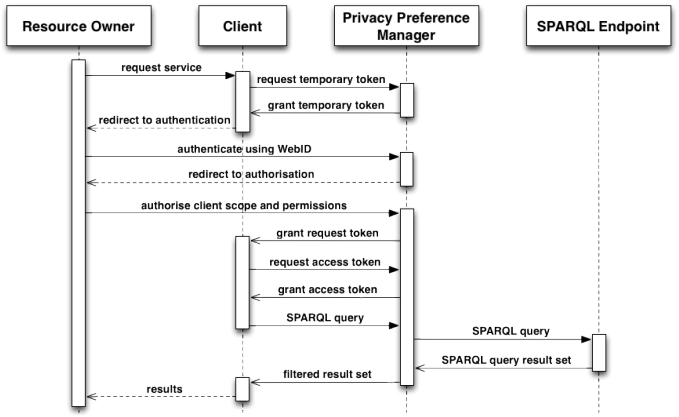

Figure 2. The Authorisation and Access Control Sequence

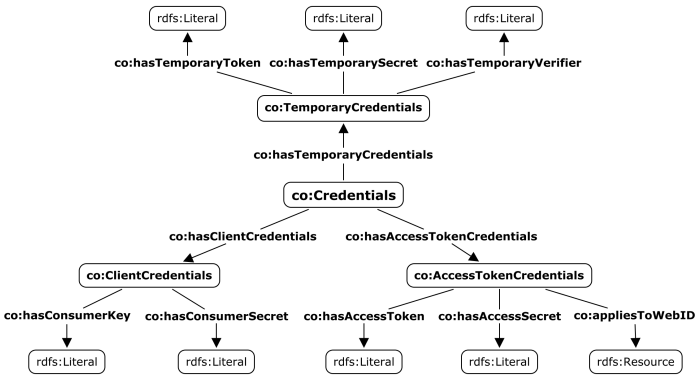

Figure 3. Credentials Ontology (CO)

the client, and the temporary credentials identify the authorisation sequence; (3) the temporary credentials are granted to the client; (4) the client redirects the resource owner to the PPM – the redirect request includes the client's callback URI and the temporary credentials; (5) the resource owner authenticates with the PPM; (6) the resource owner authorises the client by selecting which scope and permissions that will be granted to the client; (7) the PPM sends back the temporary token including a verifier to the client; (8) the client then exchanges the verified temporary token to the access token credentials by sending a request to the PPM – the request includes the temporary token and the verifier; (9) the PPM sends back the access token credentials to the client; (10) the client then sends the SPARQL query (together with the access token credentials) to the PPM; (11) the PPM sends the SPARQL query to the SPARQL Endpoint and the SPARQL Endpoint sends back the query result; (12) the PPM sends back the client only a filtered result set based on what the resource owner had authorised the client which data it can access; and (13) the client renders the service and displays the results to the resource owner.

IV. AUTHENTICATION

Most Web applications request users to provide a username and password in order to authenticate themselves into the system. In Semantic Web applications, the WebID protocol [25] is used as an authentication method. It provides a mechanism whereby users can authenticate using FOAF and X.509 certificates over SSL. The digital certificates contain the public key and a URI that points to the location where the FOAF profile is stored. The WebID authentication mechanism parses the WebID URI from the certificate and retrieves the FOAF profile from its location. The public key in the certificate and the public key in the FOAF profile are checked to grant the user access if the public keys match. The WebID certificates can be self-signed certificates.

Once the resource owner is redirected to authenticate with the PPM, the resource owner is requested to provide a WebID certificate. The PPM's authentication module handles the WebID authentication process by using a WebID verifier that checks that the keys in both the certificate and the FOAF profile match. The advantage of using URIs to identify resource owners is that it eliminates the users to register or create multiple accounts on various servers. If the keys match, then the resource owner is authenticated with the PPM.

Definition 1: Authentication. Let PPM be a PPM instance, $Cert$ an SSL digital signed certificate, O a resource owner identified by a URI and P a resource owner's FOAF profile. Let $Certificate(Cert, O)$ mean that $Cert$ is the SSL certificate of O, $Profile(P, O)$ mean that P is the profile of O, $Verify(Cert, P)$ mean that the public key in $Cert$ is verified with the public key in P and $Authenticate(PPM, O)$ mean that O is authenticated with PPM. Thus, Authentication is defined:

$$\textit{Certificate(Cert,O)} \wedge \textit{Profile(P,O)} \wedge \textit{Verify(Cert,P)} \Rightarrow \textit{Authenticate(PPM,O)} \quad (1)$$

V. MODELLING AUTHORISATION PREFERENCES

In this section we present: (1) the *Credentials Ontology (CO)* which is a light weight vocabulary to describe both the client and the Web Service (i.e. the PPM) credentials; (2) the *Client Authorisation Preferences Ontology (CAPO)* which is a light weight vocabulary to describe the client details when a client registers with the PPM; and (3) the *Web Service Authorisation Preference Ontology (WSAPO)* which is a light weight vocabulary to describe the details of the Web Service authorisation component (i.e. the PPM) to be used by the client during the authorisation process.

A. Credentials Ontology (CO)

The *Credentials Ontology (CO)* [7], illustrated in Figure 3, is a light weight vocabulary to describe three types of credentials: (1) *temporary or request token credentials*; (2) *client credentials*; and (3) *access token credentials*.

The *temporary or request token credentials* identify an authorisation sequence. These tokens are randomly generated for each authorisation request. The *client credentials* identify a particular client. These credentials are created when a client registers with a PPM in order to be able to access the data stored within the SPARQL Endpoint. The *access token credentials* are generated by the PPM each time after the resource owner authorises the client to use his/her personal data on his/her behalf. The access token credentials identify the scope and permissions which the resource owner granted the client at a particular instance.

The *Credentials Ontology (CO)* provides the following classes and properties to describe the three types of credentials:

- `co:Credentials` is the main class of CO and the credentials described using this vocabulary will be instances of this class.
- `co:TemporaryCredentials` is a class that describes the temporary or request token credentials. This class provides the `co:hasTemporaryToken` property that defines an identifier for an authorisation request. This identifier is generated whenever the client requests an authorisation sequence. The `co:hasTemporarySecret` property defines the shared secret generated by the PPM for the authorisation request. This shared secret is used for signing the authorisation requests. This class also provides a `co:hasTemporaryVerifier` property that defines a verification identifier generated by the PPM once the resource owner authenticates and completes the authorisation process.
- `co:ClientCredentials` is a class that describes the client credentials. This class provides the `co:hasConsumerKey` property that defines an identifier for a client sending requests to the SPARQL Endpoint through the PPM. This identifier is generated when the client registers with the PPM to consume the data from the SPARQL Endpoint. Therefore, the client must store this identifier and use it whilst sending requests to the PPM. This class also provides the `co:hasConsumerSecret` property that defines the shared secret generated by the PPM. This shared secret is also generated when the client registers with the PPM and it is used for signing the requests. Similar to the consumer key, the client must store this shared secret.
- `co:AccessTokenCredentials` is a class that describes the access token credentials. This class provides the `co:hasAccessToken` property which describes the identifier to the client's authorised scope and permissions authorised by the resource owner. This class also provides `co:hasAccessSecret` property which describes the shared secret for signing the requests after the authorisation process is complete. Both the `access token` and the `access secret` are generated by the PPM after the resource owner completes the authorisation sequence. This class also provides `co:appliesToWebID` property which links the access token credentials to the resource owner's WebID URI who authorised the client.

B. Client Authorisation Preferences Ontology (CAPO)

The *Client Authorisation Preferences Ontology (CAPO)* [5], illustrated in Figure 4, is a light weight vocabulary that describes client details which are stored in the CAPO repository – as illustrated in Figure 1. These details are created once the client is registered with the Web Service (i.e. the PPM). The client details are used by the PPM to verify clients during the authorisation process.

The CAPO vocabulary provides the following classes and properties:

- `capo:Client` is the main class of CAPO and instances of this class define clients that can make use of the authorisation sequence of the Web Service (i.e. the PPM).

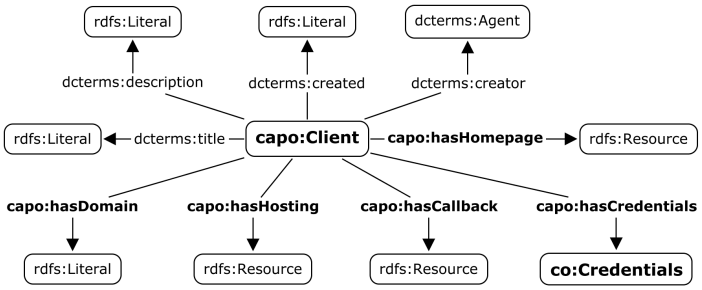

Figure 4. Client Authorisation Preferences Ontology (CAPO)

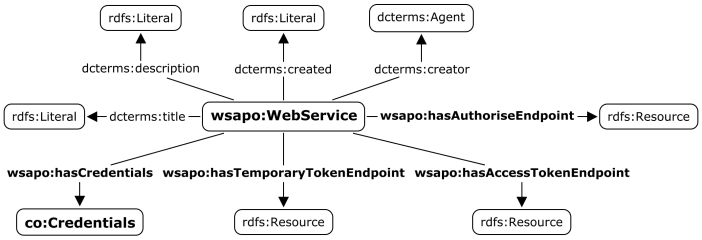

Figure 5. Web Service Authorisation Preferences Ontology (WSAPO)

- `capo:hasDomain` is a property that defines the client's domain. This is used as additional security to allow requests received only from this domain.
- `capo:hasHosting` is a property that defines the URI where the client is hosted on.
- `capo:hasCallback` this property defines the client's callback URI. Although the callback URI is passed within the requests, this is also used for additional security since the callback URI in the request must match the callback URI defined using this vocabulary.
- `capo:hasCredentials` this property defines the client's credentials defined using the Credentials Ontology (CO) which are generated on registration with the Web Service (i.e. the PPM).
- `capo:hasHomepage` this property defines the client's homepage.

Other terms could be used from other vocabularies to define other details such as `dcterms:title` defines the title given to a client; `dcterms:description` defines the client's description; `dcterms:created` defines the date when the client's details were registered; and `dcterms:creator` defines the creator of the client's details.

C. Web Service Authorisation Preferences Ontology (WSAPO)

The *Web Service Authorisation Preferences Ontology (WSAPO)* [10], illustrated in Figure 5, is a light weight vocabulary that describes the details of the Web Service authorisation component which are stored in the WSAPO repository – as illustrated in Figure 1. These details are used by the client during the authorisation process.

The WSAPO vocabulary provides the following classes and properties:

- `wsapo:WebService` is the main class of WSAPO and instances of this class define Web Services that provide the authorisation architecture such as the PPM.

 ISBN: 978-1-61208-432-9

- wsapo:hasCredentials this property defines the client's credentials defined using the Credentials Ontology (CO) which are generated on registration with the Web Service (i.e. the PPM). These are used to identify the client during the authorisation sequence.
- wsapo:hasTemporaryTokenEndpoint is a property that defines the Web Service's temporary token credentials endpoint. This allows a client to request for temporary token credentials.
- wsapo:hasAccessTokenEndpoint is a property that defines the Web Service's access token credentials endpoint. This allows a client to exchange verified temporary token credentials to access token credentials.
- wsapo:hasAuthoriseEndpoint is a property that defines the Web Service's authorisation endpoint. This allows a client to use the authorisation architecture by sending the temporary credential tokens to this endpoint. Once the authorisation is complete, the Web Service will return verified temporary token credentials (i.e. the temporary token credentials together with the verifier).

Other terms could be used from other vocabularies to define other details such as dcterms:title defines the title given to a Web Service; dcterms:description defines the Web Service's description; dcterms:created defines the date when the Web Service authorisation component details were created; and dcterms:creator defines the creator of the Web Service authorisation component details.

VI. Modelling Permissions

Apart from modelling the details of both the client and the Web Service (i.e. the PPM), the authorisation scope and permissions which the resource owner grants the client in order to access the protected resources should be modelled as well. The scope and permissions are modelled using the Client Permissions Ontology (CPO) – explained in this section. This light weight vocabulary uses the Privacy Preference Ontology (PPO) to model the permissions.

A. Privacy Preference Ontology (PPO) – Overview

The Privacy Preference Ontology (PPO) [8], [21], [23] - is a light-weight Attribute-based Access Control (ABAC) vocabulary that allows users to describe fine-grained privacy preferences for restricting or granting access to non-domain specific Linked Data elements, such as Social Semantic Data. Among other use-cases, PPO can be used to restrict part of FOAF profiles records only to clients or users that have specific attributes. It provides a machine-readable way to define settings such as "Provide my personal phone number only to my family" or "Grant write access to my technical blog only to my co-workers".

As PPO deals with RDF(S)/OWL data, a privacy preference, defines: (1) the resource, statement, named graph, dataset or context it must grant or restrict access to; (2) the conditions refining what to grant or restrict (for example defining which resource as subject or object to grant or restrict); (3) the access control privileges; and (4) a SPARQL query, (AccessSpace) *i.e.* a graph pattern representing what must be satisfied by the client or user requesting information. The access control type includes the Create, Read and Write (which also includes Update, Delete and Append) access control privileges.

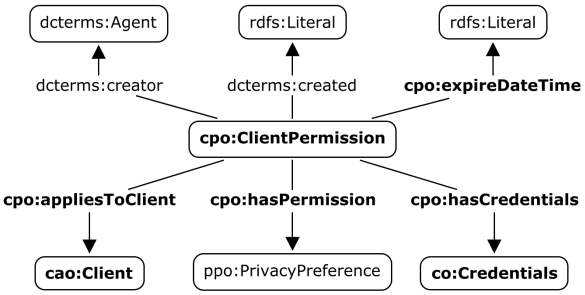

Figure 6. Client Permissions Ontology (CPO)

B. Client Permissions Ontology (CPO)

The *Client Permissions Ontology (CPO)* [6], illustrated in Figure 6, is a light weight vocabulary that describes the scope and permissions which the resource owner grants to the client. The scope and permissions are used by the PPM to grant (or deny) the client access to the resource owner's protected resources.

The CPO vocabulary provides the following classes and properties:

- cpo:ClientPermission is the main class of CPO and instances of this class define the scope and permissions the resource owner has granted a particular client.
- cpo:appliesToClient this property defines which client (as described using the CAPO vocabulary) the scope and permissions apply to.
- cpo:hasPermission is a property that defines the scope and permissions defined using the PPO vocabulary. For example, if the client wants to have access to a particular resource, for instance an email address, the cpo:hasPermission would define a ppo:PrivacyPreference that would ppo:appliesToResource the email address with an acl:Read access control privilege.
- cpo:hasCredentials this property defines the temporary token credentials and the access token credentials defined using the Credentials Ontology (CO) once these are generated by the PPM and granted to the client.
- cpo:expireDateTime this property defines when the scope and permissions expire.

Other terms could be used from other vocabularies to define other details such as dcterms:created defines the date when the scope and permissions were created and dcterms:creator defines the creator.

VII. Authorisation

Whenever the resource owner requests a service from the client, the client reads the temporary token endpoint URI from the WSAPO datastore for that particular Web Service (i.e. PPM). The client sends a request for the temporary token credentials from this endpoint URI and once retrieved, the client redirects the resource owner to authenticate with the PPM.

Once the resource owner is authenticated using WebID as explained in section IV, the PPM first checks within the CPO datastore whether there are any valid access token credentials already granted to that client by that resource owner for

 ISBN: 978-1-61208-432-9

the same request. If valid access token credentials exist, the temporary token credentials are verified and sent to the client. Moreover, the client's permissions defined using CPO are created containing the verified temporary token credentials, the access token credentials that already exist and the permissions which were already granted. Otherwise, the PPM checks if there are any privacy preferences in the PPO store created by the resource owner that authorise the client access to the protected resources. If privacy preferences exist, then the client's permissions defined using CPO are created that link to these privacy preferences. The temporary token credentials are also verified and sent to the client.

When neither any valid access token credentials or privacy preferences exist, then the resource owner is presented with an authorisation page whereby the PPM requests the user to authorise the client's request. The requested SPARQL query is first parsed using the ARC2 [4] SPARQL query parser and presented to the resource owner. The resource owner either authorises the client the whole request; or selects which protected resources the client can access; or denies the whole request. Moreover, the resource owner selects the temporality of the permissions by specifying the expiry date and time. However, any authorised credentials can be revoked any time. Depending on the resource owner's decision, the client's permissions are defined using CPO and the temporary token credentials are verified. The client then exchanges the verified temporary token credentials to access token credentials by requesting the access token endpoint URI.

Whenever the client sends the SPARQL query together with the access token credentials to the PPM, the PPM will send back only what the client is granted to access; based on the client's permissions defined using CPO.

Definition: Authorisation. Let C be a client, O a resource owner identified by a URI, R a resource and A an access control privilege. Let $Request(C, R)$ mean that C requested R, $Resource(R, O)$ mean that R is the resource of O, $Assign(A, O)$ mean that A is assigned by O, $AssignAccess(R, A)$ mean that R is assigned access A and $Authorise(R, C)$ mean that C is authorised R. Thus, Authorisation is defined:

$$\begin{aligned} &\textit{Request(C,R)} \wedge \textit{Resource(R,O)} \wedge \textit{Assign(A,O)} \\ &\quad \wedge \textit{AssignAccess(R,A)} \Rightarrow \textit{Authorise(R,C)} \end{aligned} \tag{2}$$

VIII. Conclusion and Future Work

SPARQL endpoints, which are the most commonly used Web Services in the Semantic Web, are publicly accessible and do not provide any authentication, authorisation and access control functionality. Therefore, in this paper we have presented our authorisation and access control framework that provides resource owners to authorise third-party applications to consume their resources within RDF stores on their behalf. We have presented several vocabularies, namely: (1) the *Credentials Ontology (CO)*; (2) the *Client Authorisation Preferences Ontology (CAPO)*; (3) the *Web Service Authorisation Preferences Ontology (WSAPO)*; and (4) the *Client Permissions Ontology (CPO)* that model several aspects of the authorisation sequence. We have also extended the *Privacy Preference Manager (PPM)* to handle the authorisation sequence for SPARQL endpoints.

As future work, we will enhance the PPM to assert the trustworthiness of third-party applications. The authorisation sequence will become more autonomous since clients will have to satisfy a trust value threshold in order to be authorised to consume the resource owner's protected resources.

Acknowledgment

This work is funded by the Science Foundation Ireland under grant number SFI/08/CE/I1380 (Líon 2) and by an IRCSET scholarship co-funded by Cisco systems.

References

[1] SWRL: A Semantic Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL, 2004. [Online; accessed 31-July-2015].

[2] Protocol for Web Description Resources (POWDER). http://www.w3.org/TR/powder-dr, 2009. [Online; accessed 31-July-2015].

[3] OAuth 2.0 Authorization Framework. http://tools.ietf.org/html/rfc6749, 2012. [Online; accessed 31-July-2015].

[4] ARC 2 RDF Store. https://github.com/semsol/arc2, 2013. [Online; accessed 31-July-2015].

[5] Client Authorisation Preferences Ontology (CAPO). http://vocab.deri.ie/capo#, 2013. [Online; accessed 31-July-2015].

[6] Client Permissions Ontology (CPO). http://vocab.deri.ie/cpo#, 2013. [Online; accessed 31-July-2015].

[7] Credentials Ontology (CO). http://vocab.deri.ie/co#, 2013. [Online; accessed 31-July-2015].

[8] Privacy Preference Ontology (PPO). http://vocab.deri.ie/ppo#, 2013. [Online; accessed 31-July-2015].

[9] SPARQL Query Language for RDF. http://www.w3.org/TR/sparql11-overview, 2013. [Online; accessed 31-July-2015].

[10] Web Service Authorisation Preferences Ontology (WSAPO). http://vocab.deri.ie/wsapo#, 2013. [Online; accessed 31-July-2015].

[11] Resource Description Framework (RDF). https://www.w3.org/RDF, 2014. [Online; accessed 31-July-2015].

[12] OpenLink Software Virtuoso Universal Server. http://virtuoso.openlinksw.com, 2015. [Online; accessed 31-July-2015].

[13] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284:34–43, 2001.

[14] E. Bertino, F. Buccafurri, E. Ferrari, and P. Rullo. A logic-based approach for enforcing access control. *J. Comput. Secur.*, 8(2,3):109–139, Aug. 2000.

[15] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. A Semantic Web Based Framework for Social Network Access Control. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, SACMAT '09*, 2009.

[16] S. Franzoni, P. Mazzoleni, S. Valtolina, and E. Bertino. Towards a fine-grained access control model and mechanisms for semantic databases. In *ICWS 2007*, 2007.

[17] F. Giunchiglia, R. Zhang, and B. Crispo. Ontology Driven Community Access Control. *Trust and Privacy on the Social and Semantic Web, SPOT'09*, 2009.

[18] R. Hebig, C. Meinel, M. Menzel, I. Thomas, and R. Warschofsky. A web service architecture for decentralised identity- and attribute-based access control. In *ICWS 2009.*, 2009.

[19] P. Kärger and W. Siberski. Guarding a Walled Garden Semantic Privacy Preferences for the Social Web. *The Semantic Web: Research and Applications*, 2010.

[20] Oasis. eXtensible Access Control Markup Language (XACML) Version 3.0. 2009.

[21] O. Sacco and J. G. Breslin. PPO & PPM 2.0: Extending the privacy preference framework to provide finer-grained access control for the web of data. In *I-SEMANTICS '12*, 2012.

[22] O. Sacco and A. Passant. A Privacy Preference Manager for the Social Semantic Web. In *Semantic Personalized Information Management Workshop*, SPIM'11, 2011.

 ISBN: 978-1-61208-432-9

[23] O. Sacco and A. Passant. A Privacy Preference Ontology (PPO) for Linked Data. In *Linked Data on the Web Workshop*, LDOW'11, 2011.

[24] O. Sacco, A. Passant, and S. Decker. An Access Control Framework for the Web of Data. In *IEEE TrustCom-11*, 2011.

[25] H. Story, B. Harbulot, I. Jacobi, and M. Jones. FOAF + SSL : RESTful Authentication for the Social Web. *Semantic Web Conference*, 2009.

[26] D. Tomaszuk and H. Rybiński. OAuth+UAO: A Distributed Identification Mechanism for Triplestores. In *ICCCI*, 2011.

 ISBN: 978-1-61208-432-9