PAPER

RDFacl: A Secure Access Control Model Based on RDF Triple

Jaehoon KIM[†a)], ***Member and*** **Seog PARK**[†b)], ***Nonmember***

SUMMARY An expectation for more intelligent Web is recently being reflected through the new research field called Semantic Web. In this paper, related with Semantic Web security, we introduce an RDF triple based access control model having explicit authorization propagation by inheritance and implicit authorization propagation by inference. Especially, we explain an authorization conflict problem between the explicit and the implicit authorization propagation, which is an important concept in access control for Semantic Web. We also propose a novel conflict detection algorithm using graph labeling techniques in order to efficiently find authorization conflicts. Some experimental results show that the proposed detection algorithm has much better performance than the existing detection algorithm when data size and number of specified authorizations become larger.

key words: *database security, access control, authorization conflict, RDF/OWL data, Semantic Web*

1. Introduction

Recently, we can find some efforts to secure Semantic Web (SW). Qin and Atluri [1] introduced a class level access control considering authorization propagation by various ontology inferences, and Reddivari et al. [2] introduced an Resource Description Framework (RDF) triple based access control model considering various operations in an RDF store. Jain and Farkas [3] introduced an RDF triple based access control considering an authorization conflict problem in RDF inference.

In this paper, we also introduce an access control model based on the RDF triple, which considers SW inference. However, compared with the existing studies, our model is based on explicit authorization propagation over the ontology hierarchy of upper and lower classes or properties. The explicit propagation of authorizations is that when an authorization is specified for an upper class or property, the same access authorization is also applied to all lower classes or properties over the ontology hierarchy by inheritance. This explicit authorization propagation over the ontology hierarchy, which is a graph, is necessary for more convenient authorization specification of a security administrator as in the eXtensible Markup Language (XML) access control [4]–[7] exploiting the authorization propagation over a tree. Through the explicit authorization propagation, a variety of authorization specifications can be done at one time without the need to be done separately and a security administrator can manage a much lower number of authorizations. Our contribution is, first, to introduce an RDF triple based access control model (named RDFacl) supporting the explicit authorization propagation by inheritance as well as the implicit authorization propagation by inference. Next, using these two contrary propagations, we introduce an authorization conflict problem in SW access control. As for the ontology hierarchy, in this paper, we consider the basic subsumption relationships in RDF, $subClassOf$ and $subPropertyOf$, and further consider the more complex subsumption relationships in OWL like $unionOf$, $intersectionOf$, and $oneOf$.

With regard to detecting the authorization conflict, Jain and Farkas [3] have suggested a somewhat inefficient algorithm where all RDF triples are inspected. Therefore, in this paper, we also propose an efficient conflict detection method which inspects only the formerly specified authorizations rather than all RDF instances. To efficiently detect an authorization conflict under subsumption inferences, our method inspects only the former authorizations having subsumption relationships with a new authorization.

The remainder of the paper is organized as follows. In Sect. 2, we review recent studies related to SW access control. Next, Sect. 3 briefly explains about the RDF triple and the subsumption relationship, and through Sects. 4, 5, and 6 we introduce the RDFacl. Section 7 introduces the suggested authorization conflict detection method, and Sect. 8 presents some experimental results. Section 9 finally concludes this paper.

2. Related Work

XACML [4], Damiani et al. [5] and E. Bertino et al. [6], [7] introduced the fine-grained access control models for XML documents. According to specified access authorizations, element tags and attributes in an XML tree structure are made to be selectively invisible to users. Even though RDF/OWL documents are described in XML, the existing XML access control models are not desirable as they cannot check the security violation by ontology inference. That is, when a set of access authorizations are explicitly specified for RDF and OWL tags, they do not define which authorizations should be applied to inferred information.

Jain and Farkas' study [3] is the closest work to ours. However, in our study, we exploit the explicit authorization propagation and suggest a more efficient authorization conflict detection method.

Qin and Atluri [1] considers the implicit authorization

Manuscript received November 8, 2007.

Manuscript revised August 19, 2008.

[†]The authors are with the Dept. of Computer Science, Sogang University, 1–1 Shinsu-Dong Mapo-Gu, Seoul, 121–742, Korea.

a) E-mail: jhkimygk@gmail.com

b) E-mail: spark@dblab.sogang.ac.kr

DOI: 10.1587/transinf.E92.D.41

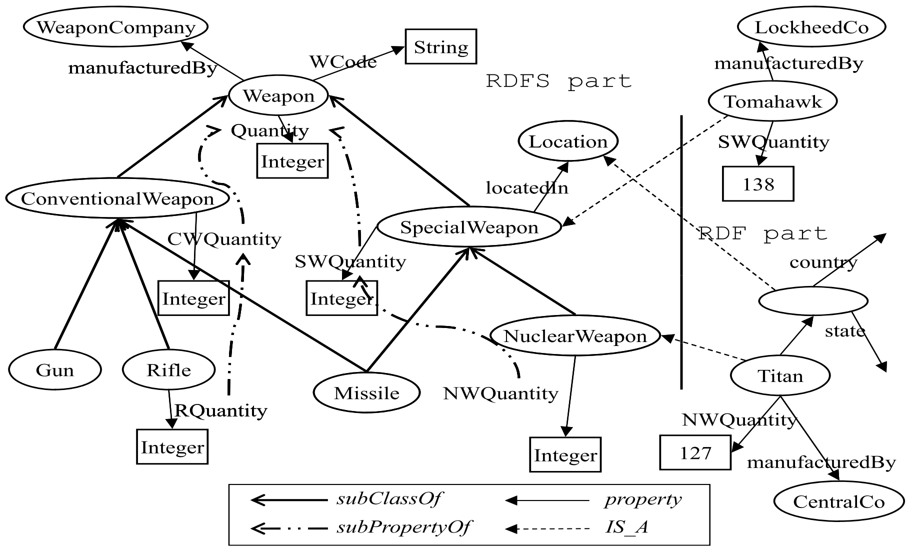


Fig. 1 A sample RDF graph.

propagation and authorization conflict problem for various semantic relations in an ontology. However, their access control policy is not based on the RDF triple structure. Hence, their methods are not incorporated with RDF Semantics [8] and OWL Semantics [9]. In addition, the security object in their model is a class, but in our case it is a property which is represented as the RDF triple. Hence we can support the more fine-grained access control.

Kaushik et al. [10] introduces an access control model for the fine-grained information disclosure of an RDF web document. The main point of their study is to introduce a formal framework to provide disclosure control over parts of an ontology. In addition, they introduce applying several methods of information hiding to RDF data, e.g., removing a specific subtree in an ontology tree or renaming a disallowed class or property according to an authorization. However they do not consider the disclosure problem for highly sensitive data by a prohibited inference. In fact, this problem is closely connected with the authorization conflict problem in this paper, because such an information disclosure arises when two authorizations having conflict relationship are both allowed.

In the paper [11], we already introduced the explicit and the implicit authorization propagation, the authorization conflict problem between two propagations, and the conflict detection algorithm. In this paper, we revise some contents in the previous article, add some experimental results which were not shown in the previous article, and particularly we expand such concepts into some subsumption relationships of OWL.

3. RDF and OWL

An RDF document consists of RDF statements, which use ontology concepts defined in an RDF Schema (RDFS). An RDF statement is represented as a triple of [*s*, *p*, *o*], and an RDF document can be represented as a graph consisting of RDF triples. An RDFS statement can also be represented as a triple.

Definition 1 (RDF graph and triple): An RDF graph is a set of RDF triples. An RDF triple is represented as [*s*, *p*, *o*], where $s \in SUBJECT$, $p \in PREDICATE$, and $o \in OBJECT$.

- The set $SUBJECT$ includes URI (Uniform Resource Identifier) nodes defining classes or properties in RDFS and instances in RDF, and blank nodes.

- The set $PREDICATE$ includes URI nodes referencing properties in RDFS.

- The set $OBJECT$ includes the URI nodes of the other classes and instances related by *p*, blank nodes, and literals.

For example, in Fig. 1, the RDF triple [*Weapon*, $manufacturedBy$, *WeaponCompany*] has the class URI constant *Weapon* as *s*, the property URI constant $manufacturedBy$ as *p*, and the class URI constant *WeaponCompany* as *o*. [$Titan$, *NWQuantity*, 127] has the instance URI constant $Titan$ as *s* and the literal 127 as *o*. [$Titan$, *locatedIn*, _] has a blank node as *o*. The RDF part of Fig. 1 represents the sample RDF document of Fig. 2.

OWL is also based on the RDF triple and adds separate vocabularies for defining various relationships between ontology concepts to the primitive vocabularies of RDF. For example, the RDF graph of Fig. 3 (b) connects the following RDF triples for the OWL document of Fig. 3 (a). Refer Sect. 4.1 in the OWL Semantics [9] for the details of mapping OWL to RDF graph.

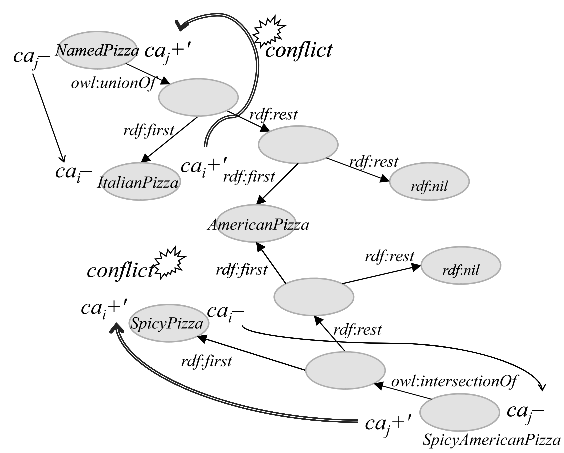
[*NamedPizza*, rdf:*type*, *owl*:*Class*]
[*NamedPizza*, *owl*:$unionOf$, _:1]
[_:1, rdf:*type*, rdf:*List*]
[_:1, rdf:$first$, *ItalianPizza*]
[_:1, rdf:$rest$, _:2]
[_:2, rdf:$first$, *AmericanPizza*]

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ex="http://example.org/schemas/weapon#" xmlns:xsd =
"http://http://www.w3.org/2001/XMLSchema">
  <ex:NuclearWeapon rdf:ID="Titan">
    <ex:manufacturedBy rdf:resource="ex:CentralCo"/>
    <ex:NWQuantity rdf:datatype="xsd:integer">127
                                          </ex:NWQuantity>
    <ex:locatdIn>
      <rdf:Description>
        <ex:country>USA</ex:country>
        <ex:state>Arizona</ex:state>
      </rdf:Description>
    </ex:locatdIn>
  </ex:NuclearWeapon>
  <ex:SpecialWeapon rdf:ID="Tomahawk">
    <ex:manufacturedBy rdf:resource="ex:LockheedCo"/>
    <ex:SWQuantity rdf:datatype="xsd:integer">138
                                          </ex:SWQuantity>
  </ex:SpecialWeapon>
</rdf:RDF>
```

Fig. 2 A sample RDF web document.

```
<owl:Class rdf:ID="NamedPizza">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#ItalianPizza"/>
    <owl:Class rdf:about="#AmericanPizza"/>
  </owl:unionOf> </owl:Class>
<owl:Class rdf:ID="SpicyAmericanPizza">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SpicyPizza"/>
    <owl:Class rdf:about="#AmericanPizza"/>
  </owl:intersectionOf> </owl:Class>
```

(a) OWL web document



(b) RDF graph

Fig. 3 An RDF graph for an OWL web document.

[_:2, rdf:$rest$, rdf:nil]

Definition 2 (subsumption relationship): In RDF, if a class c_i is the subclass of another class c_j ($c_i \subset c_j$, $subClassOf$), c_i and its instances inherit the properties of c_j, and c_i can be interpreted as c_j by inference. If a property p_i is the subproperty of another property p_j ($p_i \subset p_j$, $subPropertyOf$), p_i can be interpreted as p_j by inference. Similarly, in OWL, if a class c_j is the union of other classes c_i and c_k (c_j = ($c_i \cup c_k$, $unionOf$), c_i and c_k can be interpreted as c_j by inference. If a class c_j is the intersection of other classes c_i and c_k (c_j = ($c_i \cap c_k$, $intersectionOf$), c_j can be interpreted as c_i or c_k by inference.

4. Access Authorization

4.1 Security Object

In our authorization specification, security objects are RDF triples. A security administrator can conveniently bind up the target RDF triples into the following security object pattern.

Definition 3 (security object pattern): A security object pattern is also represented as an RDF triple [s, p, o], where s and p can be substituted by variables \$$x$ and \$$y$, respectively, and o is always the variable \$$z$ (In this study, we do not consider much more fine-grained access control according to o values). Also, a blank node for s and o is not allowed.

For example, in the RDF graph of Fig. 1, for the security object pattern [\$$x$, $NWQuantity$, \$$z$], the matching RDF triples are [$NuclearWeapon$, $NWQuantity$, $literal$] and [$Titan$, $NWQuantity$, $literal$]. In particular, the pattern [\$$x$, \$$y$, \$$z$] matches all edges in the graph.

4.2 Specifying Access Authorization

Access authorizations are formally defined as follows.

Definition 4 (access authorization): An access authorization is a five tuple of the form: <$subj$, obj, act, $sign$, $type$>.

- $subj$ is the subject to whom the authorization is granted.
- obj is the security object pattern.
- act refers to an action performed against the security object. Since in this study we consider applying our access control model to the fine-grained information disclosure of RDF/OWL documents over Web, only read operation is considered.
- $sign$ is (+) if access is allowed, and (−) if access is forbidden.
- $type$ is R (= Recursive) if an authorization should be propagated to lower classes or properties by the subsumption relationship, and L (= Local) if an authorization should not be propagated. We will explain the details of authorization propagation according to $type$ in Sect. 5.

4.3 Hidden Portions of an RDF/OWL Document according to an Authorization

We consider applying our access control model to information disclosure of RDF/OWL documents published over Web. For example, according to the authorization <Dave, [\$$x$, $manufacturedBy$, \$$z$], read, −, R>, the tag <ex:$manufacturedBy$ rdf:$resource$ = "ex:$CentralCo$"> in

Table 1 Hidden portions according to the value type of *o*.

	URI constant	Blank node	Literal
read (−)	- A property *p* should be hidden from a class or an instance *s*. - Also, the *o* value of an URI constant should be hidden. However, this does not mean that the object referenced by the URI should be hidden. The referencing relationship is only broken.	- *p* should be hidden from *s*. - Also, the *o* value of a blank node should be hidden.	- *p* should be hidden from *s*. - Also, the *o* value of a literal should be hidden.

the sample RDF document of Fig. 2 must be invisible to the subject Dave. In this subsection, we define which portions of an RDF/OWL document must be hidden according to a specified access authorization. The hidden portions are decided by the value type of $o \in$ {class or instance URI constant, blank node, literal} in Definition 1. Table 1 summarizes this.

Example 1: According to <Dave, [\$*x*, *manufacturedBy*, \$*z*], read, −, R>, the p = "*ex*:*manufacturedBy*" and the o = '*rdf*:*resource* = "*ex*:*CentralCo*"' in Fig. 2 are hidden from the instance s = "*Titan*". However, the actual instance *CentralCo* referenced by the instance URI constant "*ex*:*CentralCo*" is not hidden. According to <Dave, [\$*x*, *locatedIn*, \$*z*], read, −, R>, the p = "*ex*:*locatedIn*" and the blank node <*rdf*:*Description*> … </*rdf*:*Description*> are hidden. In the case of <Dave, [\$*x*, *NWQuantity*, \$*z*], read, −, R>, the p = "*ex*:*NWQuantity*" and the literal 127 are hidden.

Example 2: According to <Dave, [*NuclearWeapon*, \$*y*, \$*z*], read, −, L>, Dave cannot show all properties of the class *NuclearWeapon*. If all properties of a class or an instance should be invisible, the whole class or instance should be invisible, e.g., in Fig. 2, <*ex*:*NuclearWeapon rdf*:*ID* = "*Titan*"> … </*ex*:*NulcearWeapon*> is hidden.

5. Explicit Authorization Propagation Policy

When the *type* of an authorization is R, the authorization affects lower classes or properties by inheritance. In this section, we first explain the authorization propagation related with the basic subsumption relationships in RDF Semantics [8], *subClassOf* and *subPropertyOf*, and then related with some primary subsumption relationships in OWL Semantics [9], *unionOf*, *intersectionOf*, and *oneOf*.

5.1 *rdfs*:*subClassOf* Relationship

As explained in Definition 2, if $c_i \subset c_j$, c_i inherits the properties of c_j. Therefore, we define that when an authorization ca_j is specified for a property p_k of c_j, ca_j also affects the property p_k of c_i. We denote this *subClassOf* propagation as $ca_j+ \rightarrow ca_i+$ or $ca_j- \rightarrow ca_i-$. It is natural that the instances of c_j and c_i follow the authorizations ca_j and ca_i, respectively.

Example 3: When ca_j = <Dave, [*SpecialWeapon*, *SWQuantity*, \$*z*], read, −, R> is specified, ca_j derives <Dave, [*NuclearWeapon*, *SWQuantity*, \$*z*], read, −, R> and <Dave, [*Missile*, *SWQuantity*, \$*z*], read, −, R> by the propagation policy $ca_j- \rightarrow ca_i-$. When ca_j = <Dave, [*SpecialWeapon*, \$*y*, \$*z*], read, −, R> is specified, due to $p(ca_j)$ = \$*y*, ca_j is also applied to lower classes inheriting all properties of *SpecialWeapon*. That is, ca_j derives the following authorizations: <Dave, [*NuclearWeapon*, *SWQuantity*, \$*z*], read, −, R>, <Dave, [*Missile*, *SWQuantity*, \$*z*], read, −, R>, <Dave, [*NuclearWeapon*, *locatedIn*, \$*z*], read, −, R>, and <Dave, [*Missile*, *locatedIn*, \$*z*], read, −, R>.

Example 4: ca_j = <Dave, [*SpecialWeapon*, *, *], read, −, R> disallows accessing all their own properties of the lower classes as well as all inherited properties. That is, ca_j also derives <Dave, [*NuclearWeapon*, \$*y*, \$*z*], read, −, R>, and <Dave, [*Missile*, \$*y*, \$*z*], read, −, R>.

Definition 5 (* pattern): This pattern is represented as [*s*, *, *] as in the above example. This is a special security object pattern reserved for conveniently matching all properties of *s*'s lower classes as well as *s*.

5.2 *rdfs*:*subPropertyOf* Relationship

We define that if $p_i \subset p_j$, an authorization pa_j for p_j also affects the property p_i. We denote this *subPropertyOf* propagation as $pa_j+ \rightarrow pa_i+$ or $pa_j- \rightarrow pa_i-$. Also, instances having the property p_j and p_i follow the authorizations pa_j and pa_i, respectively.

Example 5: When pa_j = <Dave, [*ConventionalWeapon*, *CWQuantity*, \$*z*], read, −, R> is specified, pa_j also derives <Dave, [*Rifle*, *RQuantity*, \$*z*], read, −, R>. In the case of pa_j = <Dave, [*SpecialWeapon*, \$*y*, \$*z*], read, −, R>, pa_j also affects all subproperties of all properties of *SpecialWeapon*. For the security object * pattern <Dave, [*SpecialWeapon*, *, *], read, −, R>, pa_j also affects all subproperties of all properties of all subclasses of *SpecialWeapon*.

5.3 *owl*:*unionOf* Relationship

The above authorization propagation for the basic subsumption relationships can be easily expanded into the more complex subsumption relationships in OWL. First, let us consider the *unionOf* relationship. As in Definition 2, this relationship states that a subsuming class contains all instances of the subsumed classes. Therefore, when $c_j = (c_i \cup c_k)$, we can consider the explicit propagation $ca_j \rightarrow ca_i$ as in the *subClassOf* relationship.

Example 6: Figure 3 (b) shows the RDF graph for the OWL document of Fig. 3 (a) (Refer Sect. 4.1 in the OWL Semantics [9] for mapping OWL to RDF graph). When ca_j = <Dave, [*NamedPizza*, *, *], read, −, R> is specified, due to $p(ca_j)$ = *, ca_j is also applied to the subsumed classes. That is, ca_j- derives the following authorizations: <Dave, [*ItalianPizza*, \$y, \$z], read, −, R> and <Dave, [*AmericanPizza*, \$y, \$z], read, −, R>.

This propagation can be similarly applied to the *owl*:*oneOf* relationship which specifies the members of a class are exactly the set of enumerated individuals.

5.4 *owl*:*intersectionOf* Relationship

The *intersectionOf* relationship states that a subsumed class is exactly the intersection of the subsuming classes. Therefore, when $c_j = (c_i \cap c_k)$, we can consider the explicit propagation $ca_i \rightarrow ca_j$. Note that since the instances of c_j are shared by the c_i and c_k, c_i becomes the subsuming class and c_j becomes the subsumed class contrary to the *unionOf* relationship.

Example 7: In Fig. 3, when ca_i- = <Dave, [*SpicyPizza*, *, *], read, −, R> is specified, ca_i- is also applied to the subsumed class *SpicyAmericanPizza*. That is, ca_i- derives the authorization <Dave, [*SpicyAmericanPizza*, \$y, \$z], read, −, R>.

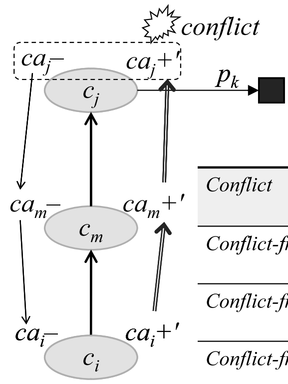
6. Implicit Authorization Propagation Policy and Authorization Conflict Problem

In this section, we explain the implicit authorization propagation by ontology inference, and analyze the authorization conflict problem between the explicit propagation and the implicit propagation.

6.1 *rdfs*:*subClassOf* Inference

The authorization conflicts in SW access control can be classified into two types. One is the explicit authorization conflict and another is the implicit authorization conflict. The explicit authorization conflict addresses that there are several authorizations explicitly having different *sign* values for the same security object. On the contrary, the implicit authorization conflict addresses that although there are no explicit authorization conflict, a conflict can occur due to ontology inference. Since the explicit authorization conflict is a trivial problem, we concentrate on the implicit authorization conflict in this paper.

• $c_i \subset c_j$, $ca_j- \rightarrow ca_i-$, $ca_i+' \Rightarrow ca_j+'$ (*conflict*) : This is the unique condition for the implicit authorization conflict in the *subClassOf* inference. Let us consider the RDF graph of Fig. 4 where $c_i \subset c_m \subset c_j$, and the authorization propagation of the first row in the table. When the authorization



	Explicit propagation	Implicit propagation
Conflict	ca_j- -> ca_m- ca_m- -> ca_i-	ca_i+' => ca_m+' ca_m+' => ca_j+'
Conflict-free	ca_j- -> ca_m- ca_m- -> ca_i-	ca_i-' no propagation
Conflict-free	ca_j+ -> ca_m+ ca_m+ -> ca_i+	ca_i-' no propagation
Conflict-free	ca_j+ -> ca_m+ ca_m+ -> ca_i+	ca_i+' => ca_m+' ca_m+' => ca_j+'

Fig. 4 Authorization conflict in *subClassOf* inference.

ca_j- is first specified for $pt_j = [c_j, p_k, \$z]$, the explicit authorization propagation arises: $ca_j- \rightarrow ca_m- \rightarrow ca_i-$. However, when the authorization ca_i+' is afterwards specified for $pt_i = [c_i, p_k, \$z]$, there is the implicit authorization propagation $ca_i+' \Rightarrow ca_m+' \Rightarrow ca_j+'$. This is because the security object pt_j can be inferred from pt_i by the *subClassOf* inference. Since $sign(ca_j+') \neq sign(ca_j-)$, an authorization conflict occurs. Similarly, when $c_i \subset c_j$, $ca_j+ \rightarrow ca_i+$, and a new authorization ca_j- with *type* = L is specified afterwards, there is also a conflict. Remind of that *type* = L indicates there is no explicit authorization propagation by the Definition 4. Therefore, just the ca_j+ is overwritten by ca_j-.

Example 8: ca_j = <Dave, [*Weapon*, *manufacturedBy*, \$z], read, −, R> drives ca_i = <Dave, [*NuclearWeapon*, *manufacturedBy*, \$z], read, −, R> by the explicit authorization propagation. Then suppose that the authorization ca_i+' = <Dave, [*NuclearWeapon*, *manufacturedBy*, \$z], read, +, R> is re-specified. Since [*Weapon*, *manufacturedBy*, \$z] can be inferred from [*NuclearWeapon*, *manufacturedBy*, \$z], ca_i+' must be also applied to [*Weapon*, *manufacturedBy*, \$z]. That is, ca_i+' drives ca_j+' = <Dave, [*Weapon*, *manufacturedBy*, \$z], read, +, L> by the implicit authorization propagation. Since $sign(ca_j+') \neq sign(ca_j-)$, this is conflict. Here, note that the *type* (= 'R' or 'L') is just related with the explicit authorization propagation by the Definition 4. The *type* has nothing to do with the implicit authorization propagation. Therfore, we simply represent that all of the implicitly propagated authorizations have just the type 'L' regardless of the 'L' or 'R' of the specified authorization.

• $c_i \subset c_j$, $ca_j- \rightarrow ca_i-$, $ca_i-' \nRightarrow ca_j-'$ (*conflict-free*) : In the same manner, as for the second row of the table in Fig. 4, let us consider $ca_j- \rightarrow ca_m- \rightarrow ca_i-$ and a new authorization ca_i-' specified afterwards for $pt_i = [c_i, p_k, \$z]$. In this case, since ca_i-' disallows accessing pt_i, any related inference cannot occur. Hence, this case is conflict-free.

• $c_i \subset c_j$, $ca_j+ \rightarrow ca_i+$, $ca_i-' \nRightarrow ca_j-'$ (*conflict-free*) : Let us consider $ca_j+ \rightarrow ca_m+ \rightarrow ca_i+$ and a new authorization ca_i-' specified afterwards as in the third row of the table.

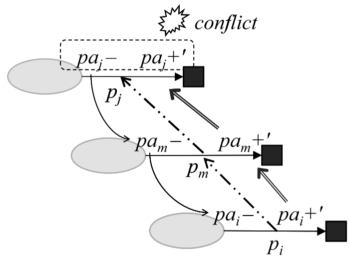


Fig. 5 Authorization conflict in *subPropertyOf* inference.

As in the previous case, since ca_i-' disallows accessing pt_i, there can be no conflict.

• $c_i \subset c_j$, $ca_j+ \rightarrow ca_i+$, $ca_i+' \Rightarrow ca_j+'$ (*conflict-free*): Let us consider $ca_j+ \rightarrow ca_m+ \rightarrow ca_i+$ and a new authorization ca_i+' specified afterwards as in the fourth row of the table. In this case, since $ca_i+' \Rightarrow ca_j+'$ and $sign(ca_j+') \equiv sign(ca_j+)$, there is no conflict.

6.2 *rdfs*:*subPropertyOf* Inference

When $p_i \subset p_j$, $pa_j- \rightarrow pa_i-$, and $pa_i+' \Rightarrow pa_j+'$ as in the *subClassOf* inference, there is a conflict. Figure 5 depicts this situation. For example, pa_j = <Dave, [*Weapon*, *Quantity*, $z], read, −, R> drives pa_i = <Dave, [*NuclearWeapon*, *NWQuantity*, $z], read, −, R> as *NWQuantity* is the subproperty of *Quantity*. If the authorization pa_i+' = <Dave, [*NuclearWeapon*, *NWQuantity*, $z], read, +, R> is specified afterwards, this is conflict. This is because [*Weapon*, *Quantity*, $z] can be inferred from [*NuclearWeapon*, *NWQuantity*, $z] by the *subPropertyOf* inference and $sign(pa_j+') \neq sign(pa_j-)$.

6.3 *owl*:*unionOf* and *owl*:*intersectionOf* Inferences

First, when $c_j = (c_i \cup c_k)$, $ca_j- \rightarrow ca_i-$, and $ca_i+' \Rightarrow ca_j+'$, the *unionOf* inference also has a conflict. For example, suppose that ca_i+' = <Dave, [*ItalianPizza*, $y, $z], read, −, L> is specified afterwards for the Example 6. Since the instance of *ItalianPizza* is also the instance of *NamedPizza*, $ca_i+' \Rightarrow ca_j+'$ and $sign(ca_j-) \neq sign(ca_j+')$.

We can also consider the same conflict related with the *oneOf* inference.

As for the *intersectionOf* inference, when $c_j = (c_i \cap c_k)$, $ca_i- \rightarrow ca_j-$, and $ca_j+' \Rightarrow ca_i+'$, there also is a coflict. For example, supposing ca_j+' = <Dave, [*SpicyAmericanPizza*, $y, $z], read, +, L> is specified afterwards for the Example 7, $ca_j+' \Rightarrow ca_i+'$ and $sign(ca_i-) \neq sign(ca_i+')$.

7. Efficiently Detecting Authorization Conflict Using Graph Labeling

Jain and Farkas [3] introduced a little inefficient algorithm for detecting the authorization conflict. In their method, whenever a new authorization is specified, the corresponding security labels are first assigned to all RDF triples. Then RDF inference is performed for all RDF triples, and for each inferred triple, it is checked if there is a security violation. That is, their method simply checks all RDF instances. This is inefficient when the number of instances become larger. Therefore, we suggest an efficient conflict detection method using graph labeling techniques [12], [13]. The basic idea is, based on the observation in the previous section, to check only the ancestor authorizations with *sign* (−) when a new authorization with *sign* (+) is specified whereas to check only the descendant authorizations with *sign* (+) when a new authorization with *sign* (−) and *type* L is specified. Here, the *ancestor/descendant* authorization means an authorization of which the security object has subsumption relationship with the security object of the new authorization. For example, a_j = <Dave, [*SpecialWeapon*, $y, $z], read, −, L> is the ancestor authorization of a_i = <Dave, [*NuclearWeapon*, $y, $z], read, +, L> because $s(a_j)$ = *SpecialWeapon* is an ancestor node of $s(a_i)$ = *NuclearWeapon* in the RDF graph of Fig. 1. In order to efficiently identify the ancestor/descendant relationship, we use the graph labeling techniques [12], [13]. In this paper, we skip the details of the graph labeling techniques. We only discuss how we can detect efficiently an authorization conflict using the information of the ancestor/descendant relationship.

The suggested detection algorithm of Fig. 8 selectively tests the cases of authorization conflict according to the conflict decision table (CDT) of Figs. 6 (a) and (b). The CDT summarizes the possibility of conflict according to the type of s and p values in Definition 1, 3, and 5: $s \in$ {class URI constant, instance URI constant} and $p \in$ {property URI constant, $y, *}. In the case of s = $x, we can get the URI constant of the highest upper class having the property p. For example, since all classes have the property *WCode* but the highest upper class is *Weapon*, s([$x, *WCode*, $z]) = *Weapon*. We now prove the CDT through the following examples. Note that since we have shown the subsumption relationships in OWL are similar to the *subClassOf* in Sects. 5 and 6, it is proven around the *subClassOf* relationship.

Example 9: Let us consider that an authorization $R5$ = <Dave, [*NuclearWeapon*, $y, $z], read, +, L> is additionally specified against the authorizations of Fig. 7. In this case, $R1$ is the only authorization which can have conflict with $R5$. Because *SpecialWeapon* is the ancestor class for *NuclearWeapon*, $R1$ has (−) *sign*, and $R5$ has (+) *sign*. Also, since $s(R5) \in$ class URI, $s(R1) \in$ class URI, $p(R5)$ = $y, and $p(R1)$ = $y, this is absolutely conflict according to the rule 1 in Fig. 6 (a). This is because "$p(R5)$ = $y" means that $R5$ is also specified for all properties inherited from *SpecialWeapon*. This example also illustrates the rules 3, 13, and 15.

Example 10: Again, let us consider a new authorization $R5$ = <Dave, [*ConventionalWeapon*, *CWQuantity*, $z],

descendant(+) \ ancestor(−)		p=$y		p=*		p=URI	
		s=class URI	s=instance URI	s=class URI	s=instance URI	s=class URI	s=instance URI
p=$y	s=class URI	1) O	2) X	3) O	4) X	5) O	6) X
	s=instance URI	7) O	8) X	9) O	10) X	11) O	12) X
p=*	s=class URI	13) O	14) X	15) O	16) X	17) O	18) X
	s=instance URI	19) O	20) X	21) O	22) X	23) O	24) X
p= URI	s=class URI	25) ?	26) X	27) ?	28) X	29) ?	30) X
	s=instance URI	31) ?	32) X	33) ?	34) X	35) ?	36) X

(a) Cases of $rdfs{:}subClassOf$, $owl{:}unionOf$, $owl{:}intersectionOf$, and $owl{:}oneOf$

descendant(+) \ ancestor(−)		p=$y		p=*		p=URI	
		s=class URI	s=instance URI	s=class URI	s=instance URI	s=class URI	s=instance URI
p=$y	s=class URI	37) O	38) X	39) O	40) X	41) O	42) X
	s=instance URI	43) O	44) X	45) O	46) X	47) O	48) X
p=*	s=class URI	49) O	50) X	51) O	52) X	53) O	54) X
	s=instance URI	55) O	56) X	57) O	58) X	59) O	60) X
p= URI	s=class URI	61) O	62) X	63) O	64) X	65) O	66) X
	s=instance URI	67) O	68) X	69) O	70) X	71) O	72) X

(b) Case of $rdfs{:}subPropertyOf$

Fig. 6 Conflict decesion tables (O: conflict, X: conflict-free, ?: verification is required).

$R1$: <Dave, [*SpecialWeapon*, \$y, \$z], read, −, R>
$R2$: <Dave, [*Rifle*, \$y, \$z], read, +, L>
$R3$: <Dave, [*ConventionalWeapon*, *CWQuantity*, \$z], read, +, R>
$R4$: <Dave, [*Tomahawk*, \$y, \$z], read, −, R>

Fig. 7 Sample authorizations specified in an RDF authorization store.

read, −, L>. In this case, $R5$ has conflict with $R2$ according to the rule 5 in Fig. 6 (a): $sign(R5) = -$, $sign(R2) = +$, $s(R5) \in$ class URI, $s(R2) \in$ class URI, $p(R5) \in$ property URI, and $p(R2) = \$y$. "$p(R2) = \$y$" means that $R2$ is also applied to the property *CWQuantity* of *Rifle* inherited from *ConventionalWeapon*. This example also illustrates the rule 17.

Example 11: Let us consider a new authorization $R5$ = <Dave, [*NuclearWeapon*, *locatedIn*, \$z], read, +, L>. Since $sign(R1) = -$, $sign(R5) = +$, $s(R1) \in$ class URI, $s(R5) \in$ class URI, $p(R1) = \$y$, and $p(R5) \in$ property URI, this requires a conflict verification according to the rule 25 in Fig. 6 (a). In this example, the final decision is conflict because the property *locatedIn* is inherited from *SpecialWeapon*. However, if the property is not inherited, that is conflict-free. This example also illustrates the rule 27. Similarly, in the case of the rule 29, it is required to check whether or not the two property URIs are the same. If equal, that is conflict, otherwise conflict-free.

Example 12: First, let us consider a new authorization $R5$ = <Dave, [*Titan*, \$y, \$z], read, +, L> and the former authorization $R1$. Since $sign(R1) = -$, $sign(R5) = +$, $s(R1) \in$ class URI, $s(R5) \in$ instance URI, $p(R1) = \$y$, and $p(R5) \in \$y$, this is absolutely conflict according to the rule 7 in Fig. 6 (a). This is because the instance *Titan* certainly inherits all properties from *SpecialWeapon*. Next, let us consider $R5$ and $R4$. In this case, since $Titan \neq Tomahawk$, that is, two instances are each other different objects, this is absolutely conflict-free. Furthermore, since an instance which is represented by URI must be unique in its web ontology, an arbitrary authorization pairs both having an instance URI for s is absolutely conflict-free. Theorem 1 represents this characteristic of the instance URI which makes the authorization conflict detection more simplified.

Theorem 1: (1) A descendant authorization, which has the s value of the instance URI type and (+) *sign*, can have conflict with only ancestor authorizations which has the s value of the class URI type and (−) *sign*. (2) An ancestor authorization with the s value of the instance URI type is absolutely conflict-free with any descendant authorization.

Proof: (1) Since the URI value of an instance is unique in its web ontology, a descendant authorization a_i with $s(a_i) \in$ instance URI and $sign(a_i) = +$ can have conflict with an ancestor authorization a_j with $s(a_j) \in$ class URI and $sign(a_j) = -$. (2) Next, since an instance can be interpreted into only the instance of its upper class by the subsumption inference, an ancestor authorization a_i with $s(a_i) \in$ instance URI cannot have conflict with any descendant authorization a_j. □

By the case (1) in Theorem 1, the rules 8, 10, 12, 20, 22, 24, 32, 34, 36, 44, 46, 48, 56, 58, 60, 68, 70, and 72 in the CDT are conflict-free. And by the case (2) in Theorem 1, the rules 2, 4, 6, 14, 16, 18, 26, 28, 30, 38, 40, 42, 50, 52, 54, 62, 64, and 66 are conflict-free.

Example 13: In the *subPropertyOf* CDT of Fig. 6 (b), all cases are conflict except the cases corresponding to Theorem 1. The *subPropertyOf* CDT does not have the cases of conflict-free for a non-inherited property as in Example 11. This is because the *subPropertyOf* relationship directly defines the subsumption relationship among properties rather than classes. For example, let us consider a new authorization $R5$ = <Dave, [*NuclearWeapn*, *NWQuantity*, \$z], read, +, L>. By the rule 25 of the *subClassOf* CDT, this requires conflict verification against $R1$, but by the rule 61 of the *subPropertyOf* CDT, this is absolutely conflict. Because although $s(R1) = \$y$, it means the property *SWQuantity* in the *subPropertyOf* relationship (See the RDF graph of Fig. 1). A new authorizations $R6$ = <Dave, [\$x, *WCode*, \$z], read, −, L> does not need to be checked for the *subPropertyOf* CDT. It should be checked only for

Algorithm Method1
Input: An authorization a_i to be specified
Output: Conflict authorization set $Conflict\_Set$

```
  /* check the ancestor authorizations with (−) sign in the subClassOf, unionOf, intersectionOf, or oneOf relationships*/
if sign(a_i) = '+' then
   RS := retrieve the ancestor authorizations with (−) sign in the subClassOf, unionOf, intersectionOf, or oneOf relationships;
if RS is not empty then
   foreach rs_i ∈ RS do
      if ((p(a_i) ∈ {property URI} ∧ p(rs_i) = $y ∧ p(a_i) ∉ properties(s(rs_i))) ∨ (p(a_i), p(rs_i) ∈ property URI ∧ p(a_i) ≠ p(rs_i)))
                                              /* this checks the case of Example 11 */
         Conflict-free; continue;
      else
         if s(rs_i) ∈ instance URI then Conflict-free; continue; /* this checks the case of Theorem 1 */
         else
            Conflict;
            Conflict_Set := Conflict_Set ∪ rs_i.authorizationID
  /* check the descendant authorizations with (+) sign in the subClassOf, unionOf, intersectionOf, or oneOf relationships*/
if sign(a_i) = '−' ∧ type(a_i) = 'L' then
   RS := retrieve the descendant authorizations with (+) sign in the subClassOf, unionOf, intersectionOf, or oneOf relationships;
if RS is not empty then
   foreach rs_i ∈ RS do
      if ((p(rs_i) ∈ {property URI} ∧ p(a_i) = $y ∧ p(rs_i) ∉ properties(s(a_i))) ∨ (p(a_i), p(rs_i) ∈ property URI ∧ p(a_i) ≠ p(rs_i)))
                                              /* this checks the case of Example 11 */
         Conflict-free; continue;
      else
         if s(a_i) ∈ instance URI then Conflict-free; continue; /* this checks the case of Theorem 1 */
         else
            Conflict;
            Conflict_Set := Conflict_Set ∪ rs_i.authorizationID
  /* check the ancestor authorizations with (−) sign in the subPropertyOf relationship*/
if sign(a_i) = '+' then
   RS := retrieve the ancestor authorizations with (−) sign in the subPropertyOf relationship;
if RS is not empty then
   foreach rs_i ∈ RS do
      if s(rs_i) ∈ instance URI then Conflict-free; continue; /* this checks the case of Theorem 1 */
      else
         Conflict;
         Conflict_Set := Conflict_Set ∪ rs_i.authorizationID
  /* check the descendant authorizations with (+) sign in the subPropertyOf relationship*/
if sign(a_i) = '−' ∧ type(a_i) = 'L' then
   RS := retrieve the descendant authorizations with (+) sign in the subPropertyOf relationship;
if RS is not empty then
   foreach rs_i ∈ RS do
      if s(a_i) ∈ instance URI then Conflict-free; continue; /* this checks the case of Theorem 1 */
      else
         Conflict;
         Conflict_Set := Conflict_Set ∪ rs_i.authorizationID
return Conflict_Set;
```

Fig. 8 Our suggested authorization conflict detection algorithm.

the $subClassOf$ CDT. Because $p(R6) = WCode$ is not included in the $subPropertyOf$ relationship.

8. Experiments

8.1 Experimental Setup

In this section, we compare our detection method with Jain and Farkas' method [3]. Since we could not obtain the optimized implementation of Jain and Farkas' method, we simulated it as follows. First, a DAG as in Fig. 9 is generated according to the experimental parameter $\#C$, $\#P$, and $\#S$ in Table 2. The circle represents a class and the rectangle represents a property. The parameter $\#S$ is the average number of $subClassOf$ relationships for each class. For example, in Fig. 9, since $c3$ is the subclass of $c1$ and $c4$, its $\#S$ is two. In this experiment, we simplified the inferences related with $unionOf$ and $intersectionOf$. They can be simply regarded as a set of $subClassOf$s having the OR and AND arch as in the DAG. For example, $c2 = (c6 \cup c7)$ and $c8 = (c3 \cap c4 \cap c5)$. In the data structure of properties, the variable $before\_sign$ stores the $sign$ value assigned by a formerly specified authorization and the variable $after\_sign$ stores the $sign$ value by the currently specified authorization. In the data structure of classes, each class has storage spaces for all inherited properties as well as its own properties. For

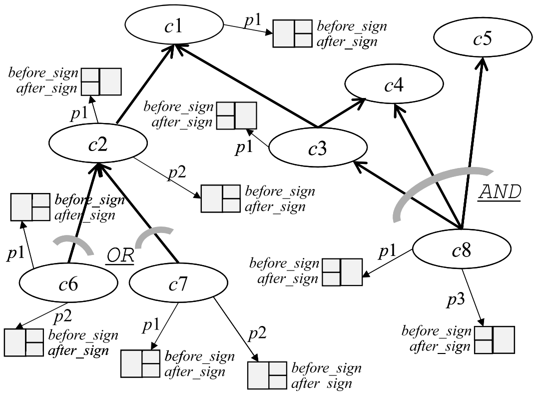


Fig. 9 Test DAG generation.

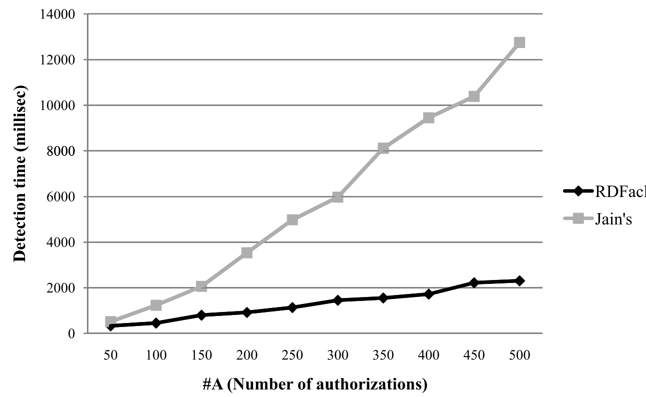


Fig. 10 Detection time comparison according to #A when #C = 200 and #S = 20.

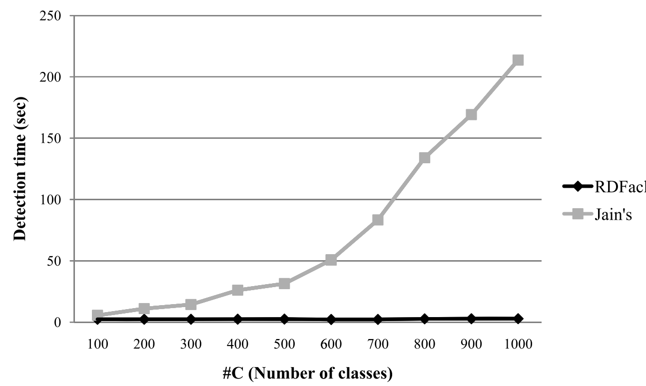


Fig. 11 Detection time comparison according to #C when #A = 500 and #S = 2.

Table 2 Workload parameters for generating RDF authorizations and data.

Parameter	Range	Description
#C	1 to 1,000	Number of classes in a DAG
#P	1 to 5	Average number of properties for each class in a DAG
#S	1 to 20	Average number of *subClassOf* for each class in a DAG
#A	1 to 500	Number of specified authorizations

example, all lower classes of $c1$ have the property $p1$.

Next, whenever a new authorization is inputted, the following conflict check is performed for Jain and Farkas' method. First, with a breadth-first traversal of the DAG, the *sign* values of the formerly specified authorizations are assigned to all variables of $before\_sign$ by the explicit authorization propagation. This step makes all RDF triples have their own most specific security *sign* value. Then the *sign* value of the new authorization is assigned to some corresponding variables of $after\_sign$ also with a breadth-first traversal. Again, with a breadth-first traversal, it is checked if there is any property with $before\_sign$ = '−' and $after\_sign$ = '+'. If such a property exists, it is a conflict.

Our suggested algorithm also uses the randomly generated DAG and authorizations. As a graph labeling technique, we used the prime number labeling scheme suggested by Wu et al. [13]. All experiments were performed on a Windows XP computer with 1 GB of memory and 3.20 GHz Pentium(R) IV CPU; all codes were written in Java.

8.2 Experimental Results

We experimented according to the parameters in Table 2. Having higher values of #C and #S means that the DAG becomes larger, that is, the size of RDF/OWL schema and data becomes larger and more complex. Also, having a higher value of #A means that there are much more authorizations to be checked against the authorization conflict. First, the graph of Fig. 10 shows the conflict detection time according to #A when #C = 200 and #S = 20. In the experiment, a new authorization was randomly generated for a test DAG, and its authorization conflict check was performed against the formerly specified authorizations. If there is no conflict, the new authorization is added to the former authorization set. This procedure was performed as much as the parameter #A. The graph shows that our RDFacl has much lower detection time than Jain and Farkas' method according to #A. We can also see that the difference becomes significant as the number of authorizations increases.

Next, the graph of Fig. 11 shows the conflict detection time according to #C when #A = 500 and #S = 2. It also shows that our RDFacl has significantly lower detection time and lower increasing rate. Through some additional experiments which vary the parameter values, we also confirmed that there is a significant difference between two methods. Although our experimental results are approximate due to the simulation of Jain and Farkas's method, the significant difference apparently shows that our suggested method has more improved detection capability than the existing method.

Although in this paper we omitted showing how our method stores access authorizations with graph labeling, regarding the efficiency of memory usage, our suggested method spends storage space approximately twice as much as the simulated Jain and Farkas's method. We believe that the additional storage cost is not significant considering the benefit of the improved detection time. In our implementa-

Table 3 Storage space (Bytes) per one access authorization.

	in the main table	in the auxiliary table	total
RDFacl	44	57	101
Jain's	44	*None*	44

tion, in order to maintain the information related with graph labeling, an auxiliary authorization table is separately required besides the main authorization table in the simulated Jain and Farkas's method. Table 3 shows the additional storage cost per one access authorization in our method.

9. Conclusions and Future Work

The RDF authorization conflict problem is an important problem in RDF access control because RDF data are related with ontology inference unlike XML data. In this paper, we have explained the RDF authorization conflict problem based on two concepts of the explicit and the implicit authorization propagation.

The key ontology inference in RDF is related with the subsumption relationships, *subClassOf* and *subPropertyOf*. Therefore, in this paper, we have focused on analyzing the authorization conflict problem in the subsumption inference. We also have shown that the analysis results can be naturally applied to the primary subsumption relationships in OWL, *unionOf*, *intersectionOf*, and *oneOf*.

Currently, our future work is to expand the suggested RDFacl into more complex ontological relationships in OWL [14], e.g., *Restriction*, *equivalentClass*, *sameAs*, *equivalentProperty*, and *complementOf*, and also analyze the authorization conflict problem in related inferences.

Acknowledgments

We would like to thank the anonymous referees for their valuable comments on earlier draft of this paper.

This study is supported in part by the Second Stage of BK21. In addition, this work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R01-2006-000-10609-0).

References

[1] L. Qin and V. Atluri, "Concept-level access control for the semantic Web," Proc. ACM Workshop on XML Security 2003, pp.94–103, Oct. 2003.

[2] P. Reddivari, T. Finin, and A. Joshi, "Policy-based access control for an RDF store," Proc. Policy Management for the Web Workshop, pp.78–83, May 2005.

[3] A. Jain and C. Farkas, "Secure resource description framework: An access control model," Proc. 11th ACM Symposium on Access Control Models and Technologies, pp.121–129, June 2006.

[4] T. Moses, "OASIS extensible access control markup language (XACML) Version 2.0," OASIS Standard, Feb. 2005.

[5] E. Damiani, S.D.C. Vimercati, S. Paraboschi, and P. Samarati, "A fine-grained access control system for XML documents," ACM Trans. Information and System Security, vol.5, no.2, pp.169–202, 2002.

[6] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti, "Specifying and enforcing access control policies for XML document sources," World Wide Web Journal, vol.3, no.3, pp.139–151, 2000.

[7] E. Bertino and E. Ferrari, "Secure and selective dissemination of XML documents," ACM Trans. Information and System Security, vol.5, no.3, pp.290–331, 2002.

[8] RDF Semantics, W3C Recommendation, http://www.w3.org/TR/rdf-mt/

[9] OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation, http://www.w3.org/TR/owl-semantics/

[10] S. Kaushik, D. Wijesekera, and P. Ammann, "Policy-based dissemination of partial Web-ontologies," Proc. 2005 Workshop on Secure Web Services, pp.43–52, Nov. 2005.

[11] J. Kim and S. Park, "An RDF triple based access authorization specification considering subsumption relationships," Proc. First International Workshop on Secure Semantic Web, pp.462–469, April 2008.

[12] V. Christophides, G. Karvounarakis, D. Plexousakis, M. Scholl, and S. Tourtounis, "Optimizing taxonomic semantic Web queries using labeling schemes," Journal of Web Semantics, vol.11, no.1, pp.207–228, Nov. 2003.

[13] G. Wu, K. Zhang, C. Liu, and J. Li, "Adapting prime number labeling scheme for directed acyclic graphs," Proc. Database Systems for Advanced Applications (DASFAA), pp.787–796, April 2006.

[14] OWL Web Ontology Language Overview, W3C Recommendation, http://www.w3.org/TR/owl-features/

Jaehoon Kim received the B.S. and M.S. degrees in computer science from Konkuk University in 1997 and 1999, respectively and the Ph.D. degree in computer science from Sogang University in 2005. He worked as a senior researcher at Telecommunication R&D center in Samsung Electronics Company from 2005 to 2006. From 2006, he is a research professor of computer science at Sogang University. His major research areas are Web database, database security, data privacy, Semantic Web, and data mining.

Seog Park received the B.S. degree in computer science from Seoul National University, Korea, in 1978, the M.S. and the Ph.D. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1980 and 1983, respectively. He is a professor of computer science at Sogang University, Seoul, Korea. Since 1983, he has been working in the Department of Computer Science of the College of Engineering, Sogang University. His major research areas are database security, real-time systems, data warehouse, digital library and web database. Dr. Park is a member of the IEEE Computer Society, ACM, and the Korea Information Science Society. Also, he has been a member of Database Systems for Advanced Applications (DASFAA) steering committee since 1999.