Semantics Supported Access Authorization Based on Decentralized Architecture

Mohammad M. R. Chowdhury, Josef Noll and Najeeb Elahi

UNIK-University Graduate Center, Post Box 70, 2027 Kjeller, Norway
Email: {mohammad; josef; najeeb}@unik.no

Abstract: Access authorization is one of the ways to provide information security and privacy assurance. This paper proposes an access authorization mechanism in organizations based on decentralized architecture. It includes decentralized organizational structures containing various attributes which are formally represented using the Web Ontology Language. Access authorization decisions are derived through semantic rules and queries. The proposed architecture is compared with its centralized counterpart and with the relational database approach from the computational complexity, management and maintenance point of views.

Keywords: access authorization, decentralized architecture, ontology, rule, semantic.

1. Introduction

An enterprise contains many business sensitive information which should not be leaked to unauthorized employees. Access to these information is based on various attributes of the employees like, roles, the department or project where one plays the roles. All the roles don't enjoy the same access rights or privileges. There are situations where a person holds multiple roles and privileges. With all these complexities in mind, managing access to proper resources through right privileges is not a trivial job. Following are some of the examples of organizational architecture and restricted access scenarios,

- Supervisor is the head of a department. Department owns some resources (administrative resources, documents, deliverables). He can read and write the documents. He can edit its administrative resources and give final approval to the deliverables.
- Department's employees will have only read and write privileges to its documents.
- Departments participate in different projects. Project leader leads a project. He can read and write project's documents. He can edit its administrative resources and give final approval to the project deliverables.
- Besides leader, projects have members. They can only read and write project's documents.

In [4], Chowdhury introduced an access authorization mechanism to meet these requirements using a centralized architecture. This work proposes a decentralized architecture providing the similar access authorization services aiming to further enhance business privacy. Each department and project maintain their separate structure and attribute sets exploiting semantic technologies. Conceptual organizational semantics are formally represented in an ontology using Web Ontology Language. Access authorization decisions are achieved using semantic rules and queries. Design and maintenance of authorization constraints in organizations are challenging problems as company structure, roles, user pools, security requirements are always changing. This paper evaluates the proposed architecture and compared with its centralized counterpart and relational database approach.

The paper is organized as follows. Section 2 and 3 presents the use case scenario and the main features of the architecture. Section 4 briefly illustrates the generic architecture and section 5 describes it in details. The next section provides the access authorization results. In section 7, the proposed architecture will be evaluated and we conclude the paper in the following section summarizing the proposed solution and briefly describing the future works.

2. Use Case Scenario

Fig. 1 illustrates an organizational environment which contains departments and projects. Department A and B both involve in project release 7 and 8. Release 9 is managed by the department A. The departments and projects have their own administration and resource repositories. Each department and project independently assigns roles to its employees with differential access rights or privileges. It deals with the roles like department employee (Emp), department supervisor (Sup), project leader (PL) and project member (PM). Access to resources depends on both the roles and the department or project one plays the role. The scenario requires support of holding multiple roles by a person across different departments or projects. But within a department or project one plays only a single role. Here, we assume that employees are already authenticated through some secure means. All the access scenarios of our use case are described in table 1 considering the roles and the departments/projects one plays the roles.

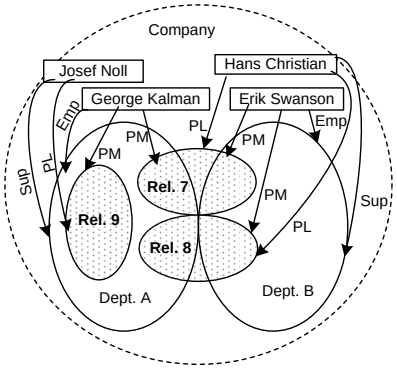

Fig. 1. The use case scenario.

Table 1. The employees, their privileges and access to resources.

Employee	Privilege	Access to Resources
Josef Noll	Administrator	Admin. Resource Dept.A Admin. Resource Rel9
	Final Approval	Deliverables Dept.A Deliverables Rel9
	Read Write	Documents Dept.A Document Rel9 Documents Rel7
Hans Christian	Administrator	Admin. Resource Dept.B Admin. Resource Rel7&Rel8
	Final Approval	Deliverables Dept.B Deliverables Rel7 &Rel8
	Read Write	Documents Dept.B Documents Rel7 &Rel8
George Kalman	Read Write	Documents Dept. A Documents Rel8 Documents Rel9
Erik Swanson	Read Write	Documents Dept. B Documents Rel8

3. Main Features

Decentralized architecture and the semantic representations of the architecture are the main features of the proposed solution.

3.1 Decentralized Architecture

Instead of maintaining a centralized architecture containing the structures of an organization and the attributes of its employees, we propose a decentralized approach. Each department and project will maintain their employee identities, attributes, and resources at own secured places. The attributes contain the roles of each employee, the rights or privileges of each role, and which department or project one plays his role. Semantic technologies enable the access authorization to resources based on these decentralized attributes.

3.2 Using Semantic Technology

3.2.1 Representation of organizational structure

Semantic Web [2] provides various technologies to formally represent organizational structures and attributes at different levels of abstraction and support the reasoning about both the structures and the properties of the elements that constitute the system. Ontologies [8] are the cornerstone technology of Semantic Web. Among the different ontology languages, the Web Ontology Language (OWL) [18] is chosen because it facilitates greater machine interpretability of the Web content than that supported by XML, RDF, and RDFS by providing additional vocabularies along with formal semantics. There are three species of OWL: OWL Lite, OWL DL and OWL Full and these are designed to be layered according to their increasing expressiveness.

3.2.2 Access authorization through rules

Apart from the representation of domain knowledge, the architecture requires more expressivity to deduce decidable conclusions which in fact provide the access authorization decisions. To enhance the expressivity of the ontology, we decided to use OWL DL which is based on the Description Logics (DL) and amenable to automated reasoning. Though OWL DL lacks in expressivity power compared with OWL Full, it maintains decidability[1] and computational efficiency. The computational efficiency is important since the scheme has to handle many relations.

As the expressivity provided by the OWL is limited by tree like structures [14], the implicit relations representing the restricted access scenarios cannot be inferred from the indirect relations between the entities. These require rule support and interworking with ontologies. One suitable rule language is the Semantic Web Rule Language (SWRL) [13]. Along with SWRL, we also use Semantic Query-Enhanced Web Rule Language (SQWRL[2]) to further enhance the expressivity of OWL.

4. Generic Architecture

Fig. 2 briefly illustrates the proposed access authorization architecture. It contains the Semantic Access Authorization Engine and distributed ontologies. Authorization Engine consists of the rule editor, rule database and rule engine. Rule editor is used to design the SWRL rules, company ontology itself serves as the rule database and Jess rule engine executes the SWRL rules and SQWRL queries. Decentralization of organizational structures and attributes is achieved though distributed ontologies which contain separate ontologies of each departments and projects. The company ontology represents the company as a whole. As the rules are integrated into it, the required classes, instances and properties of all the other ontologies are mapped to it. External interface (which is beyond the focus of this work) makes the access authorization queries and retrieves the results of these for resource deliveries to the clients (here the employees of the company).

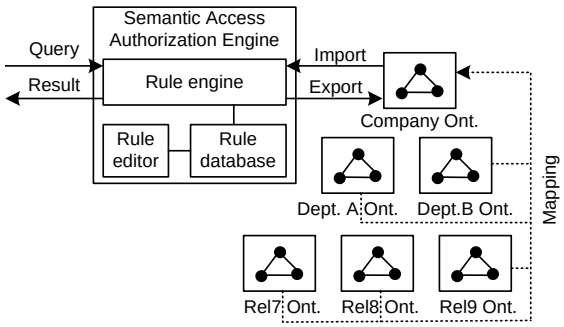

Fig. 2. The generic architecture of the proposed solution.

5. Description of the Architecture

The architecture contains the distributed ontologies and rules. The ontologies represent the decentralized architecture of the organizational structures and attributes. Access authorization decisions provide the resources access with right privileges. These are derived through the rules.

[1] Logics are decidable if computations/algorithms based on the logic will terminate in a finite time.

[2] Semantic Query-Enhanced Web Rule Language (SQWRL), http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL

5.1 Ontology Representing Organizational Structure

In the proposed architecture, each department and project have their own ontologies. Elements of the ontologies are mapped to the company ontology which facilitate the realization of a common access authorization rule. The ontologies are created using $Protégé$ ontology editor (version 3.4 beta).

Preliminaries. An ontology is a set of *classes* C, *properties* P and *instances* i. In this work, concepts have three types of relation among them: *subClassOf*, *disjointWith*, and *equivalentClass*. The semantic scope (SC) of a concept (class) C_i is represented as $(SC\,(C_i))$. The definition of these three types of relations are,

- *subClassOf*: $SC\,(C_1) \subseteq SC\,(C_2)$, the semantic scope of C_1 is narrower than that of C_2.
- *disjointWith*: $SC\,(C_1) \subseteq \neg\ SC\,(C_2)$, the semantic scope of C_1 is disjoint with that of C_2.
- *equivalentClass*: $SC\,(C_1) \equiv SC\,(C_2)$, the semantic scope of C_1 is equivalent with that of C_2.

Here we also define several additional characteristics of OWL which we use here,

- $P\,(i_1, i_2)$ states that i_1 relates with i_2 through the property P.
- *owl:equivalentProperty*: $P_1 \equiv P_2$, it can be used to state that two properties have the same property extension.
- *owl:sameAs*: $i_1 \equiv i_2$, it is used here to state that two instances are in fact same.
- *owl:symmetricProperty*: it states that if P relates $i_1\ \&\ i_2$, then P also relates $i_2\ \&\ i_1$ and can be represented as $P \equiv P^-$, where P^- is the inverse property of P.

Defining concepts through classes. In ontologies, the key concepts of the domain (here access authorization) are defined through *classes*. Each of the ontologies contain the same concepts: *EmployeeID*, *Role*, *Privilege*, *Resource*, *WorkUnit*. In departments and projects, Resource and Work Unit are further subdivided into the following class-subclass relations,

- $\{AdminResource, WebResource\} \subseteq Resource$
- $\{Deliverable, Document\} \subseteq WebResource$
- $\{Department\ or\ Project\} \subseteq WorkUnit$

Here, Web Resources can be accessed through the Web but administration resources are not available to the Web. The company ontology contains the following class-subclass relations,

- $\{$*mapped classes*$\} \subseteq CompanyEmployee \subseteq Role$
- $\{Department, Project\} \subseteq Company \subseteq WorkUnit$
- $\{$*mapped classes*$\} \subseteq \{Department\ or\ Project\}$
- $Project \subseteq \neg Department$, it means no instances of a department can be added as instances of a project.

Realizing concepts through instances. The real actors (or individuals) of a practical use case scenario are defined through the instances and they belong to the classes. The instances of department A and which classes (or subclasses) they belong to (at the right) are as follows,

- $\{GeorgeKalman, JosefNoll\}:\ EmployeeID$
- $\{Admin, FinalApproval, ReadWrite\}:\ Privilege$
- $\{AdminResDeptA\}:\ AdminResource$
- $\{DeliverableDeptA\}:\ Deliverable$
- $\{DocDeptA\}:\ Document$
- $\{DepartmentA\}:\ Department$
- $\{DeptEmployee, Supervisor\}:\ Role$

We follow the use case scenario to add the instances to the remaining department and projects.

Relating instances through properties. In the ontologies, a *property* relates two instances of the classes. Properties have a *domain* and *range*. Syntactically, *domain* links a property to a class and *range* links a property to either a class or a data range [18]. A property relates instances from the *domain* with the instances from the *range*. Table 2 lists the properties used in the ontologies. Here *hasResource* and *belongsTo* properties are defined as *owl:symmetricProperty*. It ensures that when a department or project owns a resource (by *hasResource*), then the its inverse relationship (belongsTo) is also true for the same resource. All the properties are explicitly defined except *hasAccessTo* which provides the access authorization to appropriate resource. These facts are derived through the inference using the rules in Semantic Access Authorization Engine.

Table 2. The list of properties, their domains and ranges.

Property name	Domain	Range
hasRole	EmployeeID	Role
hasResource	WorkUnit	Resource
belongsTo	Resource	WorkUnit
hasPrivilege	Role	Privilege
needPrivilege	Resource	Privilege
rolePlaysIn	Role	WorkUnit
hasAccessTo	Role	Resource

Mapping to support distributed architecture. Decentralized architecture is realized through distributed ontologies of the departments and project which are stored as separate OWL files at the local machine and are imported to the company ontology. To apply the rules and reasoning over the ontologies, we need mediation between the components of the ontologies. Among the three widely used mediation techniques [5], we use ontology mapping which represents the correspondence between the ontologies. A common set of rules can infer the access authorization decisions only when the relevant classes, instances and properties of the department/project ontologies are mapped to the corresponding elements of the company ontology. It is assumed that the administrator of the department and project is responsible for the mapping. Several classes of the department and project ontologies are mapped to that of the company ontology using *subClassOf* relation, these can be represented as follows,

- $\{DeptA\ \&\ B : AdminResource\} \subseteq Resource$
- $\{Rel7\ \&\ 8\ \&\ 9 : AdminResource\} \subseteq Resource$
- $\{DeptA\ \&\ B : WebResource\} \subseteq WebResource$
- $\{Rel7\ \&\ 8\ \&\ 9 : WebResource\} \subseteq WebResource$
- $\{DeptA\ \&\ B : Role\} \subseteq CompanyEmployee$
- $\{Rel7\ \&\ 8\ \&\ 9 : Role\} \subseteq CompanyEmployee$
- $\{DeptA\ \&\ B : Department\} \subseteq Department$
- $\{Rel7\ \&\ 8\ \&\ 9 : Project\} \subseteq Project$

The access authorization decisions also require mutual mapping of several properties between the departments/projects and the company ontologies. We use *owl:equivalentProperty* for mapping of properties. These can be represented as follows,

- $\{DeptA\ \&\ B : hasRole\} \equiv hasRole$
- $\{Rel7\ \&\ 8\ \&\ 9 : hasRole\} \equiv hasRole$
- $hasRole \equiv \{DeptA\ \&\ DeptB\}$
- $hasRole \equiv \{Rel7\ \&\ 8\ \&\ 9\}$

Similarly, *belongsTo*, *hasPrivilege*, *needPrivilege*, *hasAccessTo* and *rolePlaysIn* are also mutually mapped.

There are some common instances within these ontologies. The ontology framework requires declaring them same explicitly using the *owl:sameAs*. Thus, we map instances of *EmployeeID* and *Privilege* from the department/project ontologies to the company ontology. As for example, this can be represented as,

- $DeptA : JosefNoll \equiv JosefNoll$
- $DeptA : Admin \equiv Admin$

Similarly, all the remaining instances of these two classes are mapped.

5.2 Rule Description

The access authorization rule is realized using SWRL. The rules are formulated using the classes, properties and instances of the company ontology. The rules are formulated as follows:

$Role(?R) \wedge rolePlaysIn(?R, ?X) \wedge belongsTo(?Z, ?X) \wedge hasPrivilege(?R, ?Y) \wedge needPrivilege(?Z, ?Y) \wedge \longrightarrow hasAccessTo(?R, ?Z)$

where $R \in Role$; $X \in WorkUnit$; $Y \in Privilege$; $Z \in Resource$.

6. Access Authorization Results

The Jess rule engine executes the rule along with SQWRL queries and thus infers the access authorization decisions. These are the implicit relations derived from the explicit representations in the ontologies. SQWRL queries are included as follows,

$EmployeeID(?ID) \wedge hasRole(?ID, ?R) \wedge Privilege(?PR) \wedge hasPrivilege(?R, ?PR) \wedge needPrivilege(?Z, ?PR) \wedge hasAccessTo(?R, ?Z) \longrightarrow sqwrl : select(?ID) \wedge sqwrl : select(?Z) \wedge sqwrl : select(?PR) \wedge sqwrl : columnNames(\text{“}EmployeeID\text{”}, \text{“}AccesstoResource\text{”}, \text{“}WithPrivilege\text{”}) \wedge sqwrl : orderBy(ID?)$

It takes in the results of the rule through *hasAccessTo* (sec. 5.2) to answer the queries (using *sqwrl:select*). Jess engine first converts the relevant OWL knowledge and SWRL rules to Jess knowledge. Then it computes the implicit facts executing the rules. Fig. 3 presents the access authorization results for each of the employees.

Though the rule is designed using the classes, instances and properties of the company ontology only, due to mapping the results consider the corresponding classes, instances and properties of all the other ontologies.

SQWRLQueryTab | Rule-2

EmployeeID	Access to Resources	With Privilege
Erik_Swansson	ProjectRel8:Doc_Rel8	ReadWrite
Erik_Swansson	DepB:Doc_DeptB	ReadWrite
George_Kalman	ProjectRel9:Doc_Rel9	ReadWrite
George_Kalman	ProjectRel8:Doc_Rel8	ReadWrite
George_Kalman	DeptA:Doc_DeptA	ReadWrite
Hans_Christian	ProjectRel7:AdminResRel7	Admin
Hans_Christian	ProjectRel7:Doc_Rel7	ReadWrite
Hans_Christian	ProjectRel7:Deliverable_Rel7	FinalApproval
Hans_Christian	ProjectRel8:AdminResRel8	Admin
Hans_Christian	ProjectRel8:Doc_Rel8	ReadWrite
Hans_Christian	DepB:Doc_DeptB	ReadWrite
Hans_Christian	DepB:AdminResDeptB	Admin
Hans_Christian	DepB:Deliverable_DeptB	FinalApproval
Hans_Christian	ProjectRel8:Deliverable_Rel8	FinalApproval
Josef_Noll	DeptA:Deliverable_DeptA	FinalApproval
Josef_Noll	ProjectRel7:Doc_Rel7	ReadWrite
Josef_Noll	ProjectRel9:Deliverable_Rel9	FinalApproval
Josef_Noll	ProjectRel9:Doc_Rel9	ReadWrite
Josef_Noll	ProjectRel9:AdminResRel9	Admin
Josef_Noll	DeptA:AdminResDeptA	Admin
Josef_Noll	DeptA:Doc_DeptA	ReadWrite

Fig. 3. Access authorization results executing SWRL rule and SQWRL queries.

7. Evaluation

In this section, we compare the proposed decentralized architecture with a centralized one from the various aspects. We also evaluate the usefulness of the proposed semantics approach.

7.1 Centralized Architecture

In [4], the same use case scenario (section 2) was maintained centrally. The proposed architecture is required to support multiple roles of a person. It was addressed in [4] by deviating the class-subclass relationship of *Role* as follows, {DeptEmployee, ProjectLeader, ProjectMember,Supervisor} $\subseteq$ CompanyEmployee $\subseteq$ Role. Each of the subclasses contained instances representing an employee's specific role in a department or project. As for example, the instances of *Supervisor* was represented as, {SupHans, SupJosef}: Supervisor. *SupHans* represented the supervisor instance of Hans Christian that he plays in Department B. Excluding this deviation, the remaining class-subclasses, properties and instances are similar to those of the decentralized architecture and obviously mapping of the elements was not required there.

7.2 Reasoning Process

We include the reasoning process to compute the time required for deducing the same implicit relations. The rule and queries defined before results the access authorization decisions for each employees. In this section, we measure the time required to compute these through reasoning. The computation times are used to compare the architectures. All the implementations and the measurements are done in a desktop PC based on Windows XP with a P4 2.0 GHz processor and 1 GB RAM. As we import the ontologies from the local files, there is no

network delays. In calculation, we also exclude the loading times of ontologies to $Protégé$.

The inferred instances which will answer, 'which employee can access which resources'. In this regard, we add a class *AccessToResource* which can have values only from the instances of the *Resource* using *hasValue* property. The restrictions are represented as, $hasValue \ni Resource(where, \ni = someValuesFrom)$.

We add new subclasses of it each representing 'access to resources by each employee'. Each subclass carries restrictions which allow it to have appropriate resource instances. As for example, *AccessToResourceByJosef* ('access to resources by Josef Noll') is restricted by the department/project Josef belongs to and the privileges needed by the inferred resources. These are formulated as follows,

$(belongsTo \ni DeptA : DepartmentA) \sqcup$
$(belongsTo \ni ProjectRel7 : Rel7) \sqcup (belongsTo \ni ProjectRel9 : Rel9);$
$(needPrivilege \ni Admin) \sqcup (needPrivilege \ni FinalApproval) \sqcup$
$(needPrivilege \ni ReadWrite);$
$(accessedBy \ni Josef\_Noll)\,; where, \ni = hasValue$

Thus, we add the remaining subclasses and the corresponding restrictions. we use Pellet (version 1.5.1) reasoning service to compute the inferred instances of them. It is connected with the $Protégé$ platform through an interface (DIG interface[3]). During the reasoning, Pellet also provides the computation time. Fig. 4 presents a snapshot of the inferred instances of these subclasses. It shows the resources accessed by Josef Noll. These facts have not been defined explicitly and are the results of reasoning.

Fig. 5 illustrates the comparisons of the time to compute the inferred instances between the centralized and decentralized architecture for the same set of restricted subclasses of *AccessToResource*. For each independent run, the reasoner needs additional time to synchronize and update. So, we provide two different comparisons: one with the synchronization and update time, the other without these (bold lines). Pellet reasoner caches the results of the previously asked queries, so for the former each computation time is derived from the average of ten independent run of the reasoner. We collected these data for three different numbers of resources, first for the default number of resource (defined in section 5), and subsequently for the three and six times of it. The required ontology elements are

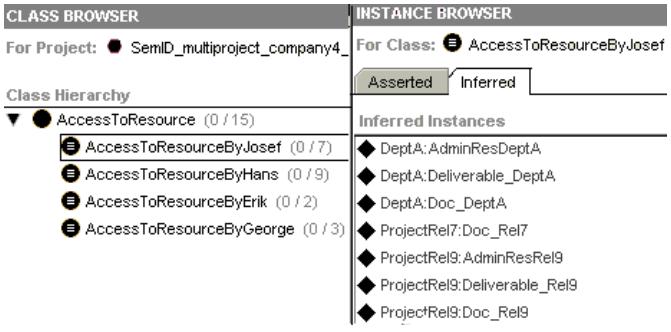

Fig. 4. Inferred instances when the reasoner is run.

exported from the $Protégé$ to the reasoner for the inferencing.

[3] Using $Protégé$-OWL reasoner API, http://protege.stanford.edu/plugins/owl/api/ReasonerAPIExamples.html

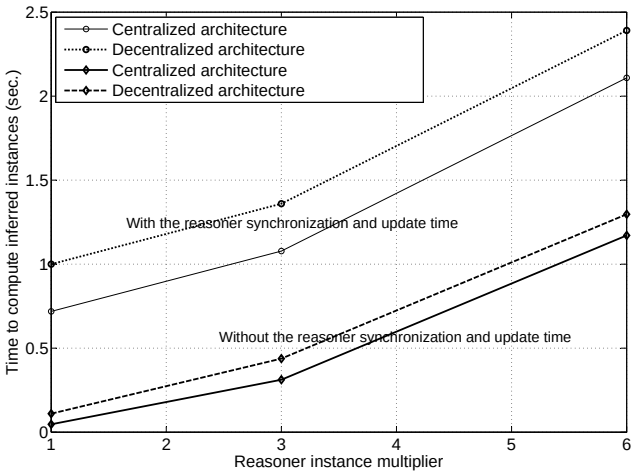

Fig. 5. Time to compute inferred instances (centralized vs decentralized architecture).

Fig. 6 shows the comparisons between the number of ontology elements exports from the centralized and decentralized architecture. Here the exported OWL axiom comprises of only the

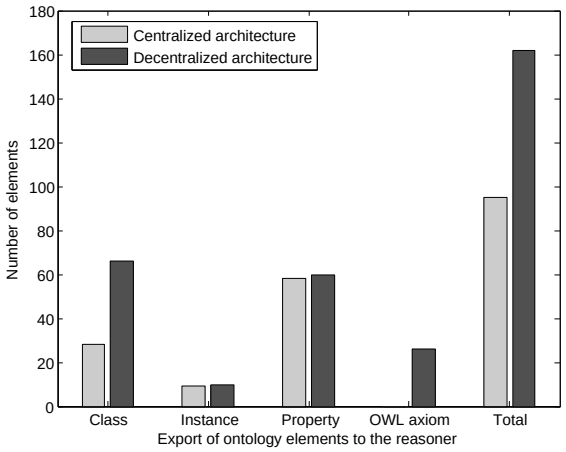

Fig. 6. Number of ontology element exports to the reasoner (centralized vs decentralized architecture).

relationships through *owl:sameAs*.

7.3 Discussion

For a large scale organization, the centralized architecture cannot provide scalability from management and maintenance point of views. This can also introduce privacy threats. To mitigate these problems, this paper proposes the decentralized architecture where access authorization is achieved based on the decentralized attributes. But it also possesses scalability concerns from the aspects of computational complexity during the inference process. Fig. 5 clearly shows that decentralized architecture always takes more time to compute inferred instances than its centralized counterpart. The trend is consistent even when we add new sets of resource instances (three and six times of the default resource instances). The results obtained without the synchronization and update time of the reasoner, maintains the similar tendency (bold lines in fig. 5). It should

be noted that Pellet may not be the fastest OWL reasoner [10], [15], but we used it only to compare the two architectures.

Fig. 6 reveals the reasons behind this phenomena. In the decentralized architecture, the identical set of classes and properties are reused in every department and project ontologies. The number of instances of the classes is equal in both of the architectures. But when the required classes and properties are mapped to the company ontology, the total number of classes exported to the reasoner multiplied. The number of instances and properties exported to the reasoner from both the architectures is also more or less the same. As decentralized architecture requires mapping through *owl:sameAs*, these relations also need to be exported. All these cause a consistent delay in the reasoning for the decentralized architecture.

In centralized architecture to support multiple role of a person, each new employee needs addition of a new role instance which arises the scalability question. In this respect, the design of its counterpart proves its superiority. Currently such access authorization solutions are provided through database management systems (DBMS). Though DBMS is computationally efficient, these systems (DBMS-only) do not support reasoning process [16]. The reasoning capabilities of semantic technologies facilitate computing complex implicit relations from the explicit specifications.

The proposed architecture provides a simple mechanism to modify access rights. One can simply add or delete the relationship between the employee and the role instance to assign or repeal a role. In a similar way, relationship between the role and privilege instances can be modified. Executing the rules or running the reasoner will deduce the new access authorization decisions. Among the few disadvantages, the proposed architecture is computationally expensive and decidability not always guaranteed by SWRL. In addition, XML processing is slower than that of database.

7.4 Related Works

The concept of role plays a significant part in the proposed architecture. Role Based Access Control (RBAC)[17] is an increasingly popular and efficient solution where users' access permissions are associated with the roles, and users are made members of appropriate roles. We consider the concept of RBAC as a part of our access authorization mechanism. There are two types of RBAC constraints: dynamic and static [7]. Though we focus on static constraints on the roles, the integration of semantics can adapt architecture to ever changing organizational structure with minimum effort. Semantic representation of RBAC has been studied in [1], [3], [6], [9], [11], [12]. All these attempts focused on centralized architecture of storing role representations. In this paper, we introduce a decentralized architecture of representing and storing roles. Besides incorporating the notion of roles, the proposed architecture includes the attribute 'where (department or project) one plays the roles' and the possession of multiple roles as the constraints to derive the access authorization decisions. Therefore, a simple notion of Attribute Based Access Control (ABAC) [19] has also been integrated in this architecture. Though the assignment of roles to employees are hard coded in this solution, roles can be designated dynamically based on various other attributes and context information. But this is beyond the focus of this paper.

8. Conclusion and Future Work

The proposed solution provides access authorization decisions based on decentralized architecture. It comprises of distributed ontologies containing organizational structures and attributes. The authorization decisions are derived employing semantic rules and queries. The novelty of the solution is, we use the benefits of semantic technologies and include the decentralized architecture and the possibilities of playing multiple roles by a person across the different departments or projects. Though the proposed solution provides advantages in scalability from the management and maintenance point of views, it is computationally expensive. Besides, we do all the mapping manually which should be automated in practice.

We plan to integrate the authentication mechanism, the access rights delegation and dynamic mapping features with this solution. Based on these, we are planning to build a standalone enterprise access control system.

References

[1] M A Al-Kahtani and R Sandhu, Rule-based RBAC with negative authorization. Proceedings of the 20th Annual Computer Security Applications Conference Tucson, AZ, 2004, pp. 405–415.

[2] T Berners-Lee, J Hendler and O Lassila, The semantic web. Scientific American Magazine, 2001.

[3] E A Cho, C J Moon, D H Park and D K Baik, An effective policy management framework using RBAC model for service platform based on components. Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications (SERA06) Seattle, Washington, 2006.

[4] M M R Chowdhury and J Noll, Capturing semantics for information security and privacy assurance. to appear in the Proceedings of 5th International Conference on Ubiquitous Intelligence and Computing (UIC-08) Oslo, 2008, pp. 187–200.

[5] J de Bruijn, M Ehrig, C Feier, F Martns-Recuerda, F Scharffe and M Weiten, Ontology mediation, merging, and aligning, In Semantic Web Technologies, J Davies, P Warren, and R Studer, Eds. John Wiley and Sons, 2006.

[6] W Di, L Jian, D Yabo and Z Miaoliang, Using semantic web technologies to specify constraints of RBAC. Proceedings of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT05) Dalian, 2005.

[7] A Dury, S Boroday, A Petrenko and V Lotz, Formal verification of business workflows and role based access control systems. Proceedings of International Conference on Emerging Security, Information Systems and Technologies Valencia, 2007, pp. 201–210.

[8] D Fensel, Ontologies: A silver bullet for knowledge management and electronic commerce (2nd ed.). Springer-Verlag, 2003.

[9] T Finin, A Joshi, L Kagal, J Niu, R Sandhu, W H Winsborough and B Thuraisingham, Role based access control and OWL. Proceedings of the Fourth OWL: Experiences and Directions Workshop 2008.

[10] T Gardiner, D Tsarkov and I Horrocks, Framework for an automated comparison of description logic reasoners. Lecture Notes in Computer Science, 2006, pp. 654–667.

[11] N Heilili, Y Chen, C Zhao, Z Luo and Z Lin, An OWL-based approach for RBAC with negative authorization, In Knowledge Science, Engineering and Management, J Lang, F Lin, and J Wang, Eds. Springer, 2006.

[12] S H Hong, E A Cho, C J Moon and D K Baik, RBAC-based

access control framework for ensuring privacy in ubiquitous computing. Proceedings of the International Conference on Hybrid Information Technology (ICHIT'06) Cheju Island, 2006.
[13] I Horrocks, P F Patel-Schneider, H Boley, S Tabet, B Grosof and M Dean, SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 2004.
[14] B Motik, U Sattler and R Studer, Query answering for OWL-DL with rules. Proceedings of International Semantic Web Conference 2004 Hiroshima, 2004, pp. 549–563.
[15] Z Pan, Benchmarking DL reasoners using realistic ontologies. Proceedings of the International workshop on OWL: Experience and Directions (OWL-ED2005) Galway, 2005.
[16] M J Park, J Lee, C H Lee, J Lin, O Serres, and C W Chung, An efficient and scalable management of ontology. Lecture Notes in Computer Science, 2007, pp. 975–980.
[17] R S Sandhu, E J Coyne, H L Feinstein and C E Youman, Role-based access control models. IEEE Computer, No. 2, 1996, pp. 38–47.
[18] M K Smith, C Welty and D L McGuinness, OWL web ontology language guide. W3C Recommendation, 2004.
[19] E Yuan and J Tong, Attributed based access control (ABAC) for web services. Proceedings of the IEEE International Conference on Web Services (ICWS05) Orlando, Florida, 2005, pp. 561–569.