International Journal of Electronic Commerce Studies
Vol.4, No.2, pp.281-294, 2013
doi: 10.7903/ijecs.1039

ACCESS CONTROL FOR SHARED ONTOLOGIES

Daniel J. Buehrer
National Chung Cheng University
168 University Rd., Min-Hsiung Township, Chiayi County, Taiwan
dan@cs.ccu.edu.tw

Tzu-Yang Wang
National Chung Cheng University
168 University Rd., Min-Hsiung Township, Chiayi County, Taiwan
wty@cs.ccu.edu.tw

ABSTRACT

Entities and their relationships for various fields of specialization can be described by ontologies. For example, the classes, relationships, and methods for designing a virus may be put into a Protégé ontology. If that design is to be integrated with a missile design, it will require a standard way of sharing and cross-referencing both classes and objects of the ontologies. This should be done in a secure way which only allows access to specified researchers using specified classes, relations, methods, and documentation. In this paper, we introduce a framework for ontology sharing that enables ontologies to easily be deployed on the web. Kernel0 is the ontology of classes, relations, attributes, and objects. It describes its own typing constraints and meta-data in the same way as data. Objects can have relationships to objects of other ontologies. Kernel-1 authorization definitions in our framework are able to dynamically determine the permissions of each user or group for queried objects. Our authorization has the ability to perform inferences via IS-A inheritance and by evaluating implicit relations and operator definitions. All modifications of values in the knowledge base are checked for both appropriate authorization and satisfaction of all typing constraints.

Keywords: Semantic Web, Ontology Sharing, Authorization Inference

1. INTRODUCTION

The Semantic Web usually uses triples to represent entities and ideas. These triples represent relationships among classes (e.g. "subclass" relationships), the user-defined attributes and relationships of both data and metadata, and the "extent" relationship from a class to its objects. The inheritance of data-type constraints into subclasses and sub-relations is built into Kernel0 of the ontology.

Researchers in computer science have proposed some knowledge representations (KR) such as RDF, RDF Schema, and OWL[1, 2]. Before these, there were data models such as the entity-relationship model (E-R Model), which diagrams the relationships between entities. In recent years, ontologies[3], have been used to describe the organization of KR data, which are usually composed of individuals, concepts, and roles (also known as objects, classes, and relations). An implementation of an ontology is a data model of basic metadata of KR. More and more extended knowledge bases can cooperate as long as a proper basic ontology such as Kernel0 is defined and shared among them.

The Semantic Web[4], developed by Sir Timothy Berners-Lee, introduced the concept that knowledge could be more efficiently used if machines were applied to a web of data that used a consistent set of resource pointers. With ontologies related to many domains and knowledge bases, inter-referencing of information can form a Semantic Web where both man and machines can reference and apply knowledge from each other. This paper proposes a framework to organize the sharing web of ontologies for the Semantic Web. We adopt a flexible three-tier architecture that is an extension of a client /server architecture.

In our three-tier architecture, clients can be applications, web GUIs, or mobile applications that communicate with a middle API. The middle layer, the essential part of the server, is responsible for user login, query parsing, proxy queries, distributed queries, merging results, and authorization computations. Current ontology sharing often does not perform authorization checks; as indicated above, however, the knowledge in an ontology is often the heart of an organization's assets. Data as well as data typing must be hidden.

A hidden backend maintains consistency of the data model. It is also in charge of translating data model queries into operations on persistent storage. Depending on the configuration, it stores data in various databases, in a file system, or even in cloud storage such as Hadoop. The upper ontology Kernel0 is based on a common data model, class algebra[5, 6], and it allows objects to cross-reference each other via relations on the Web. A

related query language[6] helps to query data in the ontology on the web via proxy queries. Furthermore, the data model contains implicit relations for which the objects in the range are dynamically calculated. This feature provides inference ability for authorization relationships. The implicit descriptions only specify the semantics of permissions instead of enumerating them. Permissions of requesters on requested objects are dynamically computed according to Kernel1.

Section 2 outlines the communication among client users, an authentication center (AC), and an ontology server as well as the communication between servers to cross-query the ontologies. The protocol for login to the middle server is described in Section 3 as well as the role of the middle server in authorizing inferences. Queries between servers must be constrained using proxies to secure authentication and authorization of users. The hidden backend's responsibilities are described in Section 4, and the authorization inference mechanism is described in Section 5.

2. KNOWLEDGE SHARING ARCHITECTURE

There is already much metadata on the Web about information exchange. Since the beginnings of the Semantic Web, XML's universal resource indicators have been used to act like one-way pointers. Instead, we use Web-based 2-way relationships between objects and classes. An ontology of classes and relations is a very useful way for machines and humans to read structured data. However, for convenience, ontology sharing is now usually public rather than secure. There is, however, the need for authorization in some situations. This paper presents a way of sharing ontologies with authorization control and distributed queries. Our architecture of ontology sharing is presented in Figure 1. The authentication center (AC) is a service that is trusted by all servers, providing verification of users and server identities. Typically, ontology servers are fully public, in which case all ontology information can be queried by anyone. In this paper, however, we consider the case where servers and their ontologies are protected by given policies.

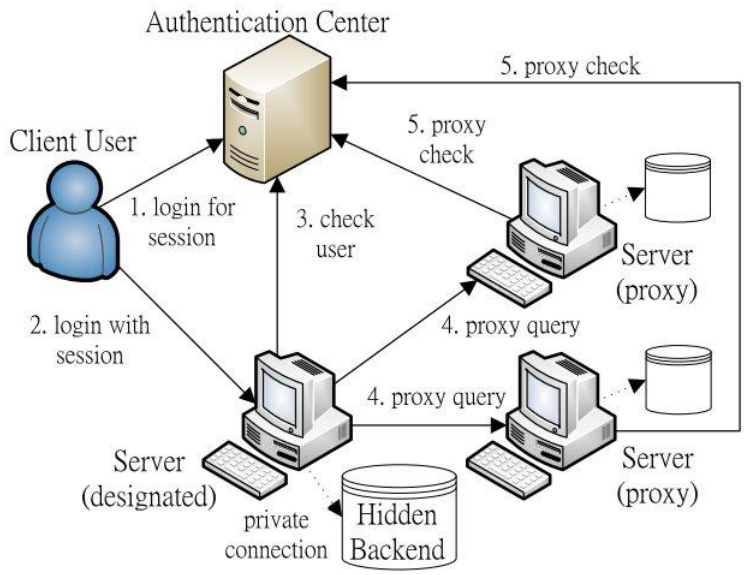

Figure 1. Three-tier architecture

Figure 1 presents the communication for ontology sharing. Objects, classes, and relations in an ontology can be distributed on the Web. This architecture allows the domain and the range of a relation to go across the Web, that is, between two different ontologies. For security reasons, we use a proxy design based on stateful verification, supported by an authentication center. During a query, there may be some objects to be referenced from different ontologies. In this case, the objects in the domain of a relation can be located at a different ontology from the range. When a query needing a proxy is sent from user *C*, a designated server *M* first invokes a proxy login to the server (denoted by *P*) which will be queried. Then, *P* asks the AC whether the key *K* from *M* is verified for proxy purposes. *P* grants *M* access as *C*'s agent for a proxy connection if the check is passed. Again, *P* is a server just like *M*, so it can set itself as a fully public server. Then the proxy check is not necessary. Every server actually has two login services: one for users and another for proxies. If a proxy login is not allowed, then ontology sharing cannot be used. After login of a user or a proxy, queries still involve controlled authorization. The authorization inference is described in Section 5. Furthermore, there is a hidden backend for every server. It takes responsibility for model translation and data storage. Section 4 describes the backend work in detail.

3. THE MIDDLE LAYER

Relative to the simple data persistency provided by the hidden backend, the main part of a Cadabia server is the middle layer, which is in charge of interactions with clients. It handles client logins, queries, and proxy queries to other servers as well as proxy logins from other servers. Objects in other ontologies that are cross-referenced by a relation of a query are accessed during proxy querying. The information of distributed objects is processed by the middle layer, which collects those objects into the result of the proxy query. Authorization, in addition, is applied to control the access rights of client users. The middle layer therefore maintains its own Kernel1 security ontology to protect itself and its ability to edit access rights and capabilities of users and their groups.

3.1 Middle Model

The User API and Proxy API define client login and proxy login, respectively. For a server to be a proxy login, the server must login as a client to another server. The access of clients applying for the proxy API is highly restricted for security reasons. Other than the definition of inverse relationships, modifications to data are expected to be made by users who directly login to a server, not via a proxy server. That is, clients who use the User API to login are granted more permission for update operations. A third way of logging in to the top layer in Figure 2 is to apply a Web service. A Web service login can be set up by using HTTPS connections to the AC. Many technologies, like AJAX, JFS, GWT, can achieve this implementation. However, a Web service which supports stateful verification by te AC is recommended if browser-only apps use OpenID[7], for example. All of these APIs provide querying of ontologies extended from Kernel0, as described in Section 5.

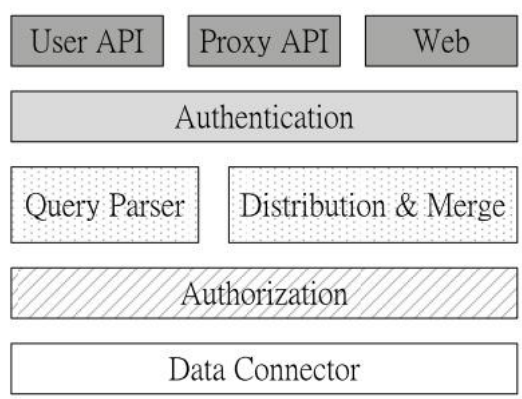

Figure 2. Components of the middle layer

The Kernel1 layer definitions involve authentication and service lookup by AC. The query parser and distribution and merge (D&M) components process queries from clients. The former deals with a query request by a client, parsing it into multiple stages. Each stage traverses a relation and possibly performs selections. Proxy queries are invoked if objects are distributed on other ontologies, and a merge process collects objects in the result. Whether local objects or distributed objects are used, authorization is applied while accessing data on a designated server or proxy server. Authorization above the data connector layer computes access to objects from definitions in the ontology. The data connector process connects to the hidden backend for real data of the ontology. This design not only protects the backend from outside hacks but also allows the backend data to be virtually distributed in a cloud.

3.2 Authorization

Our model takes advantage of the ontology to perform authorization inferences in order to determine the access rights of clients. This authorization component reads the relevant data described in Kernel1 (see Section 5)and calculates the permission of objects which the client intends to access. The result is dynamically calculated from groups[8], which can be set either explicitly or implicitly. Explicit set assignments enumerate the permissions, while implicit sets contain implicit rules for dynamically computing user authorizations based on the current user and the object properties. Details of authorization inference are described in Section 5.

4. THE HIDDEN BACKEND

On the server side, the backend plays the part of a bridge to translate the data model of an ontology into persistent storage. Both SQL and non-SQL backends have an identical API for the middle layer, so it does not have to be concerned with what kind of persistent storage is actually used. This enables persistent storage heterogeneity, and Kernel0-based ontologies behave like a federated database system[9]. Data of the ontology may be stored in a database or other persistent storage such as a file system. Having the backend hidden from outside servers protects it from attacks, so the data would be hard destroy from the outside. We suggest using a private connection between the middle and backend for advanced protection.

4.1 Data Model API and Translation

To enable cross-referencing of objects among ontologies on the Web, a data model called Class Algebra[6] is adopted for the ontology. It uses a human-readable Object Identifier (Oid) consisting of a service URL, an ontology name, a class name, and an object name of the form

"URL:ontology@Class[object]". Backend servers have a different API from the middle API. The data model API is based on the model of Class Algebra to control data. Classes, attributes, and relations in Class Algebra are all treated as definition objects, and class definitions contain specified relations for inheritance. Another component, the query translator, of the data model API processes the request and performs operations on data with regard to target persistent storage. With specified target storage, data are added/deleted/modified in the corresponding storage. Data model operations can be translated into relevant queries of database schema if a database is chosen as persistent storage. Proper database connectors (DBC) help to store data in a related database, while a data process module (DPM) is used to access files in a file system.

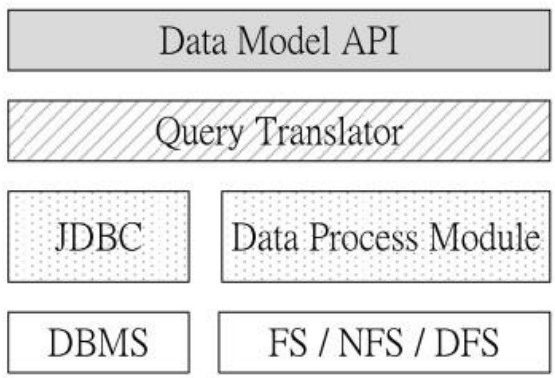

Figure 3. Components of the hidden backend

4.2 Persistent Storage

Due to the private connection between the middle and backend, there is no risk of session attacks on the backend. There will be many operations from access requests via the API to maintain consistency of objects and relationships among them, including inheritance of classes. The components at the bottom layer of the backend are database management system (DBMS) and file systems (FS), including network file systems (NFS) and Distributed File Systems (DFS). DBMS management of data in the database is connected by the DBC component. It can connect to a SQL DB, an XML DB, etc. DBC provides a bridge to access a specified DB, and the query translator uses it to access the data model as well. DPM, moreover, provides an approach to directly operate on data in an FS/NFS. This is because some large objects are not suitable for storage in a DB. They can be more efficiently stored in a file system, for example, by reading partial pieces of a file. On the other hand, there can be some persistent storage preserving data in a specified file format such as JSON or XML. The last module, DFS,

takes charge of distributing data on the Internet. It even can support cloud storage (Apache Hadoop[10], for instance). By implementing such a data distribution module, the backend can execute much more efficiently and stably by using the distribution and replication.

5. AUTHORIZATION

5.1 Upper Ontology

Kenel0 is built according to the Class Algebra[5,6] knowledge representation. Kernel0 is an upper ontology for metadata in which ClassDefn, AttributeDefn, and RelationDefn are three essential classes. All classes (also known as concepts) are defined in a ClassDefn, including "ClassDefn" itself. It describes basic attributes and relations of classes, such as isAbstract, isFinal, hasAttibutes, subclassOf, etc. Relations and attributes are defined in the extents of the RelationDefn and AttributeDefn classes.

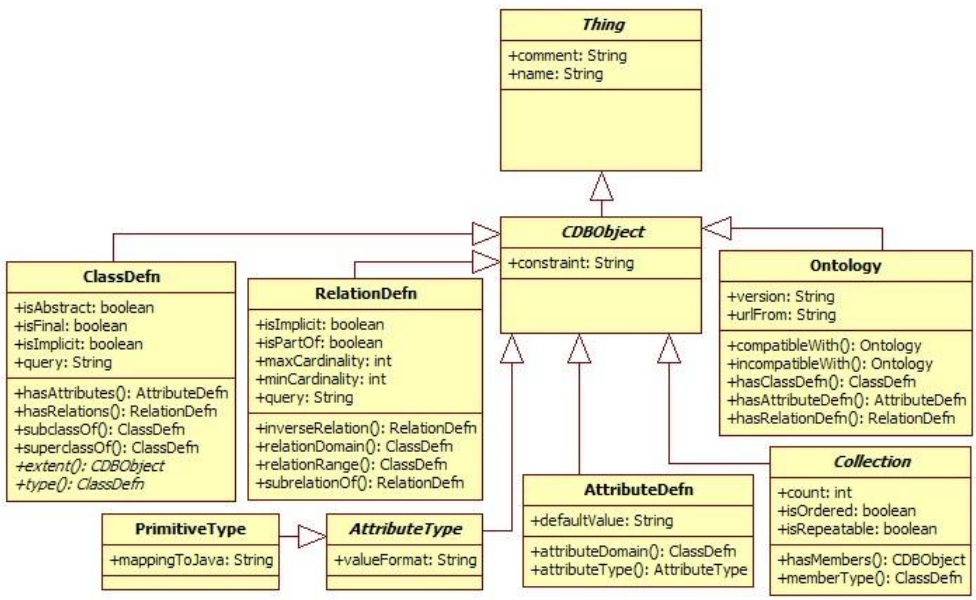

Figure 4. Kernel0

"Thing" is the unique root class of the ontology. The other class, "CDBObject", directly inherits from Thing and plays the role of a basic OBJECT concept for extending an upper ontology such as SUMO. All other classes have to be successors of CDBObject to be considered well-defined in terms of the Cadabia Kernel0. Inheritance of constraints is implemented for *subclassOf*, and inheritance of objects in extents is implemented for *superclassOf* relations. Several other classes are included in Kernel0.

AttributeType is used to define types of attributes, so all attribute types are of type *PrimitiveType*, which inherits from *AttributeType*, which defines common primitive types of attributes. A *Collection* class is an abstract representation for collections with properties such as *isOrdered*, *count*, and *hasMembers*. The Kernel0 upper ontology is outlined in Figure 4. It describes the most primitive ontology for sharing and extending data definitions.

5.2 Authorization Inference

The authorization capability is described in Kernel1. It is a variation of Role-Based Access Control (RBAC[11]) combined withKernel0. There are classes of Kernel1 related to authority, including User, AbstractGroup, UserGroup, ImplicitGroup, ExplicitGroup, BannedGroup and Permission[8], appended to Kernel0 classes; the CDBObject adds *authorities* and *owner* relations. Figure 5 shows the relations among these classes, and Figure 6 shows the whole view of Kernel1. An ExplicitGroup (EG) and ImplicitGroup (IG) define individual users either via an explicit relation or an implicitly computed formula, respectively, while BannedGroup (BG) defines those who are banned. The relationship between permissions and applying objects is defined in five kinds of fundamental permissions inherited from CDBObject. They are *CreatePermission, ReadPermission, UpdatePermission, DeletePermission* and *ExecutePermission*. Instances of *Permission* tell which objects apply what permissions defined in EG, IG, or BG. *AbstractGroup* is an abstract class representing the *GROUP* concept. Such relations of inheritance avoid nesting the references of BG as well. That is, the banned objects will never include any object of BG. Thus, we do not allow multiply negatedBG relations.

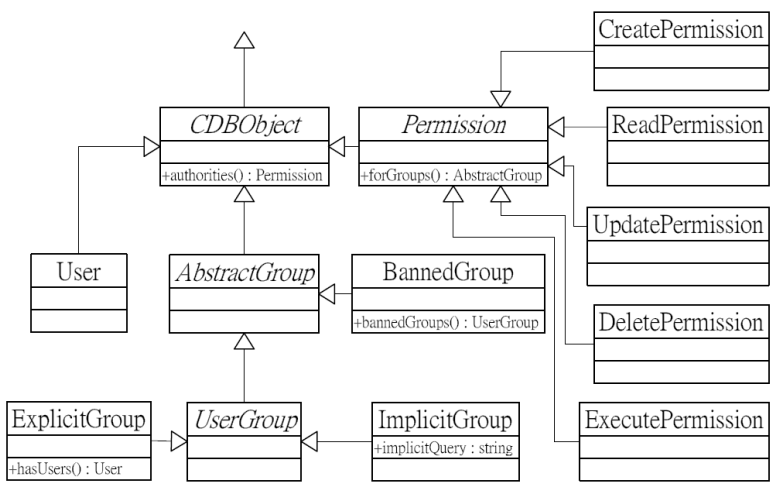

Figure 5. Authority partition of Kernel-1

Each instance of EG explicitly defines a group of users, while each instance of IG defines a group of users implicitly, which is filtered by a query constraint. Relatively, IG has much more dynamic flexibility than EG. Both can be used by instances of BG to determine rejection. EG, IG, and BG support the basic model of authorization inference. Furthermore, permissions also have priority and inheritance. Permissions that are defined in requested objects have high priority, as do banning permissions. The authorization mechanism checks whether a requesting user is banned according to the User Manager and the permissions of requested objects. The priority of banning permissions obviously should be higher than granting permissions. In the case that referenced objects have no defined permissions, it then checks the class to which they belong. If the class still does not define permissions, the permissions inherited from super-classes are applied. The authorization inference calculator computes if a requesting user is banned or granted access, which involves fundamental permissions with respect to the requested object.

IG provides the ability to dynamically calculate via inferences. Here, we give an example to illustrate authorization inference. Table 1 gives the objects defining authorization for a bookstore. Individual users are Bob, Jim, Julia, Peter and Sam. There are relations of authority classes, objects and a relevant relation has Users. The authorities relation of object O is
O.authorities+@=@ReadPermission[ValidAdults]
O.authorities+@=@WritePermission[Manager]

Table 1. Objects of the authorization example

Meta-Class	Object Name	.hasUsers (with inverse "inGroup")
ExplicitGroup	AllMembers	@User[Bob;Jim;Julia;Sam]
ExplicitGroup	OverdueMembers	@User[Bob;Peter]
ExplicitGroup	Manager	@User[Jim]

Meta-Class	Object Name	#implicitQuery
ImplicitGroup	AdultMember	@ExplicitGroup[AllMembers].hasUsers{#age>18}

Meta-Class	Object Name	.bannedGroups
BannedGroup	Overdue	@ExplicitGroup[OverdueMembers]

Meta-Class	Object Name	.forGroups
ReadPermission	ValidAdults	@ImplicitGroup[AdultMember] ; @BannedGroup[Overdue]
WritePermission	Manager	@ ExplicitGroup[Manager]

By those definitions, authorization inference calculates the permission of O. Jim, Julia, and Sam, who are all over age 18, can be granted read

permission. The calculator bans the users in set Overdue Members due to Banned Group [Overdue]. Thus, Bob is filtered out even though he matches the rule of Implicit Group [Adult Member]. Then, only Jim has write permission.

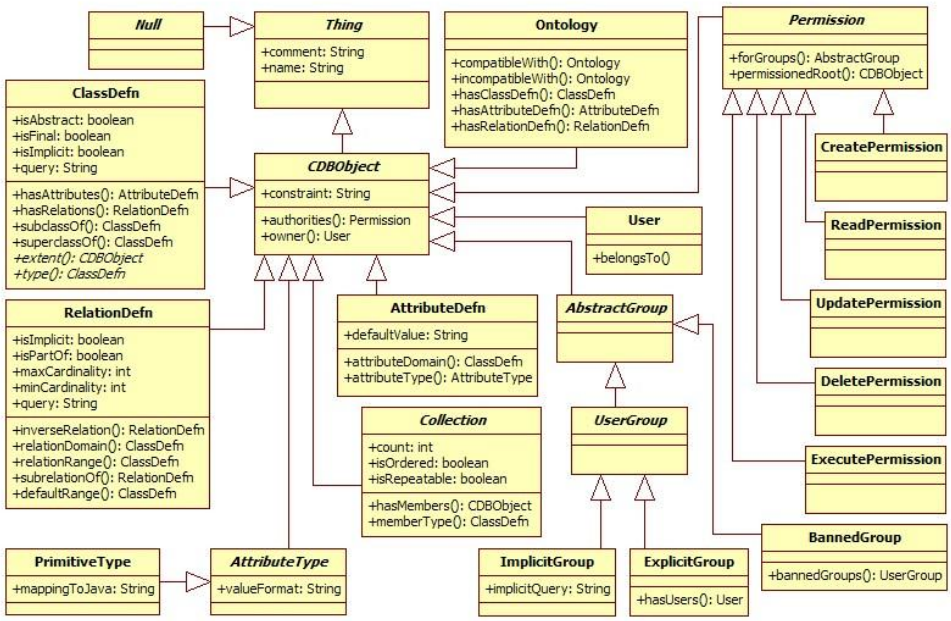

Figure 6. Whole view of Kernel-1

5.3 Access Control

Much recent research on access control in shared ontologies has resulted in proposals for policies and use of ontology resources, representing the access control mechanism for special purposes or domains. UCUP[12] is based on its user-centric user profiles where everything is asserted via user-defined policies and a Privacy Enhanced (PaPE) mechanism. It uses the Semantic Web Rule Language (SWRL[13]) to define rules in Horn clauses, but defining such rules might be a bit complex.

In the paper[14], the attributes of resources are used as the basis of authorization for access control on the grid. All services in the grid must enforce those policies on users. However, in our paper, instead of giving a copious ontology for most uses, only the most essential definitions for authorization are given in Kernel1. With our query language, users not only can query among ontologies but can also easily define implicit permission rules. Users do not need rich programming backgrounds. Knowledge base administrators can also define the permission rules of those objects owned

by themselves as administrators. Kernel1 has only the essential definitions extended from Kernel0, and Kernel1 can be further customized by extensions by authorized developers. The authorization inference follows the subclasses of *Permission* and its relations to calculate the permissions according to the requester and the requested resources. Thus, each ontology can have its own authority mechanism by extending Kernel1.

6. CONCLUSIONS

For sharing ontologies on the Semantic Web, we propose a framework with the capabilities of authorization inference, distributed queries, and cross-referencing. Our upper ontology, Kernel0 provides the essential definitions of a meta-ontology such that everyone can create his or her own ontology model or knowledge by easily extending it. Kernel0 is the main key to enabling cross-referencing of objects on the Web. The intuitive formal query can access classes and objects across the whole Web. Phased queries composed of sub-queries make use of the proposed framework to query objects in other ontologies via a proxy query. Moreover, implicit relations have the ability to relate objects much more flexibly, so that the objects of the range can be dynamically calculated instead of being explicitly enumerated. This feature also enables authorization inference in Kernel1. Kernel1, appended with authority metadata, is a variation of RBAC that achieves authorization inference via implicit relations. Permission of requesting users on requested objects is computed according to fundamental permissions and three basic user-controlling classes, including *ExplicitGroup, ImplicitGroup* and *BannedGroup*. Such implicit relations and authority inference give a flexible way of defining policies.

7. REFERENCES

[1] W3C, *Resource description framework (RDF) model and syntax specification*. Retrieved on April 14, 2012, from http://www.w3.org/TR/PR-rdf-syntax.

[2] W3C, *Resource description framework (RDF) schemas*. Retrieved on April 14, 2012, from http://www.w3.org/TR/rdf-schema.

[3] T.R. Gruber, Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies, Academic Press*, 43(5-6), p907-928, 1995. http://dx.doi.org/10.1006/ijhc.1995.1081.

[4] T. Berners-Lee, J. Hendler, and O. Lassila, The semantic web. *Scientific American*, 284(5), p34-43, 2001. http://dx.doi.org/10.1038%2Fscientificamerican0501-34.

[5] D.J. Buehrer, and L.R. Chien, Knowledge creation using class algebra. In C. Zong (Ed.), *Proceedings of IEEE 2003 International Conference*

on Natural Language Processing and Knowledge Engineering (p108-113). Beijing, China: IEEE Press, 2003. http://dx.doi.org/10.1109/NLPKE.2003.1275877.

[6] D.J. Buehrer, and T.Y. Wang, The cadabia persistent storage service. In L.J. Hyung (Ed.), *Proceedings of New Trends in Information and Service Science* (p197-202). Beijing, China: IEEE Press, 2009. http://dx.doi.org/10.1109/NISS.2009.134.

[7] D. Recordon, and D. Reed, OpenID 2.0: A platform for user-centric identity management. In A. Juels, M. Winslett, and A. Goto (Eds.), *Proceedings of the second ACM workshop on Digital identity management* (p11-16). New York, NY, USA: ACM, 2006. http://dx.doi.org/10.1145/1179529.1179532.

[8] C.M. Liu, The authority mechanism and query parsing for the cadabia middle layer. *A thesis submitted to Institute of Computer Science and Information Engineering, National Chung Cheng University, Taiwan*, 2011.

[9] D. Heimbigner, and D. McLeod, A federated architecture for information management. *ACM Transactions on Information Systems, ACM New York*, 3(3), p253-278, 1985. http://dx.doi.org/10.1145/4229.4233.

[10] Apache Hadoop, Retrieved on April 3, 2012, from http://hadoop.apache.org.

[11] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, Role-based access control models. *Computer, IEEE Press*, 29(2), p38-47, 1996. http://dx.doi.org/10.1109/2.485845.

[12] Z. Iqbal, J. Noll, S. Alam, and M.M.R. Chowdhury, Toward user-centric privacy-aware user profile ontology for future services. In J. Del Ser Lorente (Ed.), *Proceedings of Third International Conference on Communication Theory, Reliability, and Quality of Service* (p249-254). Athens, TBD, Greece: IEEE Press, 2010. http://dx.doi.org/10.1109/CTRQ.2010.49.

[13] Semantic Web Rule Language, Retrieved on April 21, 2012, from http://www.w3.org/Submission/SWRL/.

[14] I. Blanquer, V. Hern'andez, D. Segrelles, and E. Torres, Enhancing privacy and authorization control scalability in the grid through ontologies. *IEEE Transactions on Information Technology in Biomedicine*, 13(1), p16-24, 2009. http://dx.doi.org/10.1109/TITB.2008.2003369.