Improved Access Control Mechanism with XML coding and XML document updating

Meghdad Mirabi, Hamidah Ibrahim, Ali Mamat, Nur Izura Udzir
Department of Computer Science
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 Serdang, Selangor
Malaysia
{hamidah, ali, izura}@fsktm.upm.edu.my, meghdad.mirabi@gmail.com

ABSTRACT: *Now researchers felt the need for XML access control mechanism over World Wide Web. In addition, an efficient dynamic labeling scheme is required in order to eliminate the re-labeling process of existing XML nodes during XML document updating. However, the previous research on access control mechanisms for XML documents has not addressed the issue of integrating access control with a dynamic labeling scheme. Our research addressed the XML access control mechanism integrated with EXEL encoding and labeling scheme to eliminate the re-labeling process for updating the well-formed XML documents. The core of research in the paper is the authorization as a query condition to be satisfied. We have shown considerable improvements and benefits in the research.*

Keywords: XML coding, Labeling Scheme, Node Filtering, Tree-Awareness Metadata, XML Updating

Received: 12 September 2010, Revised 19 October 2010, Accepted 25 October 2010

1. Introduction

Recently, the eXtensible Markup Language (XML) [1] as a de facto standard for sharing and exchanging information over Internet and Intranet has been suggested. Thus, the need of managing XML documents over World Wide Web arises. Considering some operations such as querying and updating XML document, these kinds of operations should be quick to be carried out and more importantly safe from unauthorized access.

In order to quickly retrieve some parts of the XML document, several XML query languages such as XPath [2] and XQuery [3] have been proposed. In general, path expressions are used in these query languages for searching structural relationships between the XML nodes. These structural relationships should be evaluated efficiently. In order to determine the structural relationships between the XML nodes, they are labeled in such a way that the structural relationship between two arbitrary XML nodes can be computed efficiently. Many labeling schemes such as the region numbering scheme [4, 5] and the prefix labeling scheme [6] have been proposed.

However, these labeling schemes have high update cost; they cannot completely eliminate re-labeling process for the existing XML nodes when the XML document is updated. For instance, insertion of an XML node may change the XML tree structure, and the labels of XML nodes may need to be changed. Thus, many works have been studied to offer an efficient labeling scheme for the process of XML document updating such as [7-9].

Also, several researches have implemented the XML access control mechanisms which include studies by [10-27]. In general, XML access control mechanisms are classified into two groups: node filtering [10-15, 25] and query rewriting mechanisms [16-24, 26, 27]. In node filtering mechanism, access authorizations are determined by labeling the tree nodes with a permission (+), or a denial (-) and then pruning the tree based on associated signs. In query rewriting technique, access authorizations are employed to rewrite probable unsafe user queries into safe ones which should be evaluated against the original XML dataset. A safe query is a query which its result does not violate any access authorizations.

In this paper, we propose an XML access control mechanism tightly integrated with EXEL (Efficient XML Encoding and Labeling) [7-9] encoding and labeling scheme for XML document. The key idea is to regard an authorization as a query condition to be satisfied. Therefore, the benefit of speeding up searching and querying processes is obtained by employing such a labeling scheme in our proposed access control mechanism.

The rest of the paper is organized as follows: in Section 2, existing XML access control mechanisms are investigated. In Section 3, the access control model employed by our proposed mechanism is presented. Our proposed access control mechanism is presented in Section 4. Finally, the paper is concluded in Section 5.

2. Related Works

The first process in the traditional node filtering mechanism is to parse the XML document and generate its DOM tree then label the DOM tree based on access authorizations defined by a security administrator and finally prune unnecessary parts of the XML tree according to its labeling and show the result to the user [11, 12]. Due to traversing the DOM tree, access control mechanism proposed by [11, 12] is not scalable. If the DOM tree is very large it needs to have large memory space. Besides, to answer a user query, the whole DOM tree should be traversed which need long time to process. In order to resolve the problem, a fine grain access control mechanism which stores the XML documents as tables in relational database is presented in [25]. It is scalable with efficient response time and storage cost compared to [11, 12].

Another possible solution to overcome the shortages of the traditional node filtering mechanism is to separate the DOM and the SAX (*Simple API for XML*) when parsing the XML documents [13, 14]. If a user request has permission to read the XML document, it is processed by the SAX otherwise by the DOM.

In order to determine accessibility of XML elements, top-down and bottom-up strategies [10] traverse the paths between the root of XML tree and the element. In top-down strategy, authorization checking starts from the authorizations specified at the root while in bottom-up strategy, authorization checking starts from the ones specified at the most specific granularity level and going up through XML tree to find appropriate authorization. In the worst case, both strategies require traversing the whole XML tree which can be time consuming.

The main idea of static analysis proposed by [16], as a pre-processing access control mechanism, is to make automata for XML queries, XML access control policies, and XML schemas and then compare them. As a result, it does not examine real XML documents and runtime checking is not needed. Runtime checking is only required when real XML documents are required to check. The static analysis is not intended to entirely eliminate runtime checking, but rather intended to complement it. When static analysis cannot provide determinate answer, runtime checking is needed. This method classifies an XML query at compile time into three categories: entirely authorized, entirely prohibited, or partially authorized. Entirely authorized or entirely prohibited queries can be executed without access control. However, the static analysis cannot obtain any benefits when a query is classified as a partially authorized one. QFilter [22] as an external pre-processing XML access control system checks XPath queries against access control policies and rewrite queries according to access control policies before passing the revised queries to XML engine to process. Static analysis method [16] needs a runtime checking to filter out the unauthorized data while QFilter [22] solves this problem by rewriting XPath queries to filter out unauthorized part of input queries before passing them to XML query engine. Thus, QFilter has much better performance than static analysis method. However, if there are many access control policies for each role, NFA (Non deterministic Finite Automata) based approach in QFilter may have unacceptable overhead. Moreover, QFilter rewrites some kind of queries incorrectly according to examples derived in [24]. On the contrary, a DFA (Deterministic Finite Automata) based enforcement mechanism is devised in [18, 21] which decreases the complexity of query rewriting and always check whether the user has the right to consult the nodes that occur within the predicates.

A view based access control mechanism is proposed by [17] which generates not only a view called security view, but also a DTD view in which the security view conforms. The DTD view is generated to improve the efficiency of query rewriting and optimization. In contrast to [17], [19, 20, 26, 27] consider general XML DTDs defined in terms of regulations rather than normalized DTDs. Furthermore, [19, 20, 26, 27] do not permit dummy element types in the definition of security views.

The XML access control mechanism proposed by [23, 24] is based on access control abstraction and query rewriting. Access control abstraction is an efficient mechanism to check only the necessary access control rules based on user query instead of checking all of the access control rules. Also, user queries are rewritten by extending or eliminating XML tree nodes of DTD and operators such as union, intersection, and except are supported to transfer user queries into safe and correct queries which maintain the user's access control policies.

An efficient XML access control mechanism which integrates with query processing using DP (Dynamic Predicate) is devised in [15]. Accessibility of elements is checked during query execution using the DP. The key idea for integrating access control with XML query processing is to discover a set of elements which have the same accessibility. To effectively search the authorization, the mechanism proposed by [15] uses authorization index and nearest ancestor search technique.

3. XML Access Control Model

An access control policy includes a set of access authorizations. In general, an authorization can be defined as 5-tuple <subject, object, action, permission, propagation> where subject is the user or role concerned by the authorization. In this study, we assume that the subject is fixed; therefore the authorization is formed as 4-tuple <object, action, permission, propagation>. Object is presented by XPath expression [2] which contains element(s) of the XML document. Action is an executable action which can be Read, Insert, Delete, Update, and Rename. Permission represents the acceptance (+) or denial (-) of rights. Therefore, we refer to the access authorizations that grant access to an object as positive and those that deny access as negative. Details of the executable actions in the model are described below:

- If a user holds a Read privilege on a node *u*, s/he is permitted to read the content of node *u* and its descendants.
- If a user holds an Insert privilege on a node *u*, s/he is permitted to insert a new node as a sibling, child, or parent node of the node *u*.
- If a user holds a Delete privilege on a node *u*, s/he is permitted to delete the node *u* and its sub-trees.
- If a user holds an Update privilege on a node *u*, s/he is permitted to update the content of node *u*.
- If a user holds a Rename privilege on a node *u*, s/he is permitted to rename the node *u*.

An authorization can be explicit or implicit with the aim of limiting the number of authorizations which must be defined. An explicit authorization is explicitly defined on an XML element while an implicit authorization is implied by an explicit authorization which is defined on the nearest ancestor of XML element. Also, authorizations can be strong or weak with the purpose of providing more flexibility to the model. A strong authorization does not permit an implicit authorization to be overridden while a weak authorization permits an explicit authorization overrides an implicit authorization. The propagation policy in the model is *most specific override takes precedence*. It means an explicit authorization on an element overrides any weak authorizations defined on the ancestors of the element. Also, the *closed* policy is employed when there is not any authorization for an element. It means if there is not any authorization for an element or its ancestors explicitly or implicitly, the element is inaccessible. Besides, *denials take precedence* policy is as the conflict resolution policy. It means if both positive explicit authorization and negative explicit authorization for the same action are defined, negative authorization overrides positive ones.

An example of XML document and authorization for Read action is illustrated in Figure 1. Authorization1 does not permit the user to read the element "Patient" and its descendants. However, the Authorization2 overrides Authorization1 and permits the user to read the element "Therapy" and its descendants.

According to the authorization actions supported by the model, the query operations are as follows:

- Read (target)

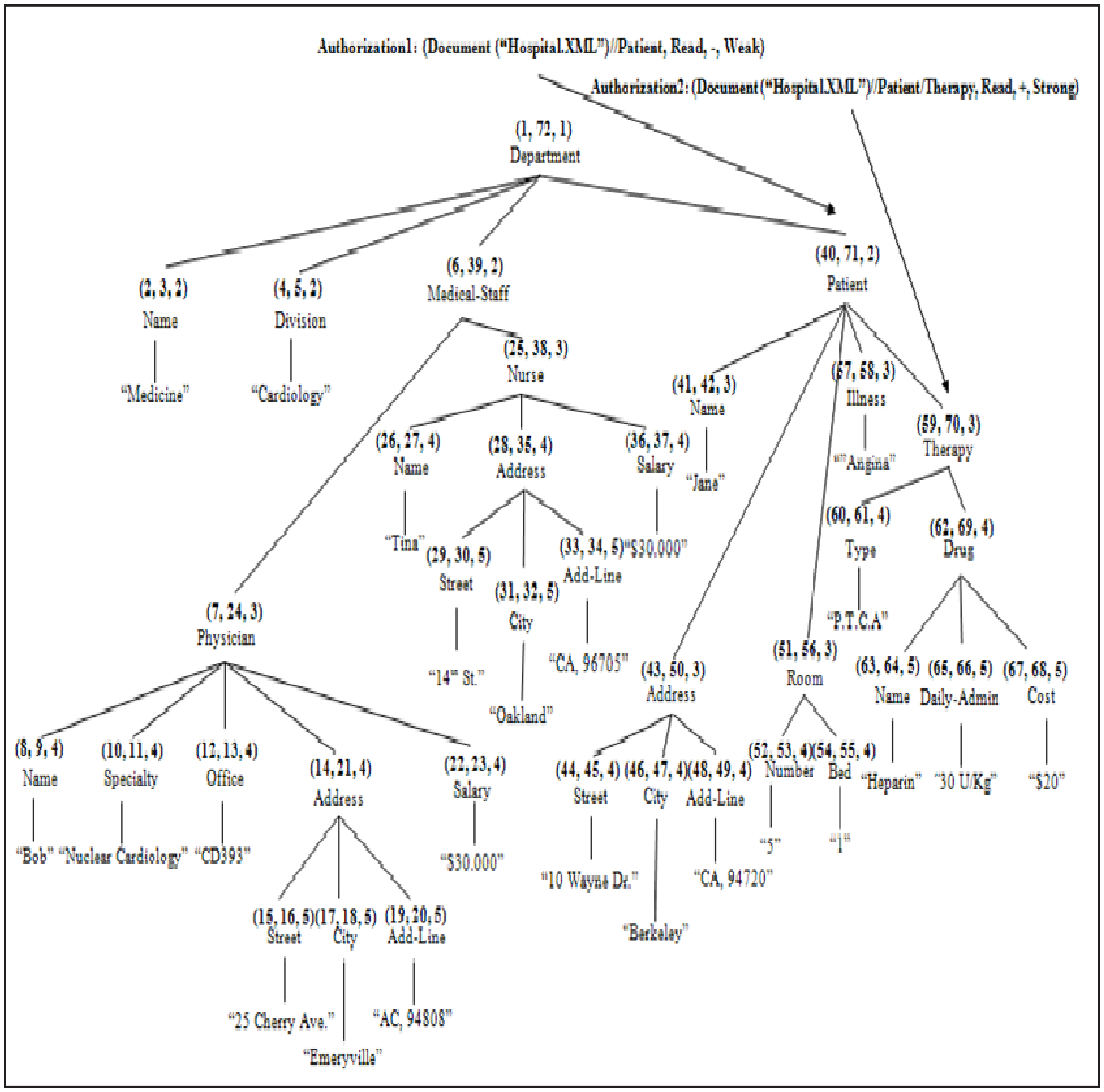
Authorization1: (Document ("Hospital.XML")//Patient, Read, -, Weak)
Authorization2: (Document("Hospital.XML")/Patient/Therapy, Read, +, Strong)
(1, 72, 1) Department
(2, 3, 2) Name "Medicine"
(4, 5, 2) Division "Cardiology"
(6, 39, 2) Medical-Staff
(25, 38, 3) Nurse
(26, 27, 4) Name "Tina"
(28, 35, 4) Address
(36, 37, 4) Salary "$30.000"
(29, 30, 5) Street "14th St."
(31, 32, 5) City "Oakland"
(33, 34, 5) Add-Line "CA, 96705"
(7, 24, 3) Physician
(8, 9, 4) Name "Bob"
(10, 11, 4) Specialty "Nuclear Cardiology"
(12, 13, 4) Office "CD393"
(14, 21, 4) Address
(22, 23, 4) Salary "$30.000"
(15, 16, 5) Street "25 Cherry Ave."
(17, 18, 5) City "Emeryville"
(19, 20, 5) Add-Line "AC, 94808"
(40, 71, 2) Patient
(41, 42, 3) Name "Jane"
(57, 58, 3) Illness "Angina"
(59, 70, 3) Therapy
(60, 61, 4) Type "P.T.C.A"
(62, 69, 4) Drug
(63, 64, 5) Name "Heparin"
(65, 66, 5) Daily-Admin "30 U/Kg"
(67, 68, 5) Cost "$20"
(43, 50, 3) Address
(44, 45, 4) Street "10 Wayne Dr."
(46, 47, 4) City "Berkeley"
(48, 49, 4) Add-Line "CA, 94720"
(51, 56, 3) Room
(52, 53, 4) Number "5"
(54, 55, 4) Bed "1"

Figure 1. An example of XML document and authorizations

- InsertChild (source, target)
- InsertBefore | After (source, target)
- InsertParent (source, target)
- Delete (target)
- Update (source, target)
- Rename (source, target)

Read is a read operation, in which target can be an element or an attribute. InsertChild is an insert operation, in which source can be a PCDATA, an element or an attribute. InsertChild inserts source as the child of element denoted by target. If the XML document contains a sequence of information, InsertBefore and InsertAfter are employed in the user query. InsertBefore inserts source before element denoted by target, and InsertAfter does after element denoted by target. In addition, InsertParent insert source as the parent of element denoted by target. Delete is a delete operation, in which target can be PCDATA, an element or an attribute. Update is an update operation, in which target can be an element or an attribute, and source can be a PCDATA. Rename is a rename operation, in which target can be an element or an attribute, and source is a new name.

4. XML Access Control Mechanism

In our proposed XML access control mechanism, relational database is employed with the purpose of storing the tree-awareness metadata of the XML document. Tree-awareness metadata contains information related to EXEL encoding and labeling scheme of the XML document tree which is required to eliminate the re-labeling process for existing XML nodes. Besides, access authorizations are stored in RDBMS with the aim of accelerating the process of access control over the XML document. Instead of traversing the whole XML tree to find the proper authorization, our proposed mechanism is capable to select only the necessary authorizations for processing a user query.

In the following, first EXEL encoding and labeling scheme proposed by [7, 8] is described and then our proposed access control mechanism integrated with the EXEL labeling scheme is explained.

4.1 EXEL Labeling and Encoding Scheme

EXEL [7, 8] encoding and labeling scheme is capable to remove the need of re-labeling as well as to compute the structural relationship between XML nodes effectively. Bit string is employed in the EXEL to encode the XML nodes. This bit string is ordinal as well as insert friendly. The definition of Lexicographical order (<) of bit string is defined as follows:

4.1.1 Lexicographical Order (<):

1. 0 is smaller than 1 ($0 < 1$) lexicographically.
2. Bit string a is equal to bit string b lexicographically, if a and b are the same ($a = b$).
3. For bit strings a_1, a_2, b_1, and b_2, $a_1b_1 < a_2b_2$, iff ($a_1 < a_2$) or ($a_1 = a_2$ and $b_1 < b_2$) or ($a_1 = a_2$ and b_1 is null (empty string)), where length (a_1) = length (a_2).

According to the above definition, for each bit string s which ends with ‘0’, the largest bit string among bit strings which are smaller than s lexicographically is the s’s longest prefix p (i.e., $s = p0$). However, we cannot generate any bit string which is greater than the prefix p and smaller than s. For example, there is not any bit string which can be inserted between ‘1110’ and its longest prefix bit string ‘111’. Thus, if the last bit of any two consecutive bit strings is ‘1’, we can insert a new one between the bit strings without any changes on them. The key idea to remove re-labeling during updating process of a node in the XML tree is *property* 1.

Property 1. For two bit strings $a1$ and $b1$, if $a1 < b1$ lexicographically, then $a < b$ lexicographically.

The algorithm of generating the bit string for nodes is illustrated in Figure 2. which is the enhanced binary encoding algorithm in [7, 8]. This algorithm obeys the *property*

```
Let N be the total number of nodes of XML tree,
  For first bit string $b(1)$, $b(1) = (0^{[log \, ^{N}_{Z}]})$
  For (i+1)th bit string $b(i+1)$, $b(i+1) = b(i) + 10$.
```

Figure 2. Algorithm of bit string generation

EXEL uses bit string generation algorithm with region labeling approach to solve the problem of re-labeling the nodes in the updating process. In region labeling approach [4, 5], each XML node of the XML tree is assigned to a region which contains a pair of start and end values which are determined by the position of the start tag and end tag of the node respectively. Additionally, to identify the Parent-Child (P-C) relationship between nodes efficiently, the level of nodes is added to this approach. An example of the XML tree labeled with <start, end, level> is illustrated in Figure 1. In order to determine the P-C relationship between nodes, the region numbering approach uses the level information but during the update process of XML data, the level information is sensitive information. For example, when a new node is inserted as an ancestor, the level information of a lot of nodes should be changed. To solve this problem, EXEL uses the parent information instead of the level information. An example of XML tree encoded and labeled by EXEL is shown in Figure 3. For more information about EXEL labeling scheme, refer to [7-9].

4.2 The Proposed Access Control Mechanism

In order to explain our proposed mechanism which is tightly integrated with EXEL encoding and labeling scheme, we consider a part of well-formed XML document with access authorizations in Figure 3.

Each element of the XML document illustrated in Figure 3. is labeled with <Start, End, ParentStart> based on EXEL encoding and labeling scheme. Such information is stored in EXEL-METADATA relation. The schema of EXEL-METADATA is as follows:

EXEL-METADATA (NodeLabel, Start, End, ParentStart)

An instance of EXEL-METADATA relation of the XML document illustrated in Figure 3 is shown in Table 1. ParentStart attribute is the parent's start value of a node.

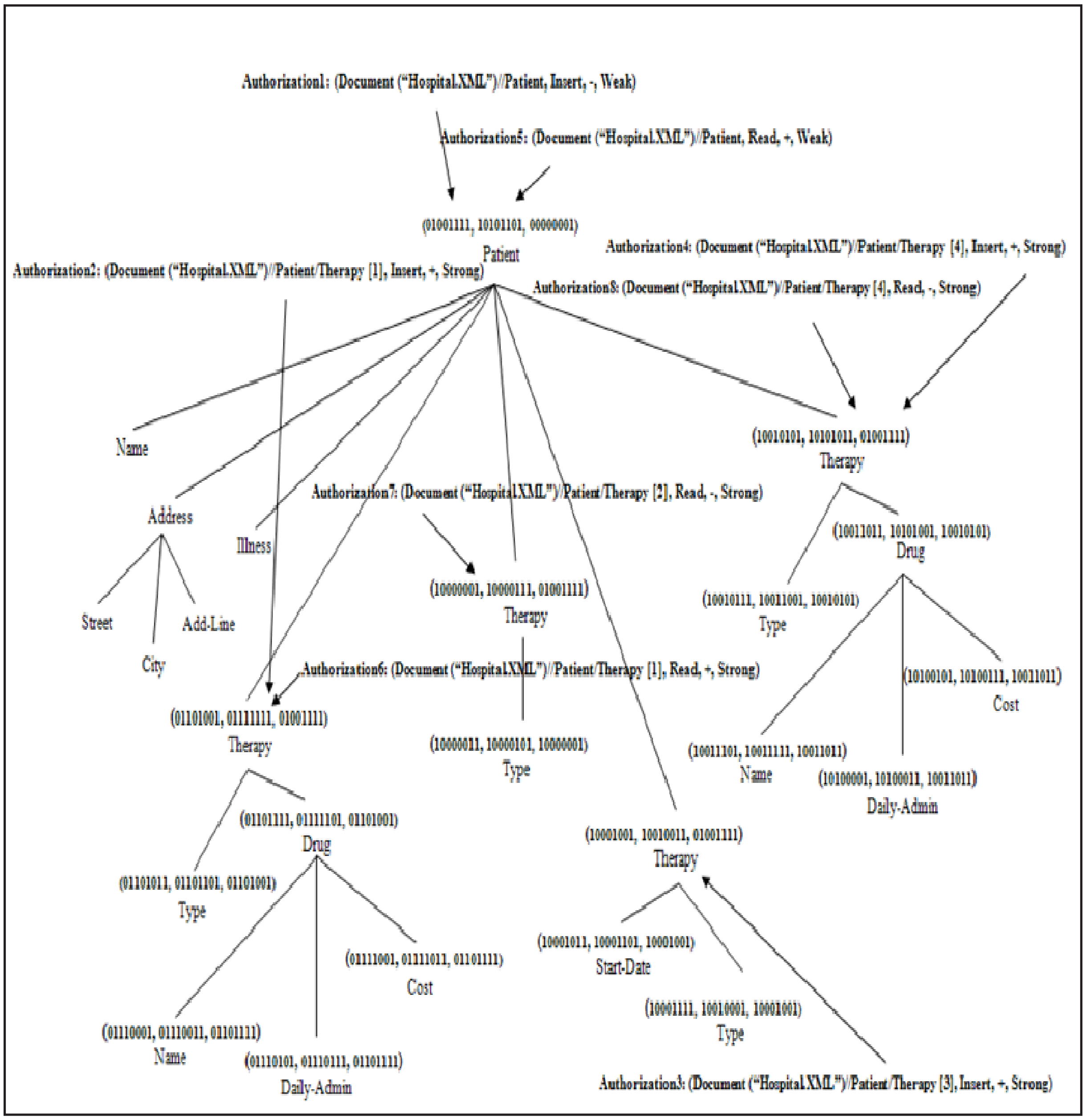

Figure 3. An Example of XML Document and Authorizations

NodeLabel	Start	End	ParentStart
Patient	01001111	10101101	00000001
…	…	…	…
Therapy	01101001	01111111	01001111
…	…	…	…
Therapy	10000001	10000111	01001111
…	…	…	…
Therapy	10001001	10010011	01001111
...	…	…	…
Therapy	10010101	10101011	01001111
…	…	…	…

Table 1.An instance of EXEL-METADATA relation

ID	Object	Start	End	Action	Permission	Type
1	//Patient	01001111	10101101	Insert	-	Weak
2	//Patient/Therapy[1]	01101001	01111111	Insert	+	Strong
3	//Patient/Therapy[3]	10001001	10010011	Insert	+	Strong
4	//Patient/Therapy[4]	10010101	10101011	Insert	+	Strong
5	//Patient	01001111	10101101	Read	+	Weak
6	//Patient/Therapy[1]	01101001	01111111	Read	+	Strong
7	//Patient/Therapy[2]	10000001	10000111	Read	-	Strong
8	//Patient/Therapy[4]	10010101	10101011	Read	-	Strong

Table 2. An instance of AUTHORIZATION relation

Given a well-formed XML document, a set of access authorizations and a user query, our proposed access control mechanism checks authorizations according to the access authorizations defined by a security administrator for the user query and execute the user query action if the user is authorized to carry out. Therefore, our proposed XML access control mechanism contains the following steps:

1. Extract the target node of the user query.
2. Retrieve EXEL metadata of the target node. The retrieved information contains EXEL metadata for a set of candidate nodes.
3. Find the nearest positive ancestor authorization for each candidate node.
   3.1. If the candidate node has such an authorization, the candidate node with EXEL metadata will be forwarded to step 4.
   3.2. Else the query will be rejected.
4. Execute the user query action for all candidate nodes forwarded from the step 3 and update metadata stored in relational database based on the user query action.

As mentioned in Section 3, authorization objects deal with XPath expression [2] which contains element(s) of the XML document. In order to extract the target node of a user query, it is desirable to define the target node.

Definition of Target Node: the last node of each XPath expression in a user query is a target node. For instance, the target node of “//Department/Patient” is the “Patient” node.

According to the above definition, the proposed mechanism is able to extract the target node of a user query in the first step. The following SQL query is constructed in the second step with the purpose of retrieving EXEL metadata of the target node. The result of the SQL query contains a set of the EXEL metadata of candidate nodes.

```
SELECT  *
FROM    EXEL-METADATA
WHERE   NodeLabel = Target-Node
```

Due to propagation policy *most specific override takes precedence* employed in our model, the accessibility of an element can be determined by finding authorizations specified on the nearest ancestor of the element. We borrow the definition of nearest ancestor authorization from [15] as defined below.

Definition of Nearest Ancestor Authorization: An authorization is called the nearest ancestor authorization $auth_{naa}$ of element *e* if it satisfies the following two conditions:

1. $auth_{naa}$ is an explicit authorization granted on the element *e* or one of its ancestor elements regardless of its authorization action;

2. No explicit authorization exists on element in the depth between the element *e* and the element on which $auth_{naa}$ is granted.

```
QueryAction ← user query action;
    For each candidate node n do {
        // AA is a set of Ancestor Authorizations;
        AA ← (Select * from AUTHORIZATION
                  Where (Start <= (Start value of n))
                        AND
                        (End >= (End value of n))
                        AND
                        (Action = QueryAction));
        MinLength ← abs ((Start value of n) – (Start value of AA[0]));
        NAA ← AA[0];    //NAA is Nearest Ancestor Authorization;
        For (i=1; i<length(AA); i++){
            temp← abs((Start value of n) – (Start value of AA[i]));
            If (temp <  MinLength) then
                MinLength ← temp;
                NAA ←AA[i];
            else if (temp == MinLength) then
                NAA← negative authorization between AA[i] and NAA
                is selected; //based on conflict resolution policy;
        }}
```

Figure 4. The Algorithm of Finding Nearest Ancestor Authorization

```
QueryAction← user query action;
Switch (QueryAction) {
    Case "Read":
        Show the candidate nodes and its descendants;
        Break;
    Case "Update":
        Update content of the candidate nodes with source parameter in the user query;
          //re-labeling the existing nodes is not needed;
        Break;
    Case "Delete":
        Delete the candidate nodes with their sub-trees;
          //re-labeling the existing nodes is not needed;
        Delete EXEL metadata of the candidate nodes and their descendants form
        EXEL-METADATA relation;
        Break;
    Case "Rename":
        Rename tag of the candidate nodes with source parameter of the user query;
          //re-labeling the existing nodes is not needed;
        Break;
    Case "InsertBefore":
        For each candidate node n do
                Insert source parameter as a sibling node before node n using
                InsertSiblingBefore Algorithm;
                Insert EXEL metadata of inserted node into EXEL-METADATA relation;
        Break;
    Case "InsertAfter":
        For each candidate node n do
                Insert source parameter as a sibling node after node n using
                InsertSiblingAfter Algorithm;
                Insert EXEL metadata of inserted node into EXEL-METADATA relation;
        Break;
    Case "InsertChild":
        For each candidate node n do
                Insert source parameter as the child of node n using InsertChildOf Algorithm;
                Insert EXEL metadata of inserted node into EXEL-METADATA relation;
        Break;
    Case "InsertParent":
        For each candidate node n do
                Insert source parameter as the parent of node n using InsertParentOf Algorithm;
                Insert EXEL metadata of inserted node into EXEL-METADATA relation;
        Break;

}
```

Figure 5. User query execution algorithm

Note: if a strong authorization satisfies the first condition, it automatically satisfies the second condition by the definition of the strong authorization.

Due to supporting different actions in our proposed mechanism, such a mechanism must be able to find the nearest ancestor authorization which its action is the same as the user query action. The algorithm of finding the nearest ancestor authorizations is shown in Fig. 4. Instead of traversing all nodes to find the proper authorization, the process of determining the accessibility of an element is accelerated with this algorithm in the step 3 of our proposed mechanism.

Due to positive and negative authorizations, the candidate nodes which the nearest ancestor authorizations have positive permission are passed to the forth step. Therefore, in the forth step, our proposed mechanism executes the user query action for all candidate nodes passed from the third step using the algorithm illustrated in Figure 5.

The reason of using EXEL as an encoding and labeling scheme in this study is its behavior in XML updating process. Relabeling the existing nodes of the XML document after some operations such as renaming the node tag, updating the value of leaf node, or deleting a leaf node or whole sub-tree is not needed. The problem of updating the XML document is in insertion operations. In XML document tree, three types of insertion can occur depending on the position of node to be inserted: insertion of a node as a child of a leaf node, insertion of a node as a sibling node, and insertion of a node as a parent node.

Before inserting a node, an algorithm is needed to generate new bit string between two preexisting bit strings. We use the MakeNewBitString algorithm proposed by [7, 8] to generate new bit string. Also, InsertSiblingAfter algorithm proposed by [7, 8] is used to insert a node as a sibling node after the node denoted by *cur*. The behavior of inserting a new sibling node before a node is similar to that of inserting a node after. In addition, InsertChildOf algorithm proposed by [7, 8] is used to insert a node as the child of node denoted by *cur* and InsertParentOf algorithm proposed by [7, 8] is used to insert a node as the parent of node denoted by *cur.* Refer to [7-9] for more information.

5. Conclusion and Future Works

In this study, we propose an XML access control mechanism integrated well with EXEL encoding and labeling scheme. Consequently, the process of relabeling the existing node is not required when the XML document is updated. Also, another benefit of integrating access control mechanism with EXEL encoding and labeling scheme for XML document is to accelerate the query response time depending on access authorizations.

As a future study, we intend to compare our proposed mechanism with the traditional node filtering mechanism. Also, we intend to extend our mechanism to support value based predicates for user queries as well as access authorizations.

References

[1] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.(2008). Extensible Markup Language (XML) 1.0 (5th Edition) W3C Recommendation, http://www.w3.org/TR/REC-xml/

[2] Clark, J., DeRose, S. (1999). XML Path Language (XPath) Version 1.0, , http://www.w3.org/TR/xpath/

[3] Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J.(2007). XQuery 1.0: An XML Query Language, htrp://www.w3.org/TR/xquery/

[4] Li, Q., Moon, B.(2000). Indexing and Querying XML Data for Regular Path Expressions. *In*: Proceedings of the 27th International Conference on Very Large Data Bases, p.361-370. Morgan Kaufmann, Roma, Italy (1)

[5] Zhang, C., Naughton, J., DeWitt, D., Luo, Q., Lohman, G.(2001). On Supporting Containment Queries in Relational Database Management Systems. ACM SIGMOD Record Journal. 30 (2) 425-436.

[6] Tatarinov, I., Viglas, S.D., Beyer, K., Shanmugasundaram, J., Shekita, E., Zhang, C.(2002). Storing and Querying Odered XML Using a Relational Database System. *In*: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, p.204-215. ACM Press, Madison, Wisconsin.

[7] Min, J.-K., Lee, J., Chung, C. W(2009). An Efficient XML Encoding and Labeling Method for Query Processing and Updating on Dynamic XML Data. *Journal of Systems and Software*. 82 (3) 503-515.

[8] Min, J.-K., Lee, J., Chung, C.-W. (2007). An Efficient Encoding and Labeling for Dynamic XML Data. *In*: Kotagiri, R., Krishna, P. R., Mohania, M., Nantajeewarawat, E. (eds.) Advances in Databases: Concepts, Systems and Applications. LNCS, V. 4443, p.715-726. Springer, Heidelberg.

[9] Mirabi, M., Ibrahim, H., Mamat, A., Udzir, N.I., Fathi, L. (2010). Controlling Label Size Increment of Efficient XML Encoding and Labeling Scheme in Dynamic XML Update, *Journal of Computer Science* 6 (12) 1529-1534.

[10] Bertino, E., Castano, S., Ferrari, E., Mesiti, M.(2000). Specifying and Enforcing Access Control Policies for XML Document Sources, *Journal of World Wide Web*. 3 (3) 139-151

[11] Damiani, E., Vimercati, S.D.C.d., Paraboschi, S., Samarati, P. (2000). Securing XML Documents, *In*: Zaniolo, C., Lockemann, P.C., Scholl, M.H., Grust, T. (eds.) EDBT 2000. LNCS, V. 1777, p.121-135. Springer, Heidelberg.

[12] Damiani, E., Vimercati, S.D.C.d., Paraboschi, S., Samarati, P.(2002). A Fine-Grained Access Control System for XML Documents, *Journal of ACM Transactions on Information and System Security* (TISSEC). 5 (2) 169-202.

[13] Jo, S.-M., Kim, K.-T., Kouh, H.-J., Yoo, W.-H.(2005): Access Authorization Policy for XML Document Security. *In*: Chen. G, Pan, Y., Guo, M., Lu, J. (eds.) Parallel and Distributed Processing and Applications. LNCS, V. 3759, p.589-598. Springer, Heidelberg.

[14] Jo, S.-M., Yang, C.-M., Yoo, W.-H. (2005). XML Access Control for Security and Memory Management. *In*: Glitho, R., Karmouch, A., Pierre, S. (eds.) Intelligence in Communication Systems. LNCS, V. 190, p.179-189. Springer, Heidelberg.

[15] Lee, J.-G., Whang, K.-Y., Han, W.-S., Song, I.-Y.(2007). The Dynamic Predicate: Integrating Access Control with Query Processing in XML Databases, *The VLDB Journal*. 16 (3) 371-387.

[16] Murata, M., Tozawa, A., Kudo, M., Hada, S.(2006). XML Access Control Using Static Analysis, *Journal of ACM Transactions on Information and System Security* (TISSEC). 9 (3) 292-324.

[17] Fan, W., Chan, C.-Y., Garofalakis, M. (2004). Secure XML Querying with Security Views. *In*: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp.587-598. ACM Press, Paris, France.

[18] Damiani, E., Fansi, M., Gabillon, A., Marrara, S.(2008). A General Approach to Securely Querying XML. *Journal of Computer Standards & Interfaces*. 30 (6) 379-389.

[19] Rassadko, N.(2007). Query Rewriting Algorithm Evaluation for XML Security Views. *In*: Jonker, W., Petkovic, M. (eds.) Secure Data Management. LNCS, vol. 4721, p.64-80. Springer, Heidelberg.

[20] Rassadko, N. (2006). Policy Classes and Query Rewriting Algorithm for XML Security Views. *In*: Damiani, E., Liu, P. (eds.), Data and Applications Security XX. LNCS, V. 4127, p.104 -118. Springer, Heidelberg.

[21] Damiani, E., Fansi, M., Gabillon, A., Marrara, S.(2007). Securely Updating XML. *In:* Apolloni, B., Howlett, R.J., Jain., L. (eds.) Knowledge-Based Intelligent Information and Engineering Systems. LNCS, v. 4694, p.1098-1106. Springer, Heidelberg.

[22] Luo, B., Lee, D., Lee, W.-C., Liu, P.(2004). QFilter: Fine-Grained Run-Time XML Access Control via NFA-based Query Rewriting. *In*: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management, p.543-552. ACM Press, Washington, D.C., USA.

[23] Byun, C., Park, S. (2006). Two Phase Filtering for XML Access Control. *In*: Jonker, W., Petkovic, M. (eds.) Secure Data Management. LNCS, vol. 4165, p.115-130. Springer, Heidelberg

[24] Byun, C., Park, S.(2006). An Efficient Yet Secure XML Access Control Enforcement by Safe and Correct Query Modification. *In*: Bressan, S., Küng, J., Wagner, R. (eds.) Database and Expert Systems Applications. LNCS, v. 4080, p.276-285. Springer, Heidelberg.

[25] Tan, K.-L., Lee, M.L., Wang, Y.(2001). Access Control of XML Documents in Relational Database Systems. *In*: Proceedings of the International Conference on Internet Computing, pp.185-191. CSREA Press, Las Vegas, Nevada, USA

[26] Kuper, G., Massacci, F., Rassadko, N.(2009). Generalized XML security views. International Journal of Information Security. 8 (3) 173-203.

[27] Kuper, G., Massacci, F., Rassadko, N.(2005). Generalized XML security views. *In*: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, p.77-84. ACM Press, Stockholm, Sweden.