Authorization Conflict Problems in Combining RIF Rules with RDF Data

Jaehoon KIM\(^{1)}\), Member

SUMMARY Resource Description Framework (RDF) access control suffers from an authorization conflict problem caused by RDF inference. When an access authorization is specified, it can lie in conflict with other access authorizations that have the opposite security sign as a result of RDF inference. In our former study, we analyzed the authorization conflict problem caused by subsumption inference, which is the key inference in RDF. The Rule Interchange Format (RIF) is a Web standard rule language recommended by W3C, and can be combined with RDF data. Therefore, as in RDF inference, an authorization conflict can be caused by RIF inference. In addition, this authorization conflict can arise as a result of the interaction of RIF inference and RDF inference rather than of RIF inference alone. In this paper, we analyze the authorization conflict problem caused by RIF inference and suggest an efficient authorization conflict detection algorithm. The algorithm exploits the graph labeling-based algorithm proposed in our earlier paper. Through experiments, we show that the performance of the graph labeling-based algorithm is outstanding for large RDF data.

key words: access control, RDF, RIF, authorization conflict, inference

1. Introduction

The Resource Description Framework (RDF) and the Web Ontology Language (OWL) are markup languages for representing information over the Web through defining and using a common ontology. In addition, RDF and OWL are used as the standard data exchange formats between heterogeneous ontology languages, and they sometimes replace existing ontology languages. Recently, related to the problem of securely accessing RDF data, several studies on RDF access control have been conducted [1]–[7]. Among these, the studies presented in [3] and [6] dealt with an authorization conflict problem caused by RDF inference, and explained the problem at a more fine-grained level of the RDF triple. The authorization conflict problem caused by RDF inference can be illustrated by the following simple example. Suppose that a user is forbidden to access an RDF triple $[s_1, p_1, o_1]$. If $s_2$ is the subclass of $s_1$, then can access to the triple $[s_2, p_1, o_1]$ be allowed? The answer is “no”. This is because $[s_1, p_1, o_1]$ can be inferred from $[s_2, p_1, o_1]$ by RDF subsumption inference. In our former study [6], we addressed the authorization conflict problem that arises particularly from subsumption inference, which is the key inference in RDF, and suggested an efficient authorization conflict detection method that uses graph labeling techniques [8].

In this paper, we also consider the authorization conflict problem caused by Rule Interchange Format (RIF) inference. RIF [9] is a Web standard rule language recommended by W3C and can be applied to RDF data in order to define additional inference rules. While RDF entailment rules [10] are statically defined for RDF inference, RIF rules can be added and deleted at any time. RIF is dynamic. As in the case of RDF inference, when a new RIF rule is specified, we have to consider the authorization conflict problem caused by the RIF inference. For example, let us consider the RIF rule $[s, p_1, o_1] \land [s, p_2, o_2] \rightarrow [s, p_3, o_3]$. This means that the triple $[s, p_3, o_3]$ is inferred from the triples $[s, p_1, o_1]$ and $[s, p_2, o_2]$. Assuming an access authorization that denies access to $[s, p_3, o_3]$, we must also consider the authorization propagation that denies access to $[s, p_3, o_3]$. This is because the possibility of access to $[s, p_3, o_3]$ indicates that the RIF rule has been executed successfully through reading the triple $[s, p_1, o_1]$. If this has occurred, inference has been violated. We also have to consider that this authorization propagation resulting from RIF inference can be combined with RDF inference. For example, the propagated access authorization that denies access to $[s, p_3, o_3]$ can conflict with other access authorizations caused by RDF inference. In this paper, we analyze the authorization conflict problem caused by RIF inference, and we suggest an efficient authorization conflict detection algorithm that considers RIF and RDF inferences. The algorithm utilizes the graph labeling-based algorithm suggested in the former study [6].

The rest of the paper is structured as follows. Section 2 reviews the related work. Section 3 briefly summarizes the formerly proposed authorization model for RDF. In Sect. 4, we analyze the authorization conflict conditions caused by RIF inference, and in Sect. 5, we suggest a conflict detection algorithm. In Sect. 6 the experimental results are presented, and finally, in Sect. 7 we present our conclusions.

2. Related Work

Reddivari et al. [2] considered the RDF triple as the security object in RDF access control. However, they did not deal with the authorization conflict problem caused by RDF inference and also by RIF inference.

Qin and Atluri [1] introduced the concept of the authorization conflict problem caused by ontology inference.
However, the security object that they considered was not the RDF triple and their method was not incorporated in the standard RDF Semantics [10]. They did not consider the RIF inference.

Javanmardi et al. [4] verified authorization propagation in the various ontology relationships in OWL, which are between classes, between class and individual, between individuals, between property and class, between properties, and between property and individual. However, their authorization model did not consider the authorization conflict problem caused by RDF and RIF inferences.

Jain and Farkas [3] considered the authorization conflict problem caused by RDF inference that is incorporated in RDF Semantics. However, their suggested conflict detection algorithm is inefficient when RDF data become large and many access authorizations are specified. Therefore, in our former study [6], we adopted a heuristic approach and suggested an efficient authorization conflict detection algorithm based on graph labeling. Jain and Farkas also did not consider the authorization conflict caused by RIF inference.

Abel et al. [5] introduced a high-level access control specification language that uses the triple pattern, as in the standard RDF query language SPARQL. The specified patterns are combined with a user’s query to enforce the access control. However, they did not consider the authorization conflict problem caused by RDF and RIF inferences.

Papakonstantinou et al. [7] considered the authorization conflict problem caused by RDF inference. Similar to Jain and Farkas’ method, their method checks an authorization conflict by inferring all RDF triples with a security sign. However, their main contribution was to suggest a flexible security sign assignment method for RDF triples. In their method, a concrete security sign value is not assigned, but rather an abstract expression consisting of access tokens and inference operators. Whenever a specific security policy is applied, the expression is dynamically calculated. They also did not consider the authorization conflict problem caused by RIF inference.

3. A Subsumption Hierarchy-Based Authorization Model for RDF

In this section, we briefly introduce the authorization model presented in the former study [6]. Several parts of the former model have been updated.

3.1 RDFacl Graph

RDF data can be represented as an RDF graph consisting of RDF triples [10]. However, the RDF graph can be complex and large because a) the RDF graph includes the RDF triples for RDF/RDFS vocabularies, such as rdfs:Resource, rdfs:Class, rdfs:Property, rdfs:domain, rdfs:range, and so forth, b) it represents user-defined properties as nodes, and c) basically all individuals as well as classes are included. This large and complex RDF graph makes a security officer’s authorization specification task complex and inefficient. Therefore, we developed the RDFacl graph, which is a simplification of the RDF graph. It is based on the subsumption hierarchy structure configured by rdfs:subClassOf, rdfs:subPropertyOf, and rdf:type. It excludes the general RDF/RDFS vocabularies, user-defined properties are represented as edges, and only the individuals specified in an access authorization are depicted in the graph.

Definition 1 (RDFacl graph): This graph represents the subsumption hierarchy structure of RDF data that is configured by rdfs:subClassOf, rdfs:subPropertyOf, and rdf:type. It consists of the triples \([s, p, o]\) as in the RDF graph, but the values for \(s, p, o\) are restricted as follows.

- \(\forall s, \forall o \in \{\text{user-defined classes, individuals specified in an access authorization, blank nodes, literals}\}\)
- \(\forall p \in \{\text{rdfs:subClassOf, rdfs:subPropertyOf, rdf:type, user-defined properties}\}\)

For example, Fig. 1 shows the RDFacl graph for RDF data related to a smart phone application program. It depicts user-defined classes, individuals, and literals as nodes, and subsumption relationships (rdfs:subClassOf, rdfs:subPropertyOf, and rdf:type) and user-defined properties as edges. The depicted individuals are all related to a specific access authorization, as shown in Fig. 2.

3.2 Access Authorization Specification

An access authorization in our system has the following five-tuple form and can be propagated along the hierarchy path of an RDFacl graph.

Definition 2 (access authorization, \(Au\)): An access authorization \(Au\) has a five-tuple form: \(<\text{subj}, \text{obj}, \text{act}, \text{sign}, \text{type}>\).

- \(\text{subj}\) is the subject to whom \(Au\) is granted.
- \(\text{obj}\) refers to a security object, which is a triple pattern, where \(s\) and \(p\) can be substituted by \(Sx\) and \(Sy\), respectively. The variables match an arbitrary value.
- \(\text{act}\) refers to an action performed for the security object. In our study, only the read operation is considered.
- \(\text{sign}\) is (+) if access is allowed, and (−) if access is forbidden.
- \(\text{type}\) is \(R\) (= Recursive) if \(Au\) can be propagated to lower classes, lower properties, or individuals along the hierarchy path of an RDFacl graph and \(L\) (= Local) if \(Au\) is not propagated. We name this propagation explicit authorization propagation.

For example, \(Au_i = \langle\text{programA, [User, $y, $z]}\rangle\), read, \(\sim, R\rangle\) forbids the access to the properties of User itself and by \(\text{type} = R\), it derives the following access authorizations: \(<\text{programA, [Owner, name, $z]}\rangle\), read, \(\sim, R\rangle\), \(<\text{programA, [o1, name, $z]}\rangle\), read, \(\sim, R\rangle\). In our study, the \(o\) in \([s, p, o]\) has the variable \$/z always.
3.3 Authorization Conflict Detection in RDF Inference

When a new access authorization is specified, an authorization conflict with previously specified access authorizations can occur as a result of RDF inference. We named this conflict implicit authorization conflict in our former study. Detection of the implicit authorization conflict is complex and requires significant costs. This is because multiple RDF inferences have to be considered for large RDF data. Jain and Farkas [3] suggested an instance-level authorization conflict detection method, where security signs are allocated to all individuals according to access authorizations, and all RDF inferences are performed for all the individuals. If opposite security signs are propagated for an individual by an RDF inference, this indicates that an authorization conflict has arisen. Although their method is complete and secure, it requires significant costs when data are large. “Complete” means that it considers all RDF inferences and “secure” means that it detects all authorization conflicts caused by the RDF inferences. Considering this scalability problem, we introduced an authorization-level conflict detection method [6], where only access authorizations are examined without instance data being accessed.

The basic idea applied in our method is a heuristic approach. Whenever a new authorization conflict condition is identified, we develop an authorization-level conflict detection module specific to the condition. In this approach, we gradually construct a complete and secure authorization conflict detector that consists of various modules. In our former study, we particularly identified the authorization conflict condition caused by subsumption inference, and suggested a graph labeling-based authorization conflict detection module. In fact, the authorization conflict caused by subsumption inference is the main conflict condition in our authorization model. This is because the RDFacl graph is based on the subsumption hierarchy structure. The following example illustrates the authorization conflict caused by subsumption inference.

Example 1: Let us consider $A_{u_{1}+} = \langle \text{programA}, [\text{Bell}, \text{hasState}, \$z], \text{read}, +, L \rangle$ against $A_{u_{1}-} = \langle \text{programA}, [\text{PhoneNumber}, \text{hasState}, \$z], \text{read}, -, L \rangle$. Two access authorizations lie in conflict. Suppose that $A_{u_{1}+}$ has been first allowed. Then, the following inference is performed.

$$t_1: [b_1, \text{hasState}, \text{off}]$$
$$t_2: [b_1, \text{rdf:type}, \text{Bell}]$$
$$t_3: [\text{Bell}, \text{rdfs:subClassOf}, \text{PhoneNumber}]$$
$$t_4: [b_1, \text{rdf:type}, \text{PhoneNumber}]$$  (by the inference rule rdfs9 in RDF Semantics [10])

This is a security violation of $A_{u_{1}-}$ since programA accesses the RDF triples $t_1$ and $t_4$.

To detect the authorization conflict caused by subsumption inference efficiently, in the previous work [6], we suggested a method in which graph labeling is applied. Figure 3 gives the outline of the suggested algorithm. First, let us briefly see how a prefix-based graph label [8] is assigned to each access authorization. For an RDFacl graph, a Directed Acyclic Graph (DAG) is constructed according to subClassOf and type relationships. Next, the prefix-based graph labeling is performed for the nodes of the DAG. When the $s$
in \([s, p, o]\) in an access authorization matches the class or individual node in the DAG, the corresponding label is allocated to the access authorization. An access authorization can have another graph label because of the subPropertyOf relationship. For the RDFacl graph, another DAG that has properties as nodes is constructed according to the subPropertyOf relationship. Again, the prefix-based graph labeling is performed for the DAG. When the \(p\) in \([s, p, o]\) in an access authorization matches the property node in the DAG, the corresponding label is allocated to the access authorization.

If each access authorization has its graph labels, the authorization conflict caused by the subsumption relationship can be efficiently identified. As in lines 2, 7, 11, and 15, an authorization conflict can exist between ancestor and descendent access authorizations each of which has a different security sign. Let us refer to Example 1 again. We do not consider the case of descendent authorizations having an \((-\)\) sign, since forbidding access to lower classes or properties prevents the subsumption inference to upper classes or properties.

Although two access authorizations whose security signs are opposite lie in a subsumption relationship, a conflict does not necessarily occur. The first case of this kind is when the properties in the two access authorizations are not in an inheritance relationship. Let us consider \(A_{i-}\) for the security object \([s_1, p_1, o_1]\) and \(A_{j+}\) for the security object \([s_2, p_2, o_2]\), where \(s_1\) is the subclass of \(s_2\). For the subClassOf inference, an authorization conflict can arise when \(p_1\) and \(p_2\) lie in an inheritance relationship. Lines 3 and 8 reflect this verification. The second conflict-free case is related to \(s\) being an individual. When the \(s\) of the two access authorizations are both an individual or the \(s\) of an ancestor authorization is an individual, there is absolutely no conflict. Lines 4, 6, 12, and 14 reflect this verification. Thus far, we have briefly introduced our authorization model and the conflict detection module for subsumption inference in particular. In the next section, based on the authorization model, we introduce the implicit authorization conflict caused by RIF inference and a conflict detection method that utilizes the algorithm shown in Fig. 3.

### 4. Implicit Authorization Conflict Caused by RIF Inference

#### 4.1 Introduction to RIF Rule

RIF\(^9\) is a standard rule language recommended by W3C and provides a framework for exchanging various heterogeneous rules over the Web. RIF has several dialects, such as RIF Core Dialect, RIF Basic Logic Dialect (BLD), and RIF Production Rule Dialect (PRD). RIF Core is closely related to Datalog, RIF BLD is more expressive in that it adds function symbols to RIF Core, and RIF PRD can express the kind of rules used by production rule engines, as in business rule systems. In this paper, we focus on the RIF Core dialect, which is indeed the basic core of the rule languages considered by RIF. The following defines RIF Core, which is represented in the Horn clause form, as is datalog. This RIF rule can be combined with RDF data, and the RIF atom in an RIF rule can be represented in the RDF triple pattern.

**Definition 3 (RIF rule):** An RIF rule \(R_i\) is a formula of the form

\[
R_i(B_1 \land R_i B_2 \land \ldots \land R_i B_j) \rightarrow R_i H,
\]

where the premises \(R_i B_j\) and the conclusion \(R_i H\) are RIF atoms. Assuming that an RIF rule is defined for RDF data, the RIF atom is an RDF triple pattern for the RDF data.

As an illustration, let us consider the RIF rule \(R_1\) given in Fig. 4(c) for the RDF data of Fig. 1. The rule infers that a person is taking part in a meeting if he is located in area A and time t2 has passed. In the rule, the RIF atoms are
Example 2: For the rule $R_i$, consider the simple authorization conflict condition. Let us first consider the case where $\text{access} \leftarrow \text{propagation} \leftarrow \text{for} \leftarrow \text{Ri Bj}$.

In our system, $R_i B_j$ and $R_i H$ are decomposed and stored in BT and HT tables. In Fig. 4(a)(b), the columns $s$, $p$, and $o$ store the RDF triple pattern in $R_i B_j$ and $R_i H$, the column label stores the graph label for the triple pattern, and the column hid in the BT table refers to the hid value in the HT table. For example, $R_1 B_1 = [s_1 \text{locatedIn}, \text{areaA}]$ and $R_1 B_2 = [\text{c1}, \text{time}, \text{t2}]$ are stored in the BT table and $R_1 H = [s_1 \text{do}, \text{meeting}]$ is stored in the HT table.

4.2 Authorization Conflict Conditions

In this subsection, we analyze the implicit authorization conflict conditions caused by RIF inference. Let us first consider the simple authorization conflict condition.

Example 2: For the rule $R_i$ in Fig. 4(c), let us consider the access authorizations in Fig. 2. Since $A_{hi}$ and $A_{new}$ allow access to $[\text{c1}, \text{time}, \text{t2}]$ and $[\text{o1}, \text{locatedIn}, \text{areaA}]$, the inferred triple $[\text{o1}, \text{do}, \text{meeting}]$ is surely accessible. However, let us consider the case where $A_{new}$ is deleted and a new access authorization, $A_{new}$, is added. At that time, access to $[\text{o1}, \text{do}, \text{meeting}]$ must be forbidden since $[\text{o1}, \text{locatedIn}, \text{areaA}]$ is not accessible.

Theorem 1 (simple conflict condition in RIF): Let us consider explicit authorization propagation for $R_i B_j$ and $R_i H$ of an RIF rule $R_i$ by access authorizations. (1) If the $(−)$ security sign is allocated to any $R_i B_j$ but $R_i H$ has the $(+)$ security sign, this constitutes a conflict. (2) If all $R_i B_j$ have the $(+)$ security sign but $R_i H$ has the $(−)$ security sign, this constitutes a conflict. (3) Let us consider another RIF rule, $R_k$, having the same conclusion as $R_i$. If the two conclusions $R_i H$ and $R_k H$ have opposite security signs by RIF inference, this constitutes a conflict.

Proof: The proof is straightforward. (1) Since access to a triple in the premises is not allowed, the RIF inference cannot start. Therefore, if the triple in the conclusion is accessible, this constitutes a conflict. (2) If all triples in the premises are accessible, the triple in the conclusion is also accessible. If access to the conclusion is not allowed, this constitutes a conflict. (3) Let us assume that the $(−)$ security sign is allocated to any $R_i B_j$. Then, $R_i H$ must have the $(−)$ security sign. If all $R_i B_j$ have the $(+)$ security sign, $R_i H$ must have the $(+)$ security sign. Since the two conclusions are the same triple pattern, this constitutes a conflict.

Next, let us consider the transitive authorization conflict condition. This is related to inference chains where a conclusion of an RIF rule exists in the premises of other RIF rules [11].

Theorem 2 (transitive conflict condition in RIF): Let us consider a transitive case where $R_i H$ in $R_i$ is related to any $R_i B_j$ in $R_k$. (1) If the $(−)$ security sign is allocated to any $R_i B_j$ but $R_k H$ has the $(+) security sign, this constitutes a conflict. (2) If all $R_i B_j$ have the $(+) security sign but $R_k H has the $(−)$ security sign, this can constitute a conflict.

Proof: (1) Since $R_k H$ has the $(+)$ security sign, all $R_i B_j$ must have the $(+)$ security sign by the simple authorization conflict condition. Since any $R_i B_j$ has the $(−)$ security sign, $R_i H$ must also have the $(−)$ security sign. Since $\text{sign}(R_i H) = \text{sign}(R_k H)$, this constitutes a conflict. (2) Since all $R_i B_j$ have the $(+)$ security sign, $R_k H$ has the $(+) security sign. If all $R_i B_j$ become to have the $(+)$ security sign by $R_i H$, $R_i H$ has the $(+)$ security sign by the simple authorization conflict condition. In this case, if $R_k H$ has already has the $(−)$ security sign by an explicit authorization propagation, this constitutes conflict.

The following example illustrates the transitive authorization conflict condition.

\[ R_i B_1 + \land R_i B_2 + \land \ldots \land R_i B_n + \land R_k H + \land R_i B_1 + \land R_i B_2 + \land \ldots \land R_i H + \land \ldots \land R_k B_1 + \rightarrow R_i H − \]

5. A Heuristic Approach for Efficient Authorization Conflict Detection

Implicit authorization propagation by an RIF rule can bring about a conflict in the interaction with RDF inference. As...
shown in Fig. 5, when a new access authorization is specified at module A, a simple or transitive conflict caused by module C can arise. However, we must also consider that, although no conflict is caused directly by RIF inference, the authorizations implicitly propagated by an RIF rule can be in conflict as a result of the RDF inference at module B. This is similar to the case where a new RIF rule is specified at module C. In this section, we suggest an authorization conflict detection algorithm that considers the interaction of RDF and RIF inferences. In particular, we consider the RDF subsumption inference presented in Sect. 3 and utilize the graph labeling-based authorization conflict detection algorithm given in Fig. 3.

5.1 When a New Access Authorization is Specified

In this subsection, we explain the algorithm given in Fig. 6. Let us denote a newly specified access authorization by \( A_{\text{new}} \) and the existing approved access authorizations by \( A_{\text{uk}} \) (\( k \in 1, 2, 3, \ldots, n \)). Before the function Method 2(\( A_{\text{new}} \)) is called, the function Method 1(\( A_{\text{new}} \)) shown in Fig. 3 is called. The function Method 1 judges whether there is an implicit authorization conflict caused by RDF inference between \( A_{\text{new}} \) and \( A_{\text{uk}} \). If there is no authorization conflict, the function Method 2 is called.

The first step in Method 2 is to assign a security sign to each \( R_i B_j \) in BT and each \( R_i H \) in HT according to \( A_{\text{new}} \) and \( A_{\text{uk}} \). This is executed by line 1. During this assignment process, it is clear that no conflict arises. This is because such conflicts have already been checked in Method 1. Figure 7(a) depicts assigning the security signs in Fig. 2 to the RIF rules in Fig. 4. ‘0’ indicates the security sign (−) and ‘1’ indicates the security sign (+). According to the new access authorization \( A_{\text{new}} \), the (−) security sign is assigned to the triple pattern \([x, \text{locatedIn}, \text{areaA}]\) and \([x, \text{locatedIn}, \text{areaB}]\). By explicit authorization propagation, \( A_{\text{u1}} \) assigns the (+) security sign to \([b1, \text{state}, \text{on}]\) and \([b1, \text{state}, \text{off}]\), \( A_{\text{u2}} \) assigns the (+) security sign to \([c1, \text{time}, \text{t2}]\), and \( A_{\text{u3}} \) assigns the (+) security sign to \([b1, \text{volume}, \text{max}]\) and \([b1, \text{volume}, \text{medium}]\). If no access authorization is specified for a triple pattern, the null value is assigned. The triple patterns \([x, \text{do}, \text{meeting}]\), \([x, \text{do}, \text{rest}]\), \([x, \text{do}, \text{walking}]\), and \([x, \text{do}, \text{standing}]\) have the null value in the sign column. As in our former study, the function Method 2 uses the graph label in BT and HT in order to efficiently assign the security sign according to the explicit authorization propagation.

The second step is to perform the implicit authorization propagation according to the existing approved RIF rules, \( R_i \). In line 2 of Method 2, according to Theorem 1, the case where all \( R_i B_j \) have the (+) security sign and the case where any \( R_i B_j \) has the (−) security sign are calculated. It is clear that, in the first case, there must be no \( R_i B_j \) that has the null sign value for the same hid value. See the BT’ table in Fig. 7(b). In line 3, if all \( R_i B_j \) have the (+) security sign, ‘1’ is obtained, but if any \( R_i B_j \) has the (−) security sign, ‘0’ is obtained. For example, in Fig. 7(b), ‘0’ is obtained for the hid values 1 and 3. Line 4 checks the authorization conflict, and lines 5 and 6 perform the implicit authorization propagation by RIF inference. See the HT’ table in Fig. 7(b). Line 7 sets up the transitive authorization propagation in Theorem 2; if a mismatch occurs in the assignment, this is the conflict in (3) in Theorem 1. The While block is iterated until no new authorization propagation is executed by line 6.

If there is no conflict in the While block, the access authorizations implicitly propagated by RIF inference can be in conflict with themselves, \( A_{\text{new}} \), and \( A_{\text{uk}} \) by RDF subsumption inference. For example, in Fig. 7(b), the vari-

Algorithm Method2

Input: A new access authorization \( A_{\text{u}} \)
Output: \( A_{\text{u}}'s \) conflict authorization set Conflict_Set

1 Assign a security sign to each triple in BT and HT according to explicit authorization propagation;

While (1)
2 \( BT' := \{ \text{select the rows having the same hid value of which all rows have the (+) sign value from BT} \} \cup \{ \text{select the rows having the same hid value of which any row has the (−) sign value from BT} \}; \)
3 Calculate the minimum sign value per the same hid value in BT’;
4 Compare the minimum sign values in BT’ with the sign value of each hid in HT. If there is a mismatch except the null value in HT, assign the hid values in HT to Conflict_Set, and return Conflict_Set;
5 \( HT' := \{ \text{select the rows having the null sign value from HT} \}; \)
6 Assign the minimum sign value of each hid in BT’ to the sign column in HT’; If no assignment occurs, return Conflict_Set;
7 Assign the security signs in HT’ to the related rows in BT and HT; If there is a mismatch, assign the hid values in HT’ to Conflict_Set and return Conflict_Set;
8 Add the hid values in HT’, of which the sign column does not have the null value, to implicitly_propagated_Au;
9 return Method 1(implicitly_propagated_Au);

Fig. 5 Authorization conflict caused by the interaction of RIF and RDF inferences.

Fig. 6 Authorization conflict detection algorithm for RIF inference.
able *implicitly_propagated_Au* includes <programA, [Sx, do, Sz], read, −, R> and can be in conflict with the access authorizations of Fig. 2. In order to detect the implicit authorization conflict caused by RDF inference, the function Method 1 is called again, as in line 9. If there is still no authorization conflict, Autnew is finally accepted. The above example finally includes the *Conflict_Set* = [Aut2, Aut4, Aut5, Aut6] according to the second iteration of Fig. 7(c).

5.2 When a New RIF Rule Is Specified

For a new RIF rule, the authorization conflict detection is similar to that described in the previous section. Let us denote a newly specified RIF rule by Rnew. The first step is to input Rnew into the BT and HT tables. The remaining steps follow lines 1 - 9 in the algorithm given in Fig. 6. If there is no authorization conflict, Rnew is accepted, otherwise it is rejected.

5.3 Several Considerations

In Sect. 5.1, we considered the null sign value for the RIF atoms that are not related to an access authorization. However, in real application domains, “denial takes precedence (DTP)” or “permission takes precedence (PTP)” will be considered for the default security sign [12]. In the case of DTP, the (−) sign is assigned as the default security sign. In the case of PTP, the (+) sign is assigned. If a default security policy is defined, the transitive authorization conflict condition does not arise. Only the simple authorization conflict condition arises. The reason is straightforward. If a security sign is always assigned to R, H in R, by the default security policy, only the simple authorization conflict condition occurs between R, H and R, Bj. Therefore, the transitive authorization conflict condition cannot arise.

6. Experiments

6.1 Experimental Setup

In this section, we compare the authorization conflict detection times between the instance-level detection method [3], [7] and our authorization-level method. As mentioned in Sect. 3.3, the instance-level method checks the authorization conflict by inferring all individuals with a security sign, and can be applied to detect an authorization conflict by RIF inference. To simulate the instance-level method, we used OWLIM [13], which is a native RDF repository and supports RDF and RIF inferences. We measured the execution times first for allocating security signs to all RDF triples, and then, for inferring the triples with a security sign.

The test data, access authorizations, and inference rules were simple, as shown in Fig. 8. We used them to show that

---

**Experimental factor** | **Value**
---|---
Number of individuals | \( I_0 \cdot 1 \leq k \leq 100,000 \)
Number of access authorizations | Aut0:
\(<\text{Dave}, [C0, P1, Sz], \text{read, } - R>\>
\(<\text{Dave}, [I_k, P1, Sz], \text{read, } + R>\>
\(1 \leq k \leq 100,000\)
Number of RIF rules | R1: \([Sx, \text{rdf} \text{ type}, C12] \rightarrow [Sx, \text{rdf} \text{ type}, C13] \)
R2: \([Sx, \text{rdf} \text{ type}, C13] \rightarrow [Sx, \text{rdf} \text{ type}, C14] \)
R3: \([Sx, \text{rdf} \text{ type}, C14] \rightarrow [Sx, \text{rdf} \text{ type}, C15] \)
R4: \([Sx, \text{rdf} \text{ type}, C15] \rightarrow [Sx, \text{rdf} \text{ type}, C16] \)
R5: \([Sx, \text{rdf} \text{ type}, C16] \rightarrow [Sx, \text{rdf} \text{ type}, C17] \)
R6: \([Sx, \text{rdf} \text{ type}, C17] \rightarrow [Sx, \text{rdf} \text{ type}, C18] \)
R7: \([Sx, \text{rdf} \text{ type}, C18] \rightarrow [Sx, \text{rdf} \text{ type}, C19] \)
R8: \([Sx, \text{rdf} \text{ type}, C19] \rightarrow [Sx, \text{rdf} \text{ type}, C20] \)
R9: \([Sx, \text{rdf} \text{ type}, C20] \rightarrow [Sx, \text{rdf} \text{ type}, C21] \)
R10: \([Sx, \text{rdf} \text{ type}, C21] \rightarrow [Sx, \text{rdf} \text{ type}, C22] \)

---

Fig. 7: Illustration of the conflict detection algorithm.

Fig. 8: Experimental configuration.
even in a simple situation, the difference between the two methods is significant. In the figure, C1, C2, ..., and C22 are subclasses, and \( I_k \) is the individual of C12. The number of individuals ranged from 100 to 100,000 to take large RDF data into consideration. In Fig. 8(b), for the test RIF rules, we considered the inference chain from C12 to C22. Simultaneously, this will bring about the RDF subClassOf inferences from each child class to its parent class.

The computer used in all experiments was equipped with an Intel(R)Core(TM)2 Quad CPU Q8400 2.66GHz, and 1GB of memory. The experiments were performed using an OWLIM standard edition and an MS SQL Server 2008 on Windows Server 2003; all codes were written in Java.

### 6.2 Experimental Results

First, Fig. 9(a) shows the authorization conflict detection time of the two methods according to the number of individuals. The number of access authorizations was set to 100, and the access authorizations, \( A_{i_0} \) (\( 1 \leq k \leq 100 \)) were specified against \( A_{i_0} \). The RIF rules \( R_1 \) and \( R_2 \) were used. The graph shows that our authorization-level method has a shorter conflict detection time than the instance-level method according to the number of individuals. In addition, if many more access authorizations and RIF rules were to be considered, the difference between the two methods would be more significant.

Second, Fig. 9(b) shows the authorization conflict detection time according to the number of access authorizations. The number of access authorizations ranged from 100 to 100,000, and the number of individuals was set to 10,000. For RIF rules, \( R_1 \) and \( R_2 \) were used. The results graph shows that the conflict detection time of the instance-level method increases with the number of access authorizations. This is because the instance-level method assigns security signs to all individuals according to access authorizations. If the number of access authorizations were large, the difference between the two methods would be more significant.

Third, Fig. 9(c) shows the authorization conflict detection time according to the number of RIF rules. The ten RIF rules in Fig. 8(b) were used, and the number of individuals was set to 10,000. For access authorizations, \( A_{i_0} \) and \( A_{i_1} \) were used. The results graph shows that the authorization-level method performs better than the instance-level method. Since Jain and Farkas’ method performs all inferences for all individuals to detect authorization conflicts, it requires a much longer detection time. However, the authorization-level method verifies only access authorizations without inferring individuals.

### 7. Conclusions and Future Work

In this paper, we analyzed the authorization conflict problem caused by RIF inference. In addition, we introduced an efficient authorization conflict detection method using graph labeling techniques. We verified experimentally that our method performs better than the instance-level authorization conflict detection method [3], [7]. However, our heuristic method was designed for the subsumption inference type only, and efficient detection modules for other inference types need to be investigated.

In the previous work [14], we suggested a non-heuristic approach for efficient authorization conflict detection for RDF inference. Contrary to the instance-level method, which infers all individuals, it infers only individuals that are specified in access authorizations and RDFS data. For example, in the access authorizations in Fig. 2, only the in-
individuals c1 and b1 and the classes in Fig. 1 are inferred. The other individuals are excluded. We named this method hybrid authorization conflict detection. The basic idea of the hybrid approach is based on the fact that no conflict between individuals is caused by any RDF inference in our authorization model. An access authorization for an individual has a conflict only with access authorizations for classes. We already verified this observation through Theorem 1 in the previous work [6]. Therefore, in our authorization model, without inferring all individuals, it is sufficient to infer only the individuals that are specified in access authorizations and RDFS data. In the previous work [14], we showed that the hybrid approach has a reasonable conflict detection time as compared with the authorization-level method. It also has the merits of completeness and secureness, as does Jain and Farkas’ method. Our future work involves applying this hybrid approach to the RIF inference. In the approach, to detect an authorization conflict, only the individuals specified in access authorizations and RIF rules need to be inferred. Finally, our future work involves extending our authorization model and authorization conflict detection methods to OWL data, which can be combined with RIF rules.

Acknowledgments

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

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(No. NRF-2010-0021059).

References


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 the Telecommunication R&D center of the Samsung Electronics Company from 2005 to 2006. From 2006 to 2009, he was a researching professor of Computer Science at Sogang University, and since 2009, he has been an assistant professor of Information & Communication at Seoul University. His major research areas are database, database security, and open-source software.