Journal of Digital Contents Society Vol. 10 No. 1 Mar. 2009(pp. 129-145)

XML 트리 레벨을 고려한 관계형 데이터베이스 기반의 XML 접근 제어 모델

김진형*, 정동원**, 백두권***

요 약

웹 환경에서 안전한 정보의 분배와 공유가 중요해짐에 따라 유동적이고 효율적인 접근 제어 시스템에 대한 요구 또한 나타나게 되었다. 또한 eXtensible Markup Language (XML)이 인터넷 시대에 정보를 저장 및 교환하기 위한 de-factor 표준으로 인식됨에 따라, 최근 보안을 고려한 XML 모델의 확장에 대한 연구가 활발히 진행되고 있다. 그러나 이러한 최근의 연구들은 여전히 XML 문서에 사용되는 데이터들이 관계형 데이터베이스에 저장 및 관리 되고 있다는 사실을 간과하고 있다. 따라서 이러한 연구들은 이미 많이 제안되고 검증된 관계형 데이터베이스에 대한 보안 모델을 활용 할 수 없다. 이 논문에서는 기존의 연구들과는 다른 접근 방법을 기술한다. 이 논문은 객체 관점에서 관계형 데이터베이스에 대한 보안 모델을 지원하기 위한 XML 보안 모델에 대한 연구에 초점을 둔다.

이 논문에서 제안하는 접근 방법에서는 (1) 사용자는 주어진 XML 뷰 또는 스키마에 XML 질의를 한다. (2) XML 데이터에 대한 접근 제어 규칙은 관계형 데이터베이스에 저장된다. (3) XML 문서의 데이터는 관계형 데이터베이스에 저장된다. (4) 접근 제어 및 질의 싱행은 관계형 데이터베이스 내에서 수행된다. (5) XML 접근 제어는 XML 트리 레벨을 고려하여 수행된다.

RDB-based XML Access Control Model with XML Tree Levels

Jinhyung Kim*, Dongwon Jeong**, Doo-Kwon Baik**

Abstract

As the secure distribution and sharing of information over the World Wide Web becomes increasingly important, the needs for flexible and efficient support of access control systems naturally arise. Since the eXtensible Markup Language (XML) is emerging as the de-facto standard format of the Internet era for storing and exchanging information, there have been recently, many proposals to extend the XML model to incorporate security aspects. To the lesser or greater extent, however, such proposals neglect the fact that the data for XML documents will most likely reside in relational databases, and consequently do not utilize various security models proposed for and implemented in relational databases.

In this paper, we take a rather different approach. We explore how to support security models for XML documents by leveraging on techniques developed for relational databases considering object perspective. More specifically, in our approach, (1) Users make XML queries against the given XML view/schema, (2) Access controls for XML data are specified in the relational database, (3) Data are stored in relational databases, (4) Security check and query evaluation are also done in relational databases, and (5) Controlling access control is executed considering XML tree levels

Keywords : Relational database, XML Access Control, Object Perspective, XML tree level

※ 제일저자(First Author) : 김진형
접수일:2008년 12월 30일, 완료일:2009년 03월 11일
* 고려대학교 컴퓨터학과
jinhyung98.kim@gmail.com
** 군산대학교 정보통계학과 교수
*** 고려대학교 컴퓨터전파통신공학부 교수

▣ 이 논문은 2007년도 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임 (KRF-2007-331-D00448)
▣ 이 논문은 제 2차 BK 21 사업의 지원을 받았음

1. Introduction

As XML [1] is becoming a de facto standard for distribution and sharing of information, the issues for an efficient secure access of XML data has become very important. Various XML access control models and enforcement methods have been proposed recently. However, these approaches either assume the support of security features from XML database or use proprietary tools outside of databases. Since, there are currently few commercial XML databases with such capabilities, the proposed approaches are not yet practical[2][3][4]. In addition, [5] and[6] executed the researches about XACML, access control language which can be managed in XML documents. However, these researches do not consider the fact that the great amount data and access control information for XML documents is still stored into a relational database. Therefore, limitations of native XML security models are summarized as follows.

- Poor Practicality: The great amount of data is still stored into relational databases and there are few commercial XML databases. In addition, commercial XML database management systems are not used widely due to problem regarding stability and data translation costs.
- Lack of Stability: Native XML security policy is poor in stability because there are not enough commercial XML database management systems for practical application. However, RDB-based security models are proved to be stable by many researches and practical uses.
- Low Performance: In case of native XML security models, all of XML documents must be loaded into the system whenever user query is given. Such systems give us performance problem on processing of XML data access control.
- Simple User-based Security Policy: Native XML security models only provide simple user-based security policy. Therefore, native XML security models cannot support a data priority-based or a XML tree level-based XML access control. Native XML security models only supports 2 types access control: the local type for the indicated node or the recursive type for the indicated node and all of descendant nodes. That is, native XML security models cannot support access control for indicated node and a part of descendant nodes.

To solve limitations of native XML security models mentioned above, many researches about XML security model using relational database have been executed. [7] presents a XENA (XML sEcurity eNforcement Architecture) which stores XML documents as relational tables with pre-processing method. [8] suggests XML access control method with XACT (XML Access Control Tree). XACT is a tree which stores access control information for each node. However, because XACT must be created for each node in XML documents, if the size of XML documents is increased, the creating costs of XACT can be exorbitant. [9, 10] suggests QFilter which provides XML access control by shared NFA and comparison evaluation between pre-processing and post-processing. [11] suggests SQ-Filter by using the Pre and the Post values. However, the Pre and Post values must be calculated for each node in XML documents. Therefore, if the size of XML documents becomes bigger, the cost for creating Pre and Post values is enormous. Therefore, problems of conventional RDB-based XML security models are summarized as follows.

- Conceptual Model Level: Currently, researches about RDB-based XML security models just describe conceptual ideas or suggested simple conceptual model That is, detail syst em architecture or practical security model is not enough.

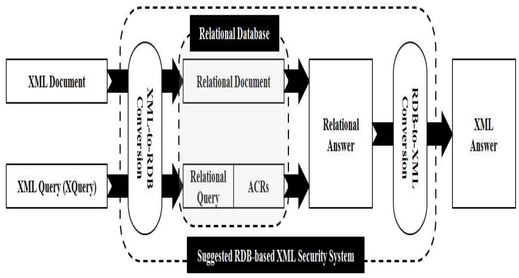
- Simple User-based Security Policy: RDB-based XML security models only provide simple user-based security policy. Therefore, RDB-based XML security models cannot support a data priority-based or a XML data level-based XML access control. As native XML security models, RDB-based XML security models also only support local and recursive model access control.
- Low Translation Efficiency: RDB-based security models store XML data into the relational database. For exact storing of XML data, we need effective XML-to-RDB and RDB-to-XML translation techniques without loss of hierarchical structure information about XML documents. However, conventional RDB-based security models need too much costs for storing hierarchical structure information of XML documents into the relational database.
- Inefficiency of Processing about User Queries: RDB-based XML security models have format heterogeneity among XML data, ACRs, and user quires. XML data is stored into the relational database. ACRs is defined as text form separately and user queries (XQuery) is translated into SQL form. These methods make us difficult to compare and analyze XML data, ACRs, and user quires. That is, cost of pre-processing is high in conventional RDB-based XML security models.

Therefore, our goal in this paper is to study how to support XML security models more effectively than conventional RDB-based XML security models by utilizing security support of relational security models. In addition, we suggest security model considering XML tree levels for supporting more detailed and exact access control than existing XML security models. In this paper, we assume that

- ✓ XML documents are converted into and stored in relational databases.
- ✓ Users give queries as XQuery form and the queries are analyzed and stored into the relational databases.
- ✓ Access control rules are defined by security administrators and stored into relational databases.

Security check and query evaluation are done by relational databases considering XML tree levels and only valid answers are returned to users in the XML format.

This paper is organized as follows, We analyze existing native XML security models and RDB-based security models as a preliminary works and explore several research issues in section 2. In section 3, we describe a framework of Suggested Models and physical storage schema of XML documents, access control rules, and user queries. In section 4, we illustrate an experiment with experimental dataset and a comparative evaluation between conventional RDB-based XML security models. Finally, we conclude this paper with future works in section 5.



(Figure. 1) Overview Architecture of Suggested Model

2. Preliminaries

2.1 Native XML Security Models

Current access control research can be categorized into two groups: access control modeling and access control enforcement mechanisms.

On the model side, several XML access control models have been proposed. Starting with [12] for HTML documents; [2][3] describes X

ML access control with an authorization sheet to each document or DTD[4]. proposed and XML access control model to deal with authorization priorities and conict resolution. [13] introduced provisional authorization and XACL.[14] formalizes the way of specifying objects in XML access control using XPath. Most of the proposals adopt either role-based access control or credential-based access control. The major di_erence between them is the way they identify users. Credential-based access control is more exible and powerful in this aspect. However, in the research of access control enforcement mechanisms, people tend to choose a relatively simple access control model to avoid distraction.

XML access control enforcement mechanisms in native XML environment have been intensively studied in recent years. They are categorized into four classes: (1) engine level mechanisms implement security check inside XML database engine; each XML node is tagged with a label [15, 16] or an authorization list [17], and _ltered during query processing. (2) view-based approaches build security views that only contain access-granted data [18]. (3) pre-processing approaches check user queries and enforce access control rules before queries are evaluated, such as the static analysis approach [19], QFilter approach [10], access condition table approach [20], policy matching tree [21], secure query rewrite (SQR) approach [22], etc.

However, native XML security models have limitations. The amount of data is still stored into relational databases and commercial XML database is poor to use regarding stability and translation costs. Therefore, native XML security models represent low practicality and stability. In addition, because whenever user queries are given, all of XML document must be loaded. This point gives low processing performance about user queries. Besides, native XML security models just support simple user-based security policy without supporting a data priority-based or a XML data level-based XML access control.

2.2 RDB-based XML Security Models

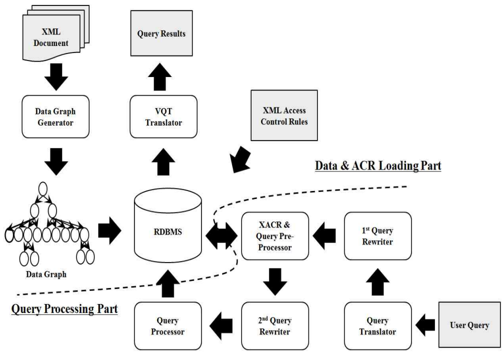
Since late 1990s, many researches about XML security model using relational databases have been performed. [7] proposes an idea of using RDBMS to handle access controls for XML documents in a limited setting. [7] suggests XENA system with schema-level XML security model, structure-based XML-RDB translation, and pre-pruning. In [9, 10], Bou suggests a practical and scalable solution, called Query Filter (QFilter). As an XML access control pre-processor external to the database engine, the QFilter checks XPath queries against access control policies. Instead of simply filtering out queries that do not satisfy access control policies and deferring the rest of queries to XML query engines for further checking and processing. QFilter takes extra steps to rewrite queries in combination of related access control policies by using shared NFA before passing the revised queries to underlying XML query engine for processing. However, QFilter includes unnecessary information in user query aspect and requests overheads for rewriting user query. [11] suggests SQ-Filter by using the Pre and the Post values. However, the Pre and Post values must be calculated for each node in XML documents. Therefore, if the size of XML documents becomes bigger, the cost for creating Pre and Post values is enormous.

3. Framework of Suggested Model

In this section, we describe architecture and detail components of RDB-based XML security model considering data levels (TL-BAC). In addition, we propose XML-to-RDB translation algorithm and use VQT algorithm [23, 24, 25] as RDB-to-XML translation algorithm. Beside s, we describe relational database schema for storing XML data, ACRs, user queries and us er query rewriting process with example.

3.1 Architecture

(Figure 2) shows a conceptual flow of acces s control processing.



(Figure 2) Conceptual Processing Flow of the Suggested Model

In the TL-BAC system, we store XML dat a from XML documents, XML access control rules by the security administrator, and user q ueries into relational database with a similar f orm. XML documents are converted into the d ata graph and stored into relational database without loss of hierarchical structure informati on. XML access control rules are defined by t he security administrator in relational database s directly. User queries are given by XQuery form and rewritten and stored into relational d atabases. Stored user queries are combined int o XML access control rules and rewritten agai n. User queries rewritten 2 times are sent to r elational databases and search results are retur ned to users. TL-BAC system consists of 9 c omponents and functions of each component ar e as <Table 1>

<Table 1> Components Constitution

Name of Component	Function
Data Graph Generator	Creates data graphs based on given XML documents
Data Translator	Analyzes data graphs and stores data into RDB based on data graphs
VQT Translator	Converts relational-form query results into XML form
ACR Manager	Defines and updates XML access control rules Translates Recursive ACR into Local ACR
Query Translator	Converts XQuery into relational form and stores into relational databases
1stQuery Rewriter	Rewrites recursive mode queries as local mode queries
Query Pre-Processor	Combines user quires and access control rules and analyzes for access control
2ndQuery Rewriter	Rewrites queries based on analysis results of query pre-processor
Query Processor	Sends user queries to relational database and executes search

3.2 XML-to-RDB and RDB-to- XML Conversion

For storing XML data into relational databa ses, we must convert XML documents into rel ational database without loss of hierarchical st ructure information of XML documents. For ef fective extraction of information about the hier archical structure from XML documents, the f ollowing processes are needed. First, we analy ze the schema for the XML document and cre ate a data graph with a hierarchical relationshi p. Second, we perform a depth-first search fro m root node to leaf nodes in XML documents based on the created data graph, and create p aths for each node. If we arrive at a leaf nod e, we create a path for the leaf node and the i ntermediate nodes. However, we create paths f or intermediate nodes just once, thus we can avoid duplicate path creation. <Table 2> repre sents descriptions about the symbols and notat ions used in the path creation algorithm.

In the data graph, each node consists of the following elements. Definition 1 represents the node constitution in the data graph.

<Table 2> Description about Symbols and Notations in Algorithm

Notations	Description
Node.number	Node number assigned by DFS search in the Data Graph
Node.visiting_flag	Visited nodes have flag set to 1, nodes which have not been visited have flag set to 0
Node.child_flag	Nodes which do not have child nodes have flag set to 0, Nodes which do have them have flag set to 1
Node.sibling_flag	Nodes which do not have sibling nodes have flag set to 0, Nodes which do have them have flag set to 1
visiting_node[ ]	Array for representing visited node lists
DFS_visit( )	Function for search in the Data Graph by DFS search method
Nextnode( )	Function for description of next ordered node in DFS search
path_temp[ ]	Array for storing created path temporally before duplication checking
path_storage[ ]	Array for storing created path after duplication checking
Createpath( )	Function for creating XPath from root node to current node
Pathcheck( )	Function for checking duplicated creation of node paths
Backtrackingpath( )	Function for execution of path backtracking in the Data Graph

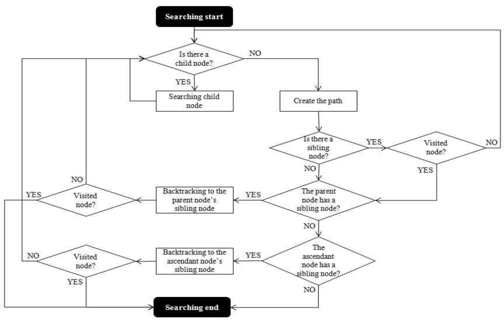
Definition 1. (Node Constitution in the Data Graph) Each node in the data graph is denoted by a 5-tuple; N(name) = (Na, Nu, V_f, C_f, S_f),where

- ✓ Na represents a specific name of each node
- ✓ Nu represents a specific node number which is assigned by DFS searching when the Data Graph is created
- ✓ V_f represents whether a node has been visited, and a path has been created for the node or not. If a node has been visited and a path has been created for the node, the visiting_flag of this node is 1, but if a node has not been visited, the flag is 0.
- ✓ C_f represents whether a node has child nodes or not. A node which has no child nodes has a flag set to 0 but a node which have child nodes has a flag set to 1.
- ✓ S_f represents whether a node has sibling nodes or not. A node which has no sibling nodes has a flag set to 0 but a node which have sibling nodes has a flag set to 1.

Definition 2. (Case Definition in the Path Creation) When we create path for each node from a Data graph, there are two representative cases and four detailed cases; case1, case2-1, case2-2, and case2-3.

- ✓ Case 1: Node.visiting_flag = 0 AND Node.child_flag = 0
- ✓ Case 2: Node.visiting_flag = 0 AND Node.child_flag = 1
- ✓ Case 2-1: Node.sibling_flag = 0 AND Sibling_node.visiting_flag = 0
- ✓ Case 2-2: Parent_node.sibling_flag = 0 AND Parent_node.siblng_node.visiting_ flag = 0
- ✓ Case 2-3: Ascendant_node.sibling_ flag=0 AND Ascendant_node.sibling_node. visiting _flag=0

Definition 2 defines various cases of path creation. Case 1 represents the case where the current node has not been visited and has child nodes. In case 1, we store the name of the current node in the visiting_node array and search the child node as the next order. Case 2 describes the case where a node has not been visited and does not have child nodes; the current node is leaf node. In this case, we create a path for the node and the intermediate nodes between the root node and the current node. However, we must check for duplicate path creation in this case, because the intermediate nodes of all sibling nodes are the same. To check for duplication of node paths, we temporarily store the created path in a path_temp array. The path stored in the path_temp array is compared with the path included in a path_storage array. If the created path is not contained in the path_storage array, the path is finally stored in the path_storage array. Case 2-1, case 2-2, and case 2-3 are backtracking cases, after we search leaf nodes. Case 2-1 illustrates th e case where a leaf node has sibling nodes which have not been visited. In this case, we perform backtracking to a parent node and search sibling nodes in the next ordering. Case 2-2 represents the case where a leaf node does not have sibling nodes which have not been visited. In this case, we perform backtracking to a parent node which has sibling nodes which have not been visited, and search sibling nodes of the parent node. If there are no sibling nodes of the parent node which have not been visited, we perform backtracking to the ascendant node. If the ascendant node has unvisited sibling nodes which have not been visited, we search these nodes, as in case 2-3. This process is iterated until we have searched every node in the Data Graph, and path creation is completed when there are no nodes in Case 1, Case 2-1, Case 2-2, and Case 2-3. (Figure 3) shows the flow of path creation in terms of search cases.



(Figure 3) Flow of Path Creation

<Table 3> Path Creation Algorithm

```
Procedure:
 Initialize i=1, j=0, k=0, m=0
 Class Node
   Initializestringname=null,
     Initializeintnumber=0,
     Initializeintvisiting_flag=0,
     Initializechild_flag=0,
     Initializesibling_flag=0
 For(Root_nodetoFinal_leaf_node)
   Initializevisiting_node[]
   DFS_visit(Node)
   if(Node.visiting_flag==0&&Node.child _flag ==0)
       visiting_node[k] = Node.name
       Incrementk
       Node.visiting_flag = 1
       Nextnode(child_node)
     Endif
     elseif(Node.visiting_flag==0&&
Node.child_flag==1)
       visiting_node[k] = Node.name
       Dowhile(visiting_node[k]!=null)
         Initializek=0
         Initializepath_temp[],path_storage[]
         path_temp[k]                          =
Createpath(Root_node,visiting_node[k])
         Forj=0toj(max)
           Form=0tom(max)
       Pathcheck(path_temp[j],path_storage[m])
           Incrementm
           if(path_temp[j]!=path_storage[m])
           path_storage[m(max)+ 1]=path_temp[j]
           Endif
         Incrementj
       Incrementk
 if(Node.sibling_flag==0&&sibling_node.visiting_flag
==0)
         Backtrackingpath(sibling_node)
         Nextnode(sibling_node)
       else
         Dowhile(parent_node!=null)      //    in case
of visited leaf nodes and sibling nodes
           Backtrackingpath(parent_node)
 if(parent_node.sibling_flag==0||parent_node.siblng_
node.visiting_flag==0)
           Nextnode(parent_node.sibling_node)
           elseLoop
     Endelseif
 End Procedure
```

RDB-to-XML translation is not focused in this paper and we use the VQT algorithm already developed by us [2][24][25]. The VQT algorithm is superiority to conventional translation algorithm because the VQT algorithm considers not only syntactic aspect but explicit/implicit semantic aspect.

3.3 Relational Database Schema

In TL-BAC system, relational databases include XML data, XML access control rules, and user queries. This information is stored as a s imilar schema structure. In TL-BAC system, XML access control rules and user queries have to be combined in a pre-processing step for query rewriting. In addition, rewritten user queries and XML data must be processed in a query processing step. If each form of information is different, additional algorithm or additional translation steps are needed for processing. Therefore, XML data, XML access control rules, and user queries are stored into relational databases as a similar schema structure for effective access control processing.

<Table 4> Storage Schema of XML Data

Doc_ID	Node_ID	Name	Parent_ID	Path
1	3	Name	2	Order/Customer_info/Name
1	5	Addr	2	Order/Customer_info/Addr
1	6	City	5	Order/Customer_info/Addr/City
1	7	Zip	5	Order/Customer_info/Addr/Zip
…	…	…	…	…

The storage schema of XML data is as <Table 4> The table for XML data consists of 5 attributes. The Doc_ID represents XML document number and the Node_ID represents node number in XML documents. In relational databases, several XML documents can be stored and each XML documents consists of many nodes. Therefore, we need identifiers for distinguishing XML documents and nodes. Name attribute represents name of each node and the Parent_ID represents the node identifier of parent node. The Parent_ID attribute can be used for searching child nodes in data priority-based or data level-based XML access control. The Path attribute represents XPath values of each node. By the XPath attribute, we can describe hierarchical structure information of XML documents in the relational databases easily and search parent node and root node simply.

<Table 5> Storage Schema of Access Control Rules Defined by the Security Administrator

Rule_ID	Subj	Role	Type	Path	Att	Value	Level
1	Bob	R+	R	Order/Order_info/Addr//	City	Seoul	.
2	Bob	W+	R	Order/Customer_info//	.	.	2
3	Jane	R-	L	Order/Customer_info/Credit_card	Name	Not Jane	.

The storage schema of XML access control rules is as <Table 5> This storage schema is a beginning schema defined by the security administrator and changed recursive form into local form automatically as <Table 6>

Definition 3. (Definition of Access Control Rules) Each access control rule in the Suggested Model is denoted by a 7-tuple; ACR=(S, R, T, P, A, V, L), where

- ✓ S represents a subject of access control rule.
- ✓ R represents a roles of access control rules. R can have R(read) or W(write). In addition, R represents a positive or negative roles by '+' or '-'.
- ✓ T represents a types of access control rules. T can have R(recursive) and L(local). The local type apply access control rules to only an indicated node. The recursive type apply access control rules to the indicated node and all of descendant nodes.
- ✓ P represents a path information that access control is applied. The path information is represents as the XPath form.
- ✓ A represent a attribute which has value.
- ✓ V represents a value that an attribute has.
- ✓ L represents a scope of access control rules. If L has 1, this means local type access control. If L has 2, this means that the acc

ess control rules are applied to the indicated node and child nodes. Through level, we can perform more detail access control except a local and recursive type access control.

<Table 6> Translated Storage Schema of Access Control Rules

Rule_ID	Subj	Role	Type	Path	Att	Value
1	Bob	R+	L	Order/Order_info/Addr	City	Seoul
2	Bob	R+	L	Order/Order_info/Addr/City	City	Seoul
3	Bob	R+	L	Order/Order_info/Addr/Zip	City	Seoul
4	Bob	W+	R	Order/Customer_info//	.	.
5	Bob	W+	R	Order/Customer_info//Name	.	.
6	Bob	W+	R	Order/Customer_info//Phone	.	.
7	Bob	W+	R	Order/Customer_info/Addr/	.	.
8	Bob	W+	R	Order/Customer_info//Credit_card	.	.
9	Jane	R-	L	Order/Customer_info/Credit_card	Name	Jane

The final storage schema of XML access control rules is as <Table 6> This storage schema is translated from <Table 5> automatically by the ACR Manager component. The table for access control rules consists of 8 attributes. The Rule_ID attribute is an identifier of each access control rule and the subj attribute is a subject of access control rule. The Role attribute is a positive/negative role of the subject and can have 'R(read)' or 'W(write)' or 'U(update)'values. Besides, '+' or '-' represent positive or negative roles. The Type attribute is an access control scope and can have 'R(recursive)' or 'L(local)' value. The recursive node includes indicated node and all child nodes of that node. However, the local node only includes indicated node. The Path attribute is a path information of node that the access control is applied to. The Att attribute is attribute information and the Value attribute include values related to the attribute. The Level attribute represents that the described rule is applied to the indicated node and child node represented as the level value. That is, level value 1 means that this access control rule is applied to the only indicated node, same as local mode. As the table 5, level value 2 means that this access control rule is applied to the indicated node and child nodes, not descendant. By using concept of level, the TL-BAC system can support detail and more exact access control that the conventional recursive mode or the local mode cannot support.

<Table 7> Storage Schema of User Queries

Query_ID	Subj	Role	Path
1	Bob	R	Order/Order_info//
2	Bob	W	Order/Customer_info//

The storage schema of XML data is as<Table 7> The table for XML data consists of 4 attributes. The Query_ID attribute represents an identifier of each user query. The Subj attribute represents a subject of user queries and the Role attribute represents a role which the subject wants to. The Path attribute represents path information of the target node. The given user queries are rewritten 2 times and detail rewriting process will be described in the next section.

3.4 Rewriting Process of User Queries

In the TL-BAC system, the given user queries are rewritten 2 times by the 1st query Rewriter component and 2nd Query Rewriter component. In the first query rewriting process, the Query Rewriter analyzes the given user queries and converts all of recursive mode queries into local model queries. The recursive mode queries includes ---//, ---/*, //---, */---, and so on. Such queries can be divided into several local mode queries. For effective combination and processing between user queries and access control rules, the local form queries are more simple and efficient for processing.

Theorem 1. (Query Writing Rules in the 1st Query Rewriting)

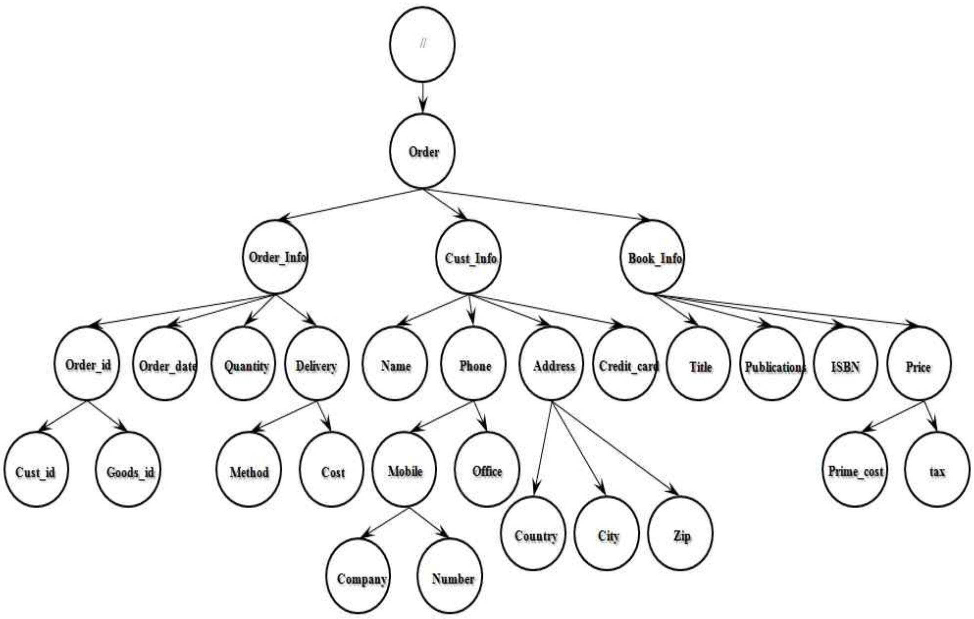
- ✓ ----// : Rewriting query with adding all of descendant nodes
- ✓ ---/* : Rewriting query with adding all of child nodes
- ✓ //--- : Rewriting query with adding all of ascendant nodes
- ✓ */--- : Rewriting queries with adding all of parent nodes

<Table 8> 1st Query Rewriting Results

Query_ID	Subj	Role	Path
1	Bob	R	Order/Order_info
2	Bob	R	Order/Order_info/Title
3	Bob	R	Order/Order_info/Publication
4	Bob	R	Order/Order_info/ISSN
5	Bob	R	Order/Order_info/Price
6	Bob	R	Order/Order_info/Addr
7	Bob	R	Order/Order_info/Addr/Zip
8	Bob	R	Order/Order_info/Addr/City
9	Bob	W	Order/Customer_info
10	Bob	W	Order/Customer_info/Name
11	Bob	W	Order/Customer_info/Phone
12	Bob	W	Order/Customer_info/Addr
13	Bob	W	Order/Customer_info/Addr/City
14	Bob	W	Order/Customer_info/Addr/Zip
15	Bob	W	Order/Customer_info/Credit_card

<Table 8> illustrates the 1st query rewriting results about the given user query shown in <Table7>. The 1st query rewriting is executed by rewriting rules mentioned above.

In the second query rewriting process, the



(Fig. 4) Experimental Dataset

Query Pre-Processor combines access control rules and user queries and rewrites queries by following rewriting rules.

Theorem 2. (Queries Writing Rules in the 2nd Query Rewriting)

- ✓ If user query included in the positive access control rules, no rewriting
- ✓ If user query included in the negative access control rules, delete query
- ✓ If user query included in the positive access control rules with attribute value, rewrite query with adding with attribute value.
- ✓ If a part of user query is included in the positive access control rules or the negative access control rules,

$$Q' = Q \cap (ACR+ \cap (ACR-)^{-1})$$
$$= (Q \cap ACR+) \cap (Q \cap (ACR-)^{-1})$$
$$= (Q \cap ACR+) - (Q \cap ACR-)$$

<Table 9> illustrates the 2nd query rewriting results about the rewritten user query show in <Table8>. The 2nd query rewriting is performed by rewriting rules mentioned above.

<Table 9> 2nd Query Rewriting Results

Query_ID	Subj	Role	Path
1_1	Bob	R	Order/Order_info/Addr[City=’Seoul]
1_2	Bob	R	Order/Order_info/Addr/Zip[City=’Seoul]
1_3	Bob	R	Order/Order_info/Addr/City[City=’Seoul]
2_1	Bob	W	Order/Customer_info
2_2	Bob	W	Order/Customer_info/Name
2_3	Bob	W	Order/Customer_info/Phone
2_4	Bob	W	Order/Customer_info/Addr
2_5	Bob	W	Order/Customer_info/Credit_card

4. Experiment and Evaluation

4.1 Experimental Dataset

This experiment is focus on the accuracy about access control of the Suggested Model. In this experiment, we use simple XML dataset shown in the (Figure 4). The XML dataset describes the book order information and includes 29 elements. In addition, the XML dataset consists of an order information, a customer information, and a book information. The XML document is translated and stored into the relational database. The translated dataset is as <Table 10>.

<Table 10> Translated Dataset in the RDB

Cust_id	Goods_id	Order_date	Quantity	Method	Cost
C01	B01	02/13	2	Post	$3
C02	B03	01/10	3	D-to-D	$5
C03	B02	03/11	1	Post	$3
...	...		...	...	...

<Order_Info Table>

Name	Company	Number	Office	Country	City	Zip	credit_card
...	...	...	...	...	...	...	...

<Cust_Info Table>

Title	Publication	ISBN	Prime_cost	tax
...	...	...	...	...

<Book_Info Table>

4.2 Definition of Access Control Rules

Access control rules for the experiment are as follows.

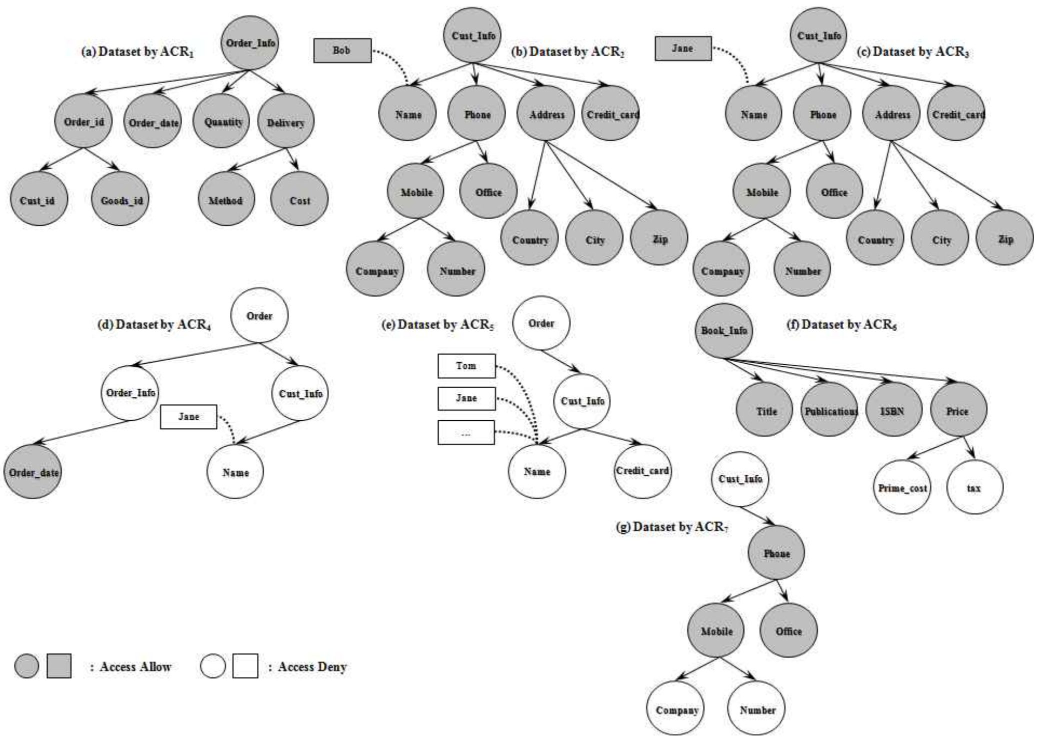
- ✓ ACR1 = (Tom, R+, R, Order/Order_info, *, *, *)
- ✓ ACR2 = (Bob, R+, R, Order/Cust_info, Name, Bob, *)
- ✓ ACR3 = (Jane, W+, R, Order/Cust_info, Name, Jane, *)
- ✓ ACR4 = (Jane, R+, L, Order/Order_info/ Order_date, Name, Jane, *)
- ✓ ACR5 = (Bob, R−, L, Order/Cust_info/ Credit_card, Name, not Bob, *)
- ✓ ACR6 = (Bob, R+, R, Book_info, *, *, 2)
- ✓ ACR7 = (Jane, R+, R, Order/Cust_info/ Phone, *, *, 2)

Described access control rules are represented as ACR=(Subject, Role, Type, Path, Attribue, Value, Level) form shown in the definition 3.

(Figure 5) describe the accessible dataset by the 7 access control rules. The access control rule 1 is the general recursive type access control rule. The access control rule is applied to the indicated node described in path tuple and all of descendant nodes. The access control rule 2 is also the recursive type access control rule with attribute value. The access control rule 2 is limited by a value of the 'Name' attribute. That is, this rule is applied when a value of the 'Name' attribute is 'Bob'. The access control rule 3 is similar to the access control rule 2. The access control rule 3 describes the positive rule about write role and is limited by a value of the 'Name' attribute. The access control rule 4 is the local type access control limited by an attribute value. The access control rule 5 is the local type negative access control rule. Generally, if the positive access control is not defined, that part is regarded as access denial. In addition, the negative access control rules are priority to the positive access control rules. In case of the access control rule 2 and 5, access control rules for 'Credit_card' attribute are duplicated. However, the access control rule 5 is priority to the access control rule because the access control rule 5 is the negative access control rule. The access control rule 6 is newly defined access control rules in this paper. The access control rule 6 is the recursive type access control rule with tree level specification. The tree level specification means that the access control rule is applied from the indicated node to descendant nodes of specified sub level. By using the concept of tree level specification, we can perform more detail access control about XML documents except for the local type and recursive type access control.

4.3 Experiment Results and Evaluation

For exmpeirment, we use 7 queries. If user quereis are given as XQuery form, the queries



(Figure 5) Accessible Dataset Part by the Access Control Rules are stored into the relational database. In addit ion, the recursive type user queries are change d into the local type user queries and combine d with access control rules by the query rewriti ng rules shown in the theorem 1 and 2.

If user queries are given as XQuery form, t he query lists are anlyzed and stored into the relational database as <Table 11>.

<Table 11> User Query Lists

Query_ID	Subj	Role	Path
1	Tom	R	Order/Order_Info//
2	Bob	R	Order/Cust_Info/Phone[Name='Bob]//
3	Tom	R	Order/Order_Info/Delivery[Name='Jane]//
4	Bob	R	Order/Cust_Info/Credit_card[Name='Jane]
5	Jane	W	Order/Cust_Info[Name='Bob]//
6	Bob	R	Order/Book_Info[Title='Les Miserables]//
7	Bob	R	Order/Book_Info/Price//
8	Jane	R	Order/Cust_Info/

The recursive type user queries are translat ed into the local type queries by the 1st query rewriting rules described in theorem 1 for effe ctive combination and comparison with the acc ess control rules.

In 2nd query rewriting, user queries are re written by the 2nd query rewriting rules show n in theorem 2. In case of query 1, the query 1 is equal to the scope of the access control r ule 1. Because the access control rule 1 is the positive rules, we do not need to rewrite the u ser query 1 in the 2nd query rewriting. The u ser query 2 is included in the scope of the ac cess control 2 and the user query 3 is include d in the scope of the access control 2. As a r esult, the user query 2 and 3 are not rewritten in the 2nd query rewriting.

The user query 4 is included in the scope o f the access control rule 5. Because the access control rule 5 is the negative rule, the user qu ery 4 is deleted. In case of the user query 5, t here is no access control rules which have the scope including the user query 5. Basically, if the access control rule is not defined for some node or part, the access control about that nod e or part is regarded as access denial. Therefo re, the user query 5 is also disposed in the 2n d query rewriting. A part of the user query 6 is included in the access control 6. The user q uery 6 includes 'Book_Info' node and all of de scendant nodes, but the scope of the access co ntrol rule 6 is from 'Book_Info' node to child nodes. Because the access control 6 is defined with level which value is 2, an intersection of the user query 6 and the access control 6 is e xecuted and then the user query 6 is rewritten by the theorem 2 in the 2nd query rewriting. A part of the user query 7 is also included in a scope of the access control rule 6. The user query 7 includes 'Price' node and descendant nodes, but the access control rule 6 is only in cludes 'Price' node. Therefore, the user query 7 is rewritten and 'Bob' can only access 'Pric e' node as a result of the intersection of the u ser query 7 and the access control rule 6.

As shown in the <Table 12>, all of user qu eries are rewritten or deleted by the theorem 1 and theorem 2. From user query 1 to user q uery6 can be processed by the Suggested Mod el illustrated in this paper and conventional X ML security models. However, access control process about user query 6, 7, and 8 can only be processed in the Suggested Model describe d in this paper. Through the processing of use r query 6, 7, and 8 by the access control rule 6 and 7, we can perform more detail access c ontrol about XML documents than conventiona l XML security models which only use the loc al or recursive type access control rules.

Query_ID	Subj	Role	Path
1_1	Tom	R	Order/Order_Info
1_2	Tom	R	Order/Order_Info/Order_id
1_3	Tom	R	Order/Order_Info/Order_id/Cust_id
1_4	Tom	R	Order/Order_Info/Order_id/Good_id
1_5	Tom	R	Order/Order_Info/Order_date
1_6	Tom	R	Order/Order_Info/Quantity
1_7	Tom	R	Order/Order_Info/Delivery
1_8	Tom	R	Order/Order_Info/Delivery/Method
1_9	Tom	R	Order/Order_Info/Delivery/Cost
2_1	Bob	R	Order/Cust_Info/Phone[Name='Bob]
2_2	Bob	R	Order/Cust_Info/Phone[Name='Bob]/Mobile
2_3	Bob	R	Order/Cust_Info/Phone[Name='Bob]/Office
2_4	Bob	R	Order/Cust_Info/Phone[Name='Bob]/Mobile/Company
2_5	Bob	R	Order/Cust_Info/Phone[Name='Bob]/Mobile/Number
3_1	Tom	R	Order/Order_Info/Delivery[Name='Jane]
3_2	Tom	R	Order/Order_Info/Delivery[Name='Jane]/Method
3_3	Tom	R	Order/Order_Info/Delivery[Name='Jane]/Cost
6_1	Bob	R	Order/Book_Info[Title='Les Miserables]
6_2	Bob	R	Order/Book_Info[Title='Les Miserables]/Title
6_3	Bob	R	Order/Book_Info[Title='Les Miserables]/ Publication
6_4	Bob	R	Order/Book_Info[Title='Les Miserables]/ISBN
6_5	Bob	R	Order/Book_Info[Title='Les Miserables]/Price
7_1	Bob	R	Order/Book_Info/Price
8_1	Jane	R	Order/Cust_Info
8_2	Jane	R	Order/Cust_Info/Name
8_3	Jane	R	Order/Cust_Info/Phone
8_4	Jane	R	Order/Cust_Info/Phone/Mobile
8_5	Jane	R	Order/Cust_Info/Phone/Office
8_6	Jane	R	Order/Cust_Info/Address
8_7	Jane	R	Order/Cust_Info/Address
8_8	Jane	R	Order/Cust_Info/Address
8_9	Jane	R	Order/Cust_Info/Address
8_10	Jane	R	Order/Cust_Info/Credit_card

<Table 12> Rewritten User Query Lists

<Table 13> Evaluation of User Queries

Query	Access Result	2nd Rewriting
Q1	FA	None
Q2	FA	None
Q3	FA	None
Q4	D	Deleted
Q5	D	Deleted
Q6	PA	Rewritten
Q7	PA	Rewritten
Q8	PA	Rewritten

FA: Fully Allow, D: Deny, PA: Partially Allow

As shown in <Table 13>, query 1, 2, and 3 are included in the scope of the positive acc ess control rules. Therefore, access control res ults of these queries are fully access allowanc e and we do not need to rewrite the queries i n the 2nd query rewriting. query 4 and 5 are included in the negative access control rule or not defined in access control rule lists. Theref ore, access control results of query 4 and 5 ar e access denial and we delete the queres in th e 2nd query rewriting. User query 6, 7, and 8 are partially included in the scope of the positi ve access control rules. Thus, access control r esults of query 6, 7, and 8 are partially access allowance and we rewrite queries by the combination with access control rules.

<Table 14> Comparative Evaluation

Query	Native XML Model	Relational DB-based Model	TL-BAC
1	Support	Support	Support
2	Support	Support	Support
3	Support	Support	Support
4	Support	Support	Support
5	Support	Support	Support
6	Not Support	Not Support	Support
7	Not Support	Not Support	Support
8	Not Support	Not Support	Support

<Table 14> shows the comparative evaluation about processing of the given queries. The native XML security model and Relational DB-based security model can support access control from query 1 to 5 as the access allowance or denial. Query 1, 2, and 3 are related to the recursive or the local type positive access control rules. Therefore, access about query 1, 2, and 3 is allowed and conventional security models and proposed Suggested Model can support access control abou query 1, 2 , and 3. In addition, query 4 and 5 are related to the recursive or the local type negative access control rules. Thus, access about query 4 and 5 is denied and conventional security models and proposed Suggested Model can support access control abou query 4 and 5. However, in case of query 6, 7, and 8, these queries are related to access control rules considering XML tree level. Native XML security models and relational DB-based security models cannot define access control rules as the Suggested Model. Therefore, native XML security models and relational DB-based security models cannot support access control processing about the given user queries.

5. Conclusion

In this paper, we suggested the RDB-based XML access control model considering XML tree levels, Suggested Model. We envisage an XML data management system in which (1) user make XML queries against a given XML Schema; (2) access control rules for XML data are specified in a relational database; (3) XML data are stored into a relational database. (4) access control process is performed considering XML tree levels. The TL-BAC system suggested in this paper can have following contribution.

- ✓ Practicality: the TL-BAC system can support more practical access control processing by using relational database, still widely used, for storing XML data.
- ✓ Stability: through the TL-BAC system adopt RDB access control techniques, already researched and practically used, to XML access control, the TL-BAC system guarantees better stability than conventional XML access control models.
- ✓ Performance: because the TL-BAC system stores XML data into the relational database, when user queries are given, we do not need to load all of XML documents. In addition, because XML data, XML access control rules, and user query are stored into the relational database with similar storage schema, processing performance for query processing is better than conventional XML access control models
- ✓ XML Tree Level-based Access Control: Through adoption of concept about XML tree levels, the TL-BAC system can support more detail access control that conventional recursive type or local type access control rules cannot support.

As a future works, we need to perform the experiment with practical XML data in commercial XML database.

References

[1] T. Bray et al., (Eds), Extensible Markup Language (XML) Version 1.0, W3C Recommendation, October, 2000.

[2] E. Damiani et al., A Fine-Grained Access Control System for XML Documents, ACM Trans. On Information System Security (TIS-SEC), Vol. 5, No. 2, May 2002.

[3] E. Damiani et al., Design and Implementation of an Access Control Processor for XML Document, Computer Netowkrs, Vol. 33, No. 6, June 2000.

[4] E. Bertino et al., Secure and Selective Dissemination of XML Documents, IEEE Trans. On information and System Security (TISSEC), Vol. 5, No. 3, August 2002.

[5] S. Godik and T. Moses (Eds), eXtensible Access Control Markup Language (XACML) Version 1.0, OASIS Specification Set, February 2003, http://www.oasis-open.org/ committees/xacml/repository.

[6] U.M.Mbaanaso et al., Privacy Preserving Trust Authorization Framework Using XACML, International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 06), Buffalo, New York, June, 2006.

[7] K.L. Tan et al., Access Control of XML Documents in Relational Database Systems, Int'l Conf. on Internet Computing (ICIC 01), Las Vegas, NV, June 2001.

[8] J. Jeon et al., Filter XPath Expressions for XML Access Control, Computers & Security, 23, 2004

[9] B. Luo et al., Pragmatic XML Access Control using Off-the-Shelf RDBMS, 12th European Symposium On Research In Computer Security (ESORICS 2007), Dresden, Germany, September 2007.

[10] B. Luo et al., QFilter: fine-grained run-time XML access control via NFA-based query rewriting, International Conference on Information and Knowledge Management (CIKM 2004), Washington, DC, USA, November 2004.

[11] C. Byun et al., An Efficient Query-based XML Access Control Enforcement Mechanism, KIISE Journal: Database, Vol. 34, No. 1, February 2007.

[12] P. Samarati et al., An Authorization Model for a Distributed Hypertext System, IEEE Trans. On Knowledge and Data Engineering (TKDE), Vol. 8, No. 4, August 1996.

[13] M. Kudo et al., XML Document Security Based on Provisional Authorization, ACM Conf. on Computer and Communications Security (CCS), Athens, Greece, November 2000.

[14] I. Fundulaki et al., Specifying Access Control Policies for XML Documents with XPath, ACM Symposium on Access Control Models and Technologies (SSACMAT), Yorktown Heights, US, June 2004.

[15] E. Damiani et al., Securing XML Document, 7th International Conference on Extending Database Technology (EDBT 2000), Konstanz, Germany, March 2000.

[16] Y. Xiao et al., Security-Conscious XML Indexing, International Conference on Database Systems for Advanced Applications (DASFAA 07), Bangkok, Thailand, April 2007.

[17] M. Jiang et al., Integration and Efficient Lookup of Compressed XML Accessibility Maps, IEEE Trans. On Knowledge and Data Engineering (TKDE), Vol. 17, No. 7, July 2005.

[18] W. Fan et al., Secure XML Querying with Security Views, ACM SIGMOD, Paris, France, June 2004.

[19] M. Murata et al., XML Access Control Using Static Analysis, ACM trans. Information Systems and Security, Vol. 9, No. 3, August 2006.

[20] N. Qi et al., Access-Condition-Table- Driven Access Control for XML Databases, 9th European Symposium on Research Computer Security (ESORICS 04), Sophia Antipolis, France, September 2004.

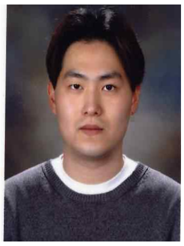
[21] N. Qi et al., XML Access Control with Policy Matching Tree, 10th European Symposium on Research in Computer Security (ESORICS 05), Milan, Italy, September 2005.

[22] S. Mohan et al., Ipac: and Interactive Approach to Access Control for Semi-structured Data, International Conference on Information and Knowledge Management (CIKM 05), Bremen, Germany, October 2005.

[23] J. Kim et al., Formal Verification ofthe Value Pattern-based Translation Algorithm, Dynamics of Continuous, Discrete and Impulsive Systems (DCDIS) Series B, Vol. 3, pp. 1359-1363, June 2007.

[24] J. Kim et al., Formal Verification and Quantitative Evaluation of QP-T Algorithm, Dynamics of Continuous, Discrete and Impulsive Systems (DCDIS) Series B, Vol. 3, pp. 1369-1373, June 2007.

[25] J. Kim et al., VQT: Value Cardinality and Query Pattern-based R-Schema to XML Schema Translation with Implicit Referential Integrity, Journal of Zhejiang University-Science A (JZUS-A), Vol. 9, No. 10, November 2008.



김 진 형

2004년 :홍익대학교 컴퓨터학과 (학사)
2006년 :고려대학교 대학원 (이학석사-전산학)
2008년 :고려대학교 대학원 (이학박사 수료-전산학)
2008년~현재 : 고려대학교 컴퓨터정보통신연구소
관심분야 : XML 데이터베이스, XML 보안, 시멘틱 웹, 유비쿼터스 컴퓨팅

정 동 원

1997년 :군산대학교 컴퓨터학과 공학사
1999년 :충북대학교 대학원 (공학석사-전산학)
2004년 :고려대학교 대학원 (이학박사-전산학)
2005년 : Visting Research Scholar, Pennsylvania State Univ.
2005년~2007년 : 군산대학교 정보통계학과 전임강사
2007년~현재 : 군산대학교 정보통계학과 교수
관심분야 : 데이터 통합, XML 데이터베이스, XML 보안, 시멘틱 웹, 유비쿼터스 컴퓨팅, 시멘틱 센서 네트워크, 시멘틱 GIS, 시멘틱 Grid

백 두 권

1974년 :고려대학교 수학과 이학사
1977년 :고려대학교 대학원 (공학석사-산업공학)
1983년 :Wayne State University (이학석사 - 전산학)
1986년 :Wayne State University (이학박사 - 전산학)
1986년~현재 : 고려대학교 컴퓨터전파통신공학부 교수
1989년~현재 : 한국정보과학회 이사/평의원/부회장
1991년~현재 : 한국시뮬레이션학괴 이사/감사/부회장/회장/고문
1991년~현재 : ISO/IEC JTC1/SC32 전문위원회 위원장
1997년~1998년 : 고려대학교 정보전산원 원장
2002년~2004년 : 고려대학교 정보통신대학 학장
2003년~2004년 : 한국정보처리학회 부회장
관심분야 : 데이터공학, 소프트웨어공학, 모델링과 시뮬레이션