2003－DBS－131　(8)
2003／7／16

Translating Authorizations of XML Documents based on Content and Structure of Documents

内容および構造に基づく XML 文書のアクセス権の変換について

Somchai Chatvichienchai, Mizuho Iwaihara, Yahiko Kambayashi

Department of Social Informatics, Kyoto University
Yoshida Sakyo Kyoto 606-8501 Japan
somchai@db.soc.i.kyoto-u.ac.jp, iwaihara@i.kyoto-u.ac.jp, yahiko@i.kyoto-u.ac.jp

Abstract: Since authorization rules (authorizations, for short) use path expressions of XPath for locating data in documents, authorization definition is related to the document format. However, the format of XML documents tends to change by various reasons such as application extension and information exchange between organizations. Therefore, authorizations must be revised whenever they become incompatible with a new format of the document. We define sufficient conditions for schema and document transformations that allow transforming authorizations without access to individual source and target XML documents. We propose an algorithm that computes authorizations for a target DTD instance from given RBAC authorizations of a source DTD instance and schema mapping information under certain schema and document transformations while preserving the authorization requirement of the source DTD instance.

1. Introduction

As XML [14] is a popular format for presenting and exchanging data on the web, it plays a crucial role in the new Internet applications ranging from e-commerce to digital government. Whenever XML data is read and modified by multiple users, access control of the clients' access to XML data is an important topic. Most contributions [1, 9, 2, 6, 4] use path expressions of XPath [12] for locating nodes in XML documents, access authorizations of users can be defined based on the content of the document itself or on the structure of the XML document. *Content-based authorizations* for XML documents are defined as regulating accesses by not only the structure of documents, but also the contents of the documents. An essential feature of content-based authorizations is *value-dependent* meaning that an authorization locates data in an XML document by a path expression, which contains predicates comparing values of elements or attributes. However, the structures of XML documents tend to change over time by various reasons, such as application expansion, and data exchange among organizations using different formats. Whenever authorizations become incompatible with a new structure of XML documents, the security manager needs to transform authorizations for source documents into those for target documents.

To the best of our knowledge, our previous work [3] is the first that has discussed the problem of transforming authorizations. In [3] we have proposed two authorization transformation algorithms. The first is an algorithm that transforms authorizations of a source XML document into those for a target XML document by using document tree mapping. The second is an algorithm that transforms value-independent authorizations for a set of documents conforming to a source DTD to those for a set of documents conforming to a target DTD by using schema-tree mapping information. Schema-tree mapping is a mapping that describes unambiguously which nodes of source DTD tree correspond to which nodes in target DTD tree. Schema-tree mapping

can be obtained from schema matching tools such as Cupid [10], COMA[5] and Xtra [11]. Note that the second algorithm focuses on the case when a path expression locates all instances of an element or attribute. However, our previous works have not discussed the problem of transforming content-based authorizations by schema-tree mapping.

The objective of this paper is to propose a novel algorithm that computes content-based authorizations for target DTD instances from given content-based authorizations for source DTD instances by using schema-tree mapping while preserving access constraints of source DTD instances. In this paper, we introduce a new paradigm of technical challenge in applying unordered tree inclusion in authorization transformation. The key idea of authorization transformation is that we represent path expressions of authorizations by tree patterns. We transform a tree pattern of a source XML document into the corresponding tree pattern for a target XML document by using schema-tree mapping. One important problem is to identify the conditions of schema and data transformation that allows tree pattern transformation such that transformed tree pattern locates target document nodes derived from source document nodes located by the source tree pattern. Our work employs the RBAC model for XML documents proposed by Hitchens et al [6] because the RBAC model provides a very flexible set of mechanisms for managing the access control of a complex system with many users, objects and applications.

The rest of the paper is organized as follows. Section 2 gives basic concepts of XML documents and DTDs. Section 3 describes the RBAC for XML documents. Section 4 discusses sufficient conditions for schema and document transformations. Section 5 presents content-based authorization transformation. Finally, Section 6 presents our conclusions and future work.

2. XML Documents and DTDs

An XML document consists of three parts: an XML declaration, a Document Type Definition (DTD), and an XML document instance. An XML document instance is a tagged document that is composed of a sequence of nested elements, each delimited by a pair of start and end tags or by an empty tag. An element can have attributes attached to it. XML document instances can be classified into two categories: *well-formed* and *valid*. An XML document instance is well-formed if it obeys the syntax of XML. A well-formed document is valid if it conforms to a proper DTD. A DTD contains a formal definition of a particular type of XML documents. In this paper, we assume that a DTD is an external file. Our algorithm can be easily adapted for XML schema [14]. We abstract DTDs as labelled trees that are defined as follows.

Definition 2.1 (Schema Trees): A schema tree is a finite node labelled and edge labelled tree $S = (V, E, Nm, r, lbl, C, f_C)$, where V is a set of vertices (nodes) representing elements and attributes of the DTD, $E \subseteq V \times V$ is a set of edges, Nm is a set of element and attribute names, r is the root of the DTD, lbl is a labelling function assigning a name from Nm to a node in V, C is the set {+, *, ?} called cardinality, and f_C is a labelling function assigning a label from $C \cup \{\varepsilon\}$ to an edge in E. □

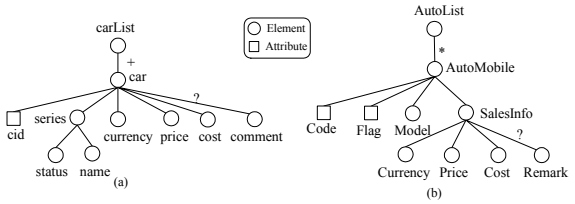

Fig.1 (a) A source schema tree and (b) a target schema tree

Figure 1 depicts an example of schema trees for source and target DTDs. It should be clear that edge labels indicate the occurrence of sub-element to be expected: +, *, ? meaning "one or more", "zero or more", and "zero or one", respectively. An edge with no label indicates that the occurrence of sub-element is "exactly one". We call schema tree of a source DTD a *source schema tree* and call schema tree of a target DTD a *target schema tree*. We define relationship between nodes of source and target schema trees by a mapping that unambiguously describes which nodes in the source schema tree correspond to which nodes in the target schema tree.

Let S and T be source and target schema trees, respectively. Let V be the set of schema nodes of

S, V' be the set of schema nodes of T, V_D be the set of schema nodes of S that are not transferred to T, and V_A be the set of schema nodes that are newly added to T. We call $V - V_D$ the set of *source schema nodes*. We also call $V' - V_A$ the set of *target schema nodes*. For V_A, the security manager is requested to define authorization for each schema node in V_A. Otherwise, authorizations of all schema nodes of V_A are defined by the default authorization. For V_D, the security manager is requested to revise definition of the authorizations that are based on values of schema nodes in V_D.

Definition 2.2 (Schema-Tree Mapping): Let S and T be source and target schema trees, respectively. Let V_S be the set of source schema nodes of S, and V_T be the set of target schema nodes of T. $smap: V_S \rightarrow V_T$ is a total and one-to-one mapping from the nodes in V_S to the nodes in V_T. □

Suppose that v is a source schema node of S, v' is a target schema node of T, s is an instance of S, and t is an instance of T. Remember that if $smap(v) = v'$, then we assume that each instance of v in s is transferred to be an instance of v' in t. Namely, there is no information loss from transferring instances of a source schema node to the corresponding target schema node. We abstract XML document instances as labelled trees (called *instance trees*) that are defined as follows.

Definition 2.3 (Instance Trees): An instance tree t is a finite node labelled tree represented by a tuple of $(V, E, Nm, Txt, r, lbl, val)$, where V is a set of nodes of t, $E \subseteq V \times V$ is a set of edges, Nm is a set of element and attribute names, Txt is a set of values, r is the root of the XML document instance, lbl is a labelling function assigning a name from Nm to a node, and val is a labeling function assigning a value from $Txt \cup \{\varepsilon\}$ to a node. □

We call an instance tree for a source document instance a *source instance tree* and call an instance tree for a target document instance a *target instance tree*. We call a node of instance tree an *instance node*.

3. RBAC for XML Documents

An authorization indicates the right to perform a specific operation on a particular data object. Authorization in the model can be fine-grained (e.g. at the element level) or coarse-grained (e.g. at level of entire document). Authorizations are then grouped together within the roles themselves.

Definition 3.1 (Authorizations): An authorization is a 7-tuple of the following form: *<pname, target, path, action, sign, prop, priority>*, where *pname* is a permission name, *target* is a list of XML documents or a DTD or a schema, *path* is an optional path expression of XPath identifying elements to which the authorization apply, *action* ∈ {read, write, create, delete, all}, *sign* ∈ {'+', '-'} specifies whether the authorization grants ('+') or disallows ('-') access, *prop* ∈ {*local, recursive*}, and *priority* is an optional value specifying the priority of the authorization. The default value of *priority* is 0. The highest value of *priority* is 99. □

If *target* is a DTD (or a schema) then the authorization applies to all instances of the DTD (or the schema, respectively). Otherwise, the authorization applies to a specific XML document. If the *prop* is *local* then the authorization only applies to the attributes, links and data of the specified elements (as defined by *path*). Otherwise, the authorization applies to the specified elements, their direct and indirect sub-elements and attributes. Conflict resolution of the model is based on *priority* of authorizations. If there is a conflict between a set of authorizations on an element then the authorizations with the highest priority are selected. If authorizations have the same priority, negative authorizations override positive authorizations.

Definition 3.2 (Roles): A role is a 3-tuple of the form: (*role_name, child_roles, pnames*), where *role_name* is a role name, *child_role* is an optional list of child roles, and *pnames* is an optional list of permission names. Note that the parent role inherits access privileges from its child roles. □

Example 1: Consider the following authorizations and roles defined for all instances of *carList.dtd* depicted in Fig. 1(a). Role *roleClient* is permitted to read car information except *cost* of all cars and *price* of the cars that are classified as secret.

Authorizations:

<pn1, carList.dtd, carList, read, +, local, 0>
<pn2, carList.dtd, car/cost, read, -, recursive, 1>
<pn3, carList.dtd, car[series/status="Secret"]
/price, read, -, recursive, 1>

Roles: *(roleClient, , {pn1, pn2, pn3})*

4. Sufficient Conditions for Schema and Document Transformations

We first give the definition of instance-tree mapping which identifies the relationship between nodes of two instance trees.

Let s and t be source and instance trees, respectively. Let V be the set of instance nodes of s, V' be the set of instance nodes of t, V_d be the set of instance nodes of s that are not transferred to t, and V_a be the set of schema nodes that are newly added to t. We call $V - V_s$ the set of *source instance nodes*. We also call $V' - V_a$ a the set of *target instance nodes*.

Definition 4.1 (Instance-Tree Mapping): Let s and t be source and target instance-trees, respectively. Let V_s be the set of source instance nodes of s, and V_t be the set of target instance nodes of t. *imap*: $V_s \rightarrow V_t$ is a total and one-to-one mapping from V_s to V_t. □

We represent path expressions of authorizations by tree patterns that are defined as follows.

Definition 4.2 (Tree Patterns): A tree pattern p is an unordered tree represented by a tuple of (V, E, Nm, Txt, r, lbl, val, opr), where V is a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges, Nm is a set of element and attribute names, Txt is a set of values, r is a node that forms the root of p, lbl is a labelling function assigning a name from Nm to a node, val is a labelling function assigning a value from $Txt \cup \{\varepsilon\}$ to a node, and opr is a labelling function assigning a symbol from {'=', '≠', '≥', '>', '≤', '<'} to a node. We represent the return node with a double-line circle. We present a child edge with a single line and present a descendant edge with a double line. Due to space constraints and the complexity of XPath, we assume that each terminal node of tree pattern is an element or attribute with a unique name. □

Example 2: Figure 2(a) depicts a tree pattern of path expression *car[price>"10000"]/series* and Fig. 2(b) depicts a tree pattern of path expression *car[//status="Secret"]/price*.

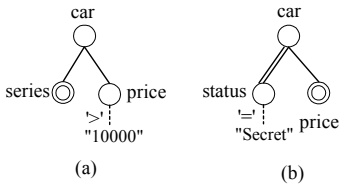

Fig. 2. Sample tree patterns

If an XML instance tree has nodes that are satisfied by a tree pattern, all nodes of the tree pattern must have a corresponding matching node in the XML instance tree, and each predecessor-successor relationship of nodes in the tree pattern should be guaranteed by those in the XML instance tree. This is also known as the tree embedding [8]. Our definition of tree embedding is inspired by the unordered path inclusion problem defined by Kilpelainen.

Definition 4.3 (Tree Embedding): Let $t = (V_t, E_t, Nm_t, Txt_t, r_t, lbl_t, val_t)$ be an instance tree, and $p = (V_p, E_p, Nm_p, Txt_p, r_p, lbl_p, val_p, opr_p)$ be a tree pattern. *emb*: $V_p \rightarrow V_t$ is an *embedding* from p into t if and only if the following conditions hold:

(1) *emb* is a function: $x \in V_p \Rightarrow emb(x) \in V_t$,

(2) *emb* is name preserving: for each $x \in V_p$, $lbl_p(x) = lbl_t(emb(x))$,

(3) *emb* is ancestor-descendant preserving:

- $(x, y) \in E_p$ is a child edge and y is a child of $x \Rightarrow (emb(x), emb(y)) \in E_t$ and $emb(y)$ is a child of $emb(x)$, and
- $(x, y) \in E_p$ is a descendant edge and y is a descendant of $x \Rightarrow emb(y)$ is a descendant of $emb(x)$,

(4) *emb* is content filtering: for each $x \in V_p$ where $emb(x) \in V_t$ is a terminal node and $opr_p(x)$ is not $\{\varepsilon\}$, the Boolean expression: $val_p(x)\ opr_p(x)\ val_t(emb(x))$ is true. □

Let u be a node of p. We call the instance node $emb(u)$ a *matched node* of u.

Definition 4.4 (Images of Tree Patterns): An image of p in instance tree t by an embedding *emb* is an unordered tree w where (1) the root node of w is a matched root node of p, (2) w contains matched nodes of p, and (3) all matched terminal nodes of p are terminal nodes of w. □

Note that for a fixed tree pattern p and a fixed instance tree t, several embedding trees may

exist, and several embeddings may lead to the same embedding tree.

Example 3: Figure 3 depicts embedding of tree pattern of *car[//status="Secret"]/price* to an instance tree. Figure 4 shows an image of this tree pattern in the instance tree of Fig. 3.

When a source document instance is transformed to a target document, we need to transform path expression of an authorization for the source into a path expression for the target, where tree patterns are preserved by the tree-instance mapping. The correspondence of tree patterns of a target instance tree with tree patterns of a source instance tree is defined as follows.

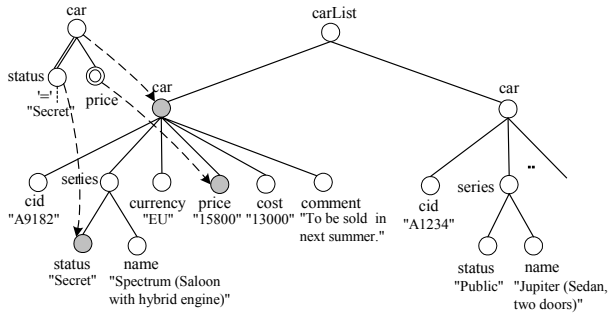

Fig. 3. Embedding of tree patterns

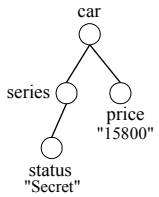

Fig. 4. An image of the tree pattern in instance tree of Fig 3

Definition 4.5 (Corresponding Tree Patterns): Let s and t be source and target instance trees, respectively. Let *imap* be the instance-tree mapping from source nodes of s to target nodes of t. Let p be a tree pattern in s, p' be a tree pattern in t, $nodeset(s, p)$ denote the set of source instance nodes of s satisfied by p, and $nodeset(t, p')$ denote the set of target instance nodes of t satisfied by p'. Tree pattern p' *corresponds* to p if and only if for each $v \in nodeset(s, p)$, there exists $imap(v) \in nodeset(t, p')$. □

In many occasions, a target schema is evolved from a source schema by (1) removing / adding elements and attributes, (2) changing names and types of elements and attributes, and (3) folding / unfolding elements. We observe that it is possible to find a tree pattern of a target document instance corresponding to given tree pattern of a source document instance by schema-tree mapping under the following schema and document transformations.

Definition 4.6 (Ancestor-Descendant Relationship Preserving Schema Transformation): Let S be a source schema tree, and T be the target schema tree transformed from S by a schema transformation F. Let V_S be the set of source schema nodes of S, V_D be the set of non-transfer nodes of S, V_T be the set of target schema nodes of T, and *smap* be schema-tree mapping from V_S to V_T. F is an *ancestor-descendant relationship preserving schema transformation* (*APST*) if and only if the following conditions hold:

- For each x, $y \in V_S$ and for each $smap(x)$, $smap(y) \in V_T$ if y is a descendant of x then $smap(y)$ is a descendant of $smap(x)$; and
- For each $x \in V_D$ if x is a non-terminal node then the occurrence of x under its parent is one. □

The first condition allows computing the corresponding tree pattern straightforwardly. The second condition restricts unfolding child nodes of the same type. This kind of unfolding may make it impossible to distinguish the corresponding target instance nodes by using only knowledge of schema-tree mapping. From now on, we denote $APST(S)$ as a target schema tree transformed from a source schema tree S by *APST*.

Definition 4.7 (Values and Ancestor-Descendant Relationship Preserving Document Instance Transformation): Let $s = (V, E, Nm, Txt, r, lbl, val)$ be a source instance tree, $t = (V', E', Nm', Txt', r', lbl', val')$ be an instance tree transformed from s by document instance transformation D. Let V_s be the set of source instance nodes of s, and V_t be the set of target instance nodes of t, and *imap* be the instance-tree mapping from V_s to V_t. *D is value and ancestor-descendant relationship preserving document instance transformation* (*VAPDT*) if and only if the following conditions hold:

- Value preserving: for each $x \in V_s$, $val(x) = val(imap(x))$, and
- Ancestor-descendant relation preserving: for each $x, y \in V_s$ and y is a descendant of $x \Rightarrow imap(y)$ is a descendant of $imap(x)$. □

From now on, we denote $VAPDT(s)$ as a target instance tree transformed from a source instance tree s by $VAPDT$.

Example 4: Figure 5 depicts a sample of $VAPDT$. Nodes a, b, c, d, and f of source instance tree s are mapped by $imap$ to nodes a', b', c', d', and f' of target instance tree t, respectively. Node e of s is not transferred to t while nodes x, y and z are newly added nodes for t. Notice that ancestor-descendant relationships among nodes a, b, c, d, and f of source instance tree s are preserved in t.

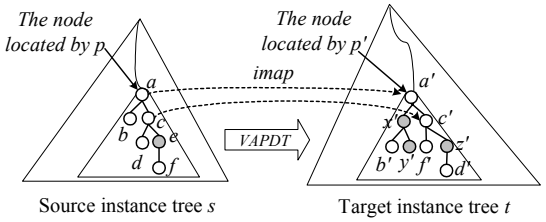

Fig. 5. A sample of *VAPDT*

It is important to note that the first condition of Definition 4.6 is not sufficient to forbid swapping of child nodes of the same type under different parent nodes of an instance tree. For example, suppose that nodes b_1, c_1 are child nodes of node a_1 and nodes b_2, c_2 are child nodes of node a_2. Conditions of Definition 4.6 cannot forbid swapping between child nodes b_1 and b_2. This swapping makes it impossible to compute the corresponding tree patterns of target instance tree by using only schema-tree mapping. Therefore, we need Definition 4.7 to forbid this kind of swapping.

Definition 4.8 (Matching Subtrees): Let s be a source instance tree, t be a target instance tree, and $imap$ be the instance-tree mapping from source nodes of s to target nodes of t. Let x be a subtree rooted by node v of s, and y be a subtree rooted by v' of t. Subtree y is a matching subtree of x if and only if (1) $v' = imap(v)$ and (2) each source instance node in subtree x is mapped by $imap$ to a target instance node of subtree y. □

Lemma 1 (Matching Subtrees Located by Corresponding Tree Patterns): Let S be a source schema tree, T be $APST(S)$, s be an instance tree of S, and t be $VAPDT(s)$ that is an instance of T, and $imap$ be the instance-tree mapping from the set of source nodes of s to the set of target nodes of t. Let p that is a tree pattern locating node v of s, and p' be a tree pattern locating node v' of t, and p' correspond to p. Then subtree y whose root node located by p' matches with subtree x whose root node located by p. □

5. Content-Based Authorization Transformation

In this section, we present two algorithms. The first is an algorithm that computes a tree pattern of a target instance tree that corresponds to a given tree pattern of a source tree instance. The second is an algorithm that transforms content-based authorizations of a source DTD instance into authorizations for a target DTD instance. From now on, we use the term "authorizations" to refer to content-based authorizations. We call a path that starts from an ancestor node of an instance tree going down to a descendant node *a linear path*. We define a linear path of the target instance tree that matches with a linear path of the source instance tree as follows.

Definition 5.1 (Matching Linear Paths): Let S be a source schema tree, T be a target schema tree, V_S be the set of source schema nodes of S, V_T be the set of target schema nodes of T, and $smap$ be the schema-tree mapping from V_S to V_T. Let $p = (V, E, Nm, Txt, r, lbl, val, opr)$ be a linear path for instances of S, $p' = (V', E', Nm'_p, Txt', r', lbl', val', opr')$ be a linear path for instances of T, and v and v' be terminal nodes of p and p', respectively. Tree pattern p' for instances of T is a *matching linear path* of p for instances of S if and only if the following conditions hold:

- $r' = smap(r)$, $v' = smap(v)$, $val(v) = val(smap(v))$ and $opr(v) = opr(smap(v))$; and
- for each $x, y \in V_S$, $smap(x)$, $smap(y) \in V_T$, y is a descendant of x, $smap(y)$ is a descendant of $smap(x)$. □

Note that a matching linear path p' should be verified by the security manager since sometimes the semantics of information located by p' may be different from that located by p.

Lemma 2 (Property of Matching Linear Paths under *APST* and *VAPDT*): Let S be a source schema tree, T be $APST(S)$, s be an instance tree of S, and t be $VAPDT(s)$ that is an instance of T. For p' of t that is a matching linear path of p of s, p' corresponds to p. □

Example 5: Linear paths *a'/x'/b'*, *a'/c'/z'/d'* and *a'/c'/f'* for target instance tree t of Fig.5 match with *a/b*, *a/c/d* and *a/c/e/f* for source instance tree s, respectively.

We present the *TreePatternTrans* algorithm (depicted in Fig. 6) that transforms a tree pattern for a source instance tree s of source schema S into the corresponding tree pattern for $VAPDT(s)$ that is an instance of $APST(S)$. We assume that there is no deleted terminal node in a given tree patterns since tree pattern transformation of this case needs intervention by the security manager.

Theorem 1 (Corresponding Tree Pattern Computation): Let S be a source schema tree, T be $APST(S)$, and *smap* be a schema-tree mapping from the set of source schema nodes of S to the set of target schema nodes of T, and p be a tree pattern of a source instance of S. Then, for a target instance tree of T that is transformed from a source instance tree of S by *VAPDT*, the output tree pattern p' computed by *TreePatternTrans* corresponds to p. □

We now present the *AuthTrans* algorithm depicted in Fig. 7 for transforming authorizations for a source instance s of source schema S into authorizations for $VAPDT(s)$ which is an instance of $APST(S)$. In Fig. 7, the closed-policy [7] is used for defining the default authorization. Therefore, we define negative authorizations with highest priority value for newly added schema elements of target DTD. However, our algorithm can be easily adapted to the case when the default authorization is based on other policies. As depicted in Fig.7, *AuthTrans* can transform authorizations for all source instances of a source schema into those for all instances of a target schema by setting *new-target* to be identification of the target schema. We use the following example for explaining how *AuthTrans* performs authorization transformation.

TreePatternTrans (*S, T, smap, p, p', result*)
Input:
(1) Source schema tree *S*,
(2) target schema tree *T* transformed from *S* by *APST*,
(3) schema tree mapping *smap* from the set of source schema nodes of *S* into the set of target schema nodes of *T*, and
(4) input tree pattern *p* for a source instance of *S*.
Output:
(1) The output tree pattern *p'* for *T*'s instance that is transformed from the instance of *S* by *VAPDT*, and
(2) *result* = *NO* if we cannot find tree pattern that corresponds to *p*. Otherwise, *result* = *YES*.
Process:
- Change *p* into *p''* by expanding all descendant edges into child edges.
- Suppose that *r* is the root node of *p''*, and $v_1, v_2, .., v_m$ are terminal nodes of *p''*.
- *result* = *NO*.
- Decompose *p''* into simple paths $x_1, x_2, .., x_m$ where x_i $(1 \le i \le m)$ is a simple path from *r* to v_i.
- **If** there is a matching simple path of *T* for each x_i **then**
  -- Let y_i $(1 \le i \le m)$ be the matching simple path of *T* for x_i.
  -- Set value and operator symbol of terminal node of y_i $(1 \le i \le m)$ by value and symbol of terminal node of x_i, respectively.
  -- Combine $y_1, y_2, .., y_m$ into tree pattern *p'*.
  -- Mark the return node of *p'* that is the matched return node of *p*.
  -- *result* = *YES*.
return *result*, *p'*

Fig. 6. The *TreePatternTrans* algorithm

AuthTrans(S, T, V_A, smap, new-target, AUTH, AUTH')
Input:
(1) Source schema tree *S*,
(2) target schema tree *T* transformed from *S* by *APST*,
(3) a set V_A of newly added schema element of *T*,
(4) schema-tree mapping *smap* from the set source schema nodes of *S* to the set of target nodes of *T*,
(5) *new-target* that is the new target id for transformed authorizations, and
(6) a set *AUTH* of authorization for an instance of *S*.
Output: The set *AUTH'* of authorizations for target instance of *T* that is transformed from the source instance of *S* by *VAPDT*.
Process:
- Set *AUTH'* to the empty set.
- **Do** the following steps until *AUTH* become empty
  -- Get authorization a_i from *AUTH*.
  -- *result* = *NO*.
  -- Let p_i be a tree pattern of path expression of a_i, and v_i be the return node of p_i.
  -- **If** there exists no $smap(v_i)$ in *T* then
    --- Create new authorizations with the same operation, sign, priority and type as those of a_i while path expressions of the new authorizations locate child nodes of v_i.
    ---- Set type of the new authorization to local if its path expression locates a terminal node.
    ---- Add the new authorizations to *AUTH*.
  -- **else**
    --- Call *TreePatternTrans* (p_i, *S, T, smap, result,* p_i').
    --- **If** *result* = *YES* then
      ---- Create a new authorization a'_i with the same operation, sign, priority and type as those of a_i. Set path expression of a'_i to be the path expression represented by p_i'.
      ---- Set *target* of a'_i to *new-target*. Add a'_i to *AUTH'*.
  -- Remove a_i from *R*.
- **End Do**
- **For** each v_k in V_A do the followings:
  -- Generate a permission rule a'_k that has *new-target* id, all operations, negative sign, local type, the highest priority and path expression locating v_k.
  -- Add a'_k to *AUTH'*.
- **End For**
return *AUTH'*

Fig. 7. The *AuthTrans* algorithm

Theorem 2 (Authorization Preservation): Let S be a source schema tree, T be $APST(S)$, V_A be the set of newly added nodes of T, *smap* be a schema-tree mapping from the set of source schema nodes of S to the set of target schema nodes of T. Let *new-target* be a target identification for transformed authorizations, and $AUTH$ be a set of authorizations for an instance of S. Given S, T, V_A, *smap* and $AUTH$, the set $AUTH'$ of authorizations (computed by *AuthTrans* for an instance of T derived from the source instance by *VAPDT*) *preserves AUTH*. □

6. Conclusions and Future Work

Content-based authorization is crucial for various applications since it can provide data access control that matches with requirements of applications. In order to perform content-based authorization transformation, we have proposed an algorithm that transforms tree pattern representing path expression of a source authorization into the corresponding tree pattern of the target DTD instance by using schema-tree mapping under certain schema and data transformations. This algorithm is based on the paradigm of unordered tree inclusion. Based on tree pattern transformation algorithm, we have proposed an algorithm that automatically computes authorizations for a target DTD instance from given authorizations of a source DTD instance. The goal of authorization transformation is that authorizations for the target DTD instance preserve the same access restriction of authorizations for the source DTD instance. Our algorithm can be easily adapted to existing XML access control models.

References

[1] E. Bertino, S. Castano, S. Ferrari and M. Mesiti. "Specifying and Enforcing Access Control Policies for XML Document Sources," World Wide Web, Baltzer Science Publishers, Netherlands, vol. 3, no. 3, 2000.

[2] E. Bertino, S.Castano, E.Ferrari, "On specifying security policies for web documents with an XML-based language", In Proc. of the Sixth ACM Symposium on Access control models and technologies, May 2001

[3] S. Chatvichienchai, M. Iwaihara, Y. Kambayashi, "Towards Translating Authorizations for Transformed XML Documents," in Proc. of the 3rd Int. Conf. on Web Information Systems Engineering (WISE 2002), Singapore, pp.291-300, Dec 2002.

[4] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati. "A Fine-Grained Access Control System for XML Documents", ACM Transaction on Information and System Security, Vol. 5, No. 2, pp. 169–202, May 2002.

[5] H.H. Do, E. Rahm: "COMA – A System for Flexible Combination of Schema Matching Approach," In Proc. Of VLDB 2002, Hong Kong, pp. 610-621, Aug 2002.

[6] M. Hitchens and V. Varadharajan, "RBAC for XML Document Stores" Information and Communications Security, Third International Conference, ICICS 2001, Xian, China, November 13-16, 2001.

[7] S. Jajodia, P. Samarati, V. S. Subrahmanian, E. Bertino . "A unified framework for enforcing multiple access control policies," in Proc. of the 1997 ACM SIGMOD Int. Conf. on Management of data, Arizona, pp.474-485, 1997.

[8] P. Kilpelainen and H. Mannila. "Ordered and unordered tree inclusion" Siam Journal on Computing, pp.340-356, 1995.

[9] M. Kudo and S. Hada. "XML Document Security based on Provisional Authorization". Proceedings of the 7th ACM conference on Computer and Communications Security, pp. 87-96, Athens Greece, November 2000.

[10] J. Madhavan, P.A. Bernstein and E. Rahm. "Generic Schema Matching with Cupid," in Proc. of the 27th VLDB Conference, Roma, Italy, pp.49-58, 2001.

[11] H. Su, H. Kuno, E.A. Rundensteiner, "Automating the Transformation of XML Documents" Advances in Web-Age Information Management, Second International Conference WIDM 2001: 68-75, July 9-11, 2001.

[12] W3C (1999). XML Path Language (XPath) Version 1.0. Available at http://www.w3c.org/TR/xpath (November 1999).

[13] W3C (2000). Extensible Markup Language (XML) 1.0 (Second Edition). Available at http://www.w3c.org/TR/REC-xml (October 2000).

[14] W3C (2001). XML Schema. Available at http://www.w3c.org/XML/Schema, 2001.