Translating Access Authorizations for Transformed XML Documents

Somchai Chatvichienchai, Mizuho Iwaihara, and Yahiko Kambayashi

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

Abstract. XML access control models proposed in the literature enforce access restrictions directly on the structure and content of an XML document. Therefore, the authorization must be revised whenever the structure of an XML document is changed. In this paper we present an approach that translates the authorizations for the transformed XML document. We focus on the case where schemas of the source and transformed documents represent the same concept. This approach is strongly based on schema mapping information between the source and transformed XML schemas.

1. Introduction

As XML [3] emerges as an increasingly popular format for representation and exchange of data, it will lead to web data sharing and data integration. Therefore, it becomes critical to define and enforce access restrictions on XML documents to ensure that only authorized users can access to the information. In recent years, contributions [2, 6, 8] have been made to XML access control models. These models enforce access restrictions directly on the structure and content of XML documents. In this way, information in XML format can be protected at a finer level of granularity (e.g., the element level) than the whole document. Each XML document is associated with a set of authorizations specifying access rights of users on information within the document. An object in the authorization is described by path expression identifying an element or attribute within the document. However, the structure of XML documents tend to change over time for a multitude of reasons, for example to correct design errors in the schema, to allow expansion of the application scope over time, or to account for the merging of several businesses into one. When an XML document is transformed to conform to a new schema, the associated authorizations must be translated for the transformed document. However, the translation of authorizations is a complicated task since its scope covers XML data model, schema matching, XML access control model, and application requirements. In general, the schema matching is a laborious manual work. Fortunately, recent contributions have been made in the area of schema matching and document transformation. Xtra [10] provides a set of

R. Cicchetti et al. (Eds.): DEXA 2002, LNCS 2453, pp. 290–299, 2002.

schema transformation operations that establish semantic relationships between two XML document schemas. Xtra also offers an algorithm that discovers XSLT [5] script to transform the source XML document into the target XML document. TranScm [9] examines and finds similarities/differences between the source and target schemas. This is done using a rule-based method that defines a possible common matching between two schema components, and provides means for translating an instance of the first to an instance of the second. These works can provide some schema mapping information needed for authorization translation.

The objective of this paper is to present an approach that translates authorizations of the XML document. To the best of our knowledge, no previous study has addressed authorization translation for XML documents. Our work focuses on the case where schemas of the source and transformed documents represent the same concept. For an XML document that is not associated with a DTD, we may obtain its DTD by applying DTD generating functions of existing XML document processing tools [1, 7]. The goal of authorization translation is that authorizations of the transformed XML document must enforce the same access restrictions as provided by the authorizations of the source XML document. This paper also indicates the limitation of authorization translation.

The rest of the paper is organized as follows. In Section 2 we give basic concepts of XML and an XML access control model. Section 3 discusses the impact of document structure transformation on authorization translation. In Section 4 we present a technique of translating authorizations for the target document. Section 5 introduces an algorithm for translating a path expression of an object to the corresponding path expression of the target schema. Finally, Section 6 concludes our work.

2. Basic Concepts

XML Documents, DTDs and XPath

An XML document 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. These attributes represent properties of the element. Both elements and attributes are allowed to contain values. The structure of an XML document is described by a DTD. A DTD can be modeled as a labeled tree containing a node for each attribute and element in the DTD. An example of XML document and its DTD are depicted in Fig.1(a) and (b), respectively. XPath [4] is a language for locating textual data, elements, and attributes in an XML document. In addition to its use for addressing, XPath can add conditions in the navigation.

An XML Access Control Model

In this paper, we adopt the XML access control model of Damiani [6]. Our approach can be easily adapted to other XML access control models. This model regulates the access of users to elements and attributes within an XML document on the basis of the user's identity and rules, called *authorizations*, which specify for each user the types of accesses that the user can/cannot exercise on each object. Authorizations can

be positive or negative to an XML element or attribute. Authorizations specified on an element can be defined as applicable to its attributes only (local authorizations) or, in a recursive approach to its subelements and attributes (recursive authorizations). This model provides document-level and schema-level authorizations. Schema-level authorizations are applicable to all XML documents that are instances of the DTD. Document-level authorizations allows user to tailor security requirements for each document. Document-level authorizations usually take precedence over the schema-level ones. To address the situations where the precedence criteria should not be applied, the model allows users to specify the authorization (either local or recursive) as weak type.

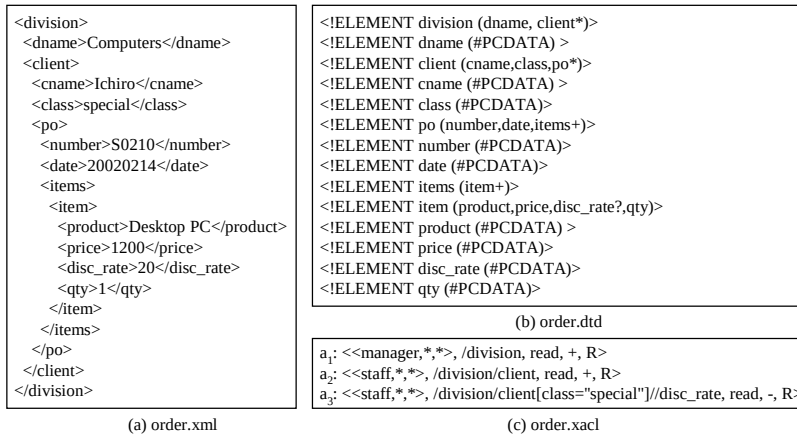


Fig. 1. A sample of XML document (a), DTD (b), and access control list (c).

Definition 1 (Authorization): An authorization is a 5-tuple of the form:
<*subject*, *object*, *action*, *sign*, *type*>, where

- *subject* is a user to whom the authorization is granted. *subject* is described by a triple (user-id, IP-address, symbolic-address),
- *object* described by a path expression identifying an element and attribute,
- *action* is the read operation,
- *sign* ∈ {'+', '-'},
- *type* ∈ {*L*,*R*,*LW*,*RW*} is an authorization propagation type (*L*ocal, *R*ecursive, *L*ocal *W*eak, and *R*ecursive *W*eak, respectively).

•

We call an authorization whose object definition is based on values of elements or attributes a *value-dependent* authorization. We call an authorization whose object definition is not based on values of elements or attributes a *value-independent* authorization. An example of access control list for order.xml is shown in Fig.1(c). Authorization a_1 and a_2 are value-independent authorizations. a_2 states that Staff is allowed to read information of the clients. a_3 is a value-dependent authorization. It states that Staff is not allowed to read discount rates of the order items of special class

customers. For simplicity, we consider *L* and *LW* as a local type. We consider *R* and *RW* as a recursive type.

3. Impact of XML Document Structure Transformation

We first analyze the impact of XML document transformation on authorization translation. We classify the impact as follows:

Total Mapping / Partial Mapping
Total mapping indicates that every schema element in a schema has relationship with the schema element(s) in another schema. Partial mapping occurs when some schema elements in either schema have no relationship with those in another schema. Note that a schema element is an XML element or attribute. We call a source schema element that has no relationship with any target schema elements an *unmapped source schema element* (USE). We call a target schema element that has no relationship with any source schema elements an *unmapped target schema element* (UTE). In case source and target schemas represent the same concept, USE and UTE are internal elements. We define an authorization for UTEs by three optional policies: Open policy, Authorization-inheritance policy, and User-defined policy. Open policy allows all subjects to access UTEs. Authorization-inheritance policy allows UTEs to inherit authorizations from their parents. User-defined policy allows a security administrator to predefine authorizations for UTEs before translating authorizations.

Semantic Relationship between Source and Target Schema Elements
Semantic relationship between source and target schema elements is classified into one-to-one, one-to-many, many-to-one, and many-to-many relationships. The authorization of a target schema element *e* is computed by combining the authorizations of all source schema elements that have semantic relationships with *e*. In case object of authorization is based on values of schema elements that have one-to-many, many-to-one, or many-to-many relationships with the target elements, the authorization translation needs guidance from the security administrator who knows the value mapping between source and target schema elements.

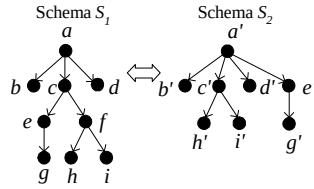


Fig. 2. A sample of change of element-subelement relationship.

Element-subelement Relationship
As an element-subelement relationship is changed, the descendant elements inheriting security policies from a given recursive authorizations may become different. For

example, schema S_1 and S_2 that are depicted in Fig.2 represent the same concept. Element *f* of schema S_1 is an USE. Suppose that there exists an authorization *auth*: <<*staff, *, *>, /a/c, read, +, R*> for XML document D_1 conforming to schema S_1. Descendant elements of element *c* of schema S_1 are different from those of element *c'* of schema S_2. Therefore, we cannot directly translate this authorization to <<*staff, *, *>, /a'/c', read, +, R*> for the document D_2 that is transformed from D_1 to conform to schema S_2. To solve this problem, we first convert *auth* into a set of authorizations that have the same authorization policies of *auth*. Therefore *auth* is converted to (1) a local authorization for element *c* and (2) recursive authorizations for elements *e*, *h* and *i* that are *c*'s the closest descendant elements, which are not USEs. We next translate path expressions of these authorizations to the corresponding path expressions of schema S_2.

Element and Attribute Values
Due to document transformation, values of some elements and attributes of XML target document may be different from the values of the corresponding elements and attributes of the source document. The security administrator uses value mappings between these source and target elements / attributes for translating value-dependent authorizations.

4. Translating Authorizations

We observe that in many occasions the semantic relationship between source and target schema element is a one-to-one mapping. In these cases, we offer a simple approach for translating authorizations. We first give definitions of a *DTD* graph and partial mapping.

Definition 2 (DTD Graph): A DTD graph is a 3-tuple $DG = (V, E, l)$, where *V* is the set of nodes in the graph, *E* is the set of edges, and *l* is the labeling function representing the properties of a node. We categorize a node based on its label:
(a) Element node: each element node *n* represents an element type. $l(n) = < N(n), A(n) >$ where $N(n)$ is *n*'s name.
(b) Attribute node: each attribute node *a* represents an attribute. $l(a) =< N(a), A(a) >$ where $N(a)$ is *a*'s name.
We assign a symbol in {*, +, ?} on edge *e*: $n_i \rightarrow n_j$ to indicate how many times n_j occurs in n_i's content model.

•

Definition 3 (Map Function): Let graphs $DG = (V, E, l)$ and $DG' = (V', E', l')$ be DTD graphs of source and target schemas, respectively. $map: V \rightarrow V'$ is a partial mapping from the nodes in *V* into the nodes in *V'*. $map\ (v) = v'$, where $v' \in V'$, $v \in V$, and node *v'* has semantic relationship with node *v*.

•

The partial mapping *map* can be derived from the schema matching, which is performed by manual work and a schema-matching tool. We represent *v* and *v'* as abso-

lute path expressions. For example, *map*(/*client*/*class*) = /*customer*/*@category*. We now give a formal definition of authorization preservation in translating an authorization.

Let D_1 be a document of schema S_1, D_2 be the document transformed from D_1 to conform to schema S_2, $AUTH = \{auth_1, auth_2, .. , auth_m\}$ be a set of authorizations for D_1, and $AUTH' = \{auth'_1, auth'_2, .. , auth'_n\}$ be a set of authorizations for D_2.

Definition 4 (Authorization Preservation): Let $V = \{v_1, v_2, .. , v_p\}$ be a set of schema elements of S_1, $V' = \{v'_1, v'_2, .. , v'_q\}$ be a set of schema element of S_2, $req_i = (subject_r, object_i, action_r)$ be an access request for an instance x of v_i $(1 \le i \le p)$ of D_1, and $req_j' = (subject_r, object'_j, action_r)$ be an access request for an instance x' of v'_j $(1 \le j \le q)$ of D_2. *AUTH' preserves* authorization policies of *AUTH* if and only if the following conditions are satisfied:
(for each instance x in v_i) (for each instance x' in v_j) ($v'_j = map(v_i)$) (x' corresponds to x): req_i and req_j' have the same permission decision (either granted or denied) by *AUTH* and *AUTH'*, respectively for $(1 \le i \le p)$ and $(1 \le j \le q)$.

•

It is worth to note that $object_i$ does not indicate USEs of schema S_1 while $object'_j$ does not indicate UTEs of schema S_2. We now describe how access control model decides whether to grant permission to a given access request.

Definition 5 (Grant Decision): Let $x \preccurlyeq y$ denote the fact that y is a descendant-or-self of node x, $subject \Rightarrow subject'$ denote the fact that *subject* is satisfied by definition of *subject'*. An access request $req = (subject_r, object_r, action_r)$ is granted by *AUTH* if and only if the following conditions are satisfied:
$(\exists auth_i)$ $(\neg\exists auth_j)$ $(subject_r \Rightarrow subject_i) \wedge (action_r = action_i) \wedge (sign_i = \text{`+'}) \wedge (subject_r \Rightarrow subject_j) \wedge (action_r = action_j) \wedge (sign_j = \text{`-'}) \wedge (((type_i = recursive) \wedge (type_j = recursive) \wedge (object_i \preccurlyeq object_j \preccurlyeq object_r)) \vee (object_j = object_r)))$, where $auth_i = \langle subject_i, object_i, action_i, sign_i, type_i \rangle$ and $auth_j = \langle subject_j, object_j, action_j, sign_j, type_j \rangle$.

•

An access request is denied by *AUTH* if the conditions of definition 5 are not satisfied. For an access request to an instance x of a schema element v, the grant/deny decision is based on the definitions of the authorizations whose path expressions indicate v. Therefore we can derive *AUTH'* that preserves policies of *AUTH* by creating the corresponding authorization $auth'_i$ of each $auth_i$ as an authorization of *AUTH'*. The *subject*, *action* and *sign* of $auth'_i$ are obtained from those of $auth_i$. The *object* of $auth'_i$ is derived by the result of translating the path expression of *object* of $auth_i$ to the corresponding path expression of the target schema. As we discussed in the previous section, there are some cases where we cannot directly translate an authorization of *AUTH*. We now give a formal definition for the authorization that can be directly translated to the corresponding authorization.

Definition 6 (Translatable Form): Let $auth_i = \langle subject_i, object_i, action_i, sign_i, type_i \rangle$ be an authorization of *AUTH* of D_1, $auth_j' = \langle subject_j', object_j', action_j', sign_j', type_j' \rangle$ be an authorization of *AUTH'* of D_2, v and v' be element nodes indicated by $object_i$

and *object*$_j$*'*, respectively. Let *desc*(*v*) be a set of *v*'s descendant elements that are not USEs, and *desc*(*v'*) be a set of *v'* 's descendant elements that are not UTEs. *auth' corresponds* to *auth* if and only if: (1) (*subject*$_i$ = *subject*$_j$*'*) and (*action*$_i$ = *action*$_j$*'*) and (*sign*$_i$ = *sign*$_j$*'*), (2) *v' = map*(*v*), (3) *object* corresponds to *object'*, and one of the following conditions are satisfied:

- (*type*$_i$ and *type*$_j$*'* are recursive types) and (each schema element in *desc*(*v*) has one-to-one mapping with a schema element in *desc*(*v'*)).
- *type*$_i$ and *type*$_j$*'* are local types.

In this case, we say that *auth*$_i$ is in *translatable* form.

•

We give a formal definition of condition (3) in the next section. For a recursive authorization *auth*$_i$ whose *object* indicates *v*, if *auth*$_i$ is not in translatable form, we generate *auth"*$_1$, *auth"*$_2$, .. , *auth"*$_q$ for *v*'s closest descendant elements v_1, v_2, .. , v_q, respectively. Note that v_1, v_2, .. , v_q are not USEs. The *subject*, *action*, and *sign* of *auth"*$_j$ ($1 \le j \le q$) are obtained from those of *auth*$_i$. If v_i is an internal element, the *type* of *auth"*$_j$ is defined as a recursive type. Otherwise, *type* of *auth"*$_j$ is defined as a local type. The path expression of *object"*$_j$ of *auth"*$_j$ is defined as concatenation value of path expression of *object* of *auth*$_i$ and /v_j. If *v* is not an USE, we change *type* of *auth*$_i$ to *local*. Otherwise, we remove *auth*$_i$ from *AUTH*. If *auth"*$_j$ ($1 \le j \le q$) is not in translatable form, we recursively apply the same approach in generating a set of authorizations for *auth"*$_j$ until all authorizations are in translatable form. We next translate path expression of *object* of each authorization in *AUTH* to the corresponding path expression of the target schema.

Proposition 1: Given *AUTH* for D_1, we can derive *AUTH"*={*auth"*$_1$, *auth"*$_2$, .. , *auth"*$_n$} from *AUTH* for D_1 where each *auth"*$_i \in$ *AUTH"* is in translatable form.

•

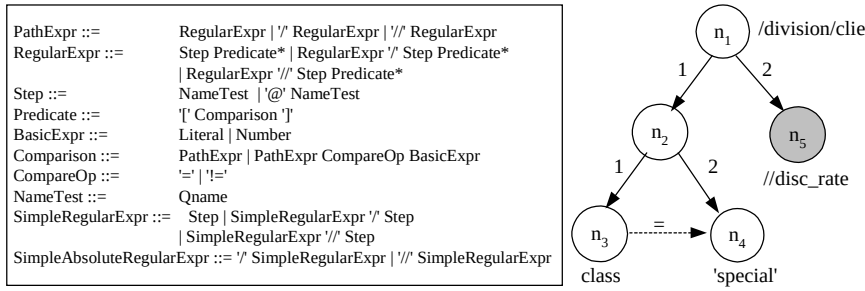
Proposition 2: Let *AUTH"* = {*auth"*$_1$, *auth"*$_2$, .. , *auth"*$_n$} be a set of translatable authorizations for D_1, *AUTH'* = {*auth'*$_1$, *auth'*$_2$, .. , *auth'*$_p$} be a set of authorization of D_2. *AUTH'* preserves the authorization policies of *AUTH"* if:
$\forall$*auth"*$_i$ $\exists$*auth'*$_j$ such that *auth"*$_i$ corresponds to *auth'*$_j$, where ($1 \le i \le n$) and ($1 \le j \le p$).

•

5. Translating a Path Expression

This section presents how to translate path expression of object of each authorization to the corresponding path expression of the target schema. We give a definition of the path expression used for defining the objects in the authorizations. We focus on the core part of XPath since it can sufficiently express location of authorized objects. We name the core part *XPathAuth*. We assume that path expression used for defining the objects are given in the form of *PathExpr* of Fig.3(a). From the syntax rule, we observe that the XPathAuth can be represented by the following sequence: $A_1\{P_1\}+ A_2\{P_2\}+ \ldots A_{n-1}\{P_{n-1}\}+ A_n\{P_n\}*$,where $n \ge 1$. A_1 represents a language of nonterminal symbols: *SimpleRegularExpr* or *SimpleAbsoluteRegularExpr*, A_i ($2 \le i \le n$) and P_j ($1 \le j \le n$) represent a language of nonterminal symbols: *SimpleAbsoluteRegularExpr*

and *Predicate*, respectively. '{}+' and '{}*' are meta symbols, which represent 'one or many occurrences' and 'zero or many occurrences' respectively. For example, the path expression */division/client[class='special']//disc_rate* can be viewed as a concatenation of A_1, P_1, and A_2 where A_1 is */division/client*, P_1 is *[class='special']*, and A_2 is *//disc_rate*. To clarify the relationship among *SimpleRegularExpr*, *SimpleAbsoluteRegularExpr*, and *predicate* in the path expression of an object, we introduce XPathAuth graph adapted from [11].



(a) Syntax of XPathAuth (b) A sample of XPathAuth graph

Fig. 3. (a) Syntax of XPathAuth and (b) a sample of XPathAuth graph.

Definition 7 (XPathAuth Graph): The XPathAuth graph is a directed graph $G(N, E)$ satisfying the following constraints:

- Every node has a node type that is one of the following five nonterminal symbols: *SimpleRegularExpr, SimpleAbsoluteRegularExpr, Literal, Number, and Boolean.* Every node that is not *Boolean* type has a value. For a node of type *T*, the value of the node is a language of *T*. There is exactly one node in *N* called the *output node* of *G*. A shaded circle depicts the output node.
- *E* is the union of two mutually disjoint sets of edges: E_t (*tree edges*) and E_c (*comparison edges*). A tree edge is depicted by a solid line. A comparison edge is depicted by a dashed line.
- The graph (N, E_t) is a tree with a root. In (N, E_t), children of a node are ordered. A tree edge from a parent *n* to its *i*-th child *m* is denoted by (n, i, m). A comparison edge has a *CompareOp* as a label. A comparison edge from *n* to *m* with a label θ is denoted by (n, θ, m).

•

The XPathAuth graph of the path expression */division/client[class='special'] //disc_rate* is shown in Fig.3(b). Node n_2 is a Boolean node. Intuitively, this path expression has three component paths: */division/client*, */division/client/class*, and */division/client//disc_rate*. These component paths are concatenated values of nodes in XPathAuth graph. The concatenated value is defined as follows.

Definition 8 (Concatenated-value of a Node in XPathAuth Graph) Let *n* be a node of *SimpleRegularExpr* or *SimpleAbsoluteRegularExpr* type in an XPathAuth graph *G*, *value* (*n*) be the value of *n*, and *concat*(*n*) be concatenated-value of *n* (in *G*). *concat*(*n*) is defined recursively as follows:

(1) If *n* has no ancestor node of *SimpleRegularExpr* or *SimpleAbsoluteRegularExpr* type, *concat*(*n*) = *value*(*n*).
(2) Otherwise, let n_a be the closest ancestor node of *SimpleRegularExpr* or *SimpleAbsoluteRegularExpr* type. *concat*(*n*) is computed as follows:
   - if *n* is of *SimpleRegularExpr* type, *concat*(*n*) = *concat*(n_a) & '/' & *value*(*n*),
   - if *n* is of the *SimpleAbsoluteRegularExpr* type, *concat*(*n*) = *concat*(n_a) & *value*(*n*), where & denotes the concatenation operator.

•

Definition 9 (Path Expression Correspondence): Let *path* and *path'* be path expressions of schema S_1 and S_2, respectively. Let *G*(*N*, *E*) and *G'*(*N'*, *E'*) be XPathAuth graphs representing *path* and *path'*, respectively. *path* corresponds to *path'* if and only if all the following conditions are satisfied:

(1) $|N| = |N'|$ and $|E| = |E'|$,
(2) $\forall i\ (\exists e_i \in E)\ (\exists e'_i \in E')$ (label of e_i is as same as label *of* e'_i) for $1 \le i \le |E|$,
(3) (For each $n_i \in N$ where n_i's type is either *Literal* or *Number*) $(\exists n'_i \in N')$ ($value(n_i) = value(n'_i)$) for $1 \le i \le |N|$, and
(4) (For each $n_i \in N$ where n_i's type is either *SimpleRegularExpr* or *SimpleAbsoluteRegularExpr*) $(\exists n'_i \in N')$ ($map(v_i) = v'_i$) for $1 \le i \le |N|$, where v_i and v'_i are full path style of *concat*(n_i) and *concat*(n'_i), respectively.

•

We present the algorithm *TranslatePath* (shown in Fig.4) that translates path expression of the object. Given an XPathAuth graph, we first parse the values of nodes of the XPathAuth graph in preorder to obtain the component paths in the XPathAuth graph. We expand the component path from wildcard form into full path. For example, */division/client//disc_rate* is expanded into */division/client/po/items/item/*

TranslatePath (*G, DG, DG', p*)
Input: (1) An XPathAuth graph *G*,
(2) a source DTD graph *DG* = (*V, E, l*), and
(3) a target DTD graph *DG'* = (*V', E', l'*).
Output: A path expression *p*.
Algorithm:
Traversing the nodes in the XPathAuth *G* in preorder.
For each node *m* that is not *Boolean* type, do the following steps:
Let $v_i \in V$ be the schema element indicated by *concat*(*m*), $v_k \in V$ be the schema element indicated by the concatenated-value of the closest ancestor node of *m*, and *val* be the location path from $map(v_k)$ to $map(v_i)$ of the target DTD graph.
If v_i is an USE **then** report translation error on v_i and terminate algorithm.
If node *m* is not connected with comparison edge **then**
p:= p & *val*. (where & denotes the concatenation operator)
Otherwise
Let θ be the label on the comparison edge connected to node *m*.
If comparison edge connected to node *m* is an out-edge
Then *p:= p* & '[' & *val* & θ . **Otherwise** *p:= p* & *val* & ']'.
return *p*.

Fig. 4. The algorithm TranslatePath.

disc_rate. We apply *map* function and the target DTD graph to translate each component path into the corresponding component path of the target DTD. If we can't find the corresponding path for a component path, the algorithm produces translation error report and terminates the translation. We next merge the translated component paths with values of nodes of *Literal* and *Number* types to obtain the corresponding path expression of the target schema.

6. Conclusion

In this paper, we present an approach that translates the authorizations for the XML document that is transformed from an XML source document. We focus on the case where schemas of the source and transformed documents represent the same concept. This approach is strongly based on schema mapping information between the source and transformed schemas. We conclude that XML document structure transformation has strong impact on definitions of authorizations of the transformed document. We have addressed problems and limitations of authorization translation.

References

1. AlphaWorks. Data Descriptors by Example, January 8, 2001. http://www.alphaworks.ibm.com/tech/DDbE.
2. E. Bertino et al., "Specifying and Enforcing Access Control Policies for XML Document Sources," World Wide Web, Baltzer Science Publishers, Netherlands, vol. 3, no. 3, 2000.
3. T. Bray et al. "Extensible Markup Language (XML) 1.0 (Second Edition)". World Wide Web Consortium (W3C). http://www.w3c.org/TR/REC-xml (October 2000).
4. J. Clark et al. "XML Path Language (XPath) Version 1.0". World Wide Web Consortium (W3C). http://www.w3c.org/TR/xpath (November 1999).
5. J. Clark. "XSL Transformations (XSLT) Version 1.0". World Wide Web Consortium (W3C). http://www.w3c.org/TR/xslt (November 1999).
6. E. Damiani, et al., Securing XML documents. In Proceedings of the 2000 International Conference on Extending Database Technology (EDBT'2000), Germany, March 2000.
7. M. H. Kay. SAXON DTDGenerator, http://users.iclway.co.uk/mhkay/saxon/saxon5-5-1/dtdgen.html, 13 April 1999.
8. M. Kudo and S. Hada. "XML Document Security based on Provisional Authorization". Proceedings of the 7th ACM conference on Computer and Communications Security. November 2000, Athens Greece.
9. T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In VLDB, pages 122-133, 1998.
10. H. Su, H. Kuno, E.A. Rundensteiner, "Automating the Transformation of XML Documents" Advances in Web-Age Information Management, 2nd International Conference WIDM 2001: 68-75, July 9-11, 2001.
11. M. Yoshikawa, et al., XRel: "A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases". ACM Transactions on Internet Technology, Vol.1, No.1, pp.110-141, 2001.