Efficient Storage Construction for Semi-Structured Microarray Data Exploiting Structural Similarity

Dongkyoo Shin and Dongil Shin

Department of Computer Engineering, Sejong University
98 Gunja-Dong, Gwangjin-Gu, Seoul 143-747, Korea
{shindk, dshin}@sejong.ac.kr

Abstract

To promote molecular biology studies, public repositories for microarray data need to be constructed; the minimum contents for analysis of microarray experiment have been defined and standardized. Public repositories have been constructed by some researches which follow the standards such as MIAME-compliant data and MAGE-OM/ML. However, enough consideration has not been taken into the design of storage structure for the hierarchy of microarray data. In this paper, we propose alternative mapping strategy to mine the structural similarity and an advanced mapping rule from the algorithm. Object-relational mapping technique is used for extracting advanced storage design schema for microarray data and structural similarity of elements is evaluated for efficient storage construction. The mapping strategy reduced the number of relational tables remarkably. The strategy will contribute to design of the storage structure of microarray data and performance enhancement of a public repository.

Keywords: *structural similarity, decision tree, microarray database, XML, schema mining*

1. Introduction

To promote molecular biology studies, microarray data must be shared among researchers. Hence, the minimum contents for analysis microarray experiment have been defined and standardized. With the definitions, a standard protocol such as MAGE-ML (Microarray Gene Expression - Markup Language) has been announced to make microarray contents exchangeable among the application programs of the research groups. To promote the interoperable foundation that enables microarray data to be exchanged in application level, it is necessary to construct infrastructure complying with the standard. Researchers have paid their efforts to construct public repositories that follow the standard. As a result, the implementation of the standard-compliant storage has become the main issue of related research [1, 2, 3].

Even though public repositories have been constructed successfully, not enough consideration has been given to the design of storage structure for the hierarchy of microarray data. For advanced database design, DTD-dependent database design scheme was introduced by many researchers. This scheme supports hierarchy-oriented searching system and perfect parsing method for XML (eXtensible Markup Language) documents because DTD (Document Type Definition) reflects the hierarchy of a structured data. In the research, RDBMS (Relational DataBase Management System) is used as XML storage because it has the advantages on storage space and speed. The main idea is how to preserve the structural integrity of semi-structured data in RDBMS.

Two obstacles to achieving the idea exist. The first is difficulty in mapping object data into relational data, and the other is deterioration of database performance that is involved in the mapping process. To overcome these difficulties, several XML storage methods were proposed [4, 5, 6]. These approaches reduced the complexity of DTD or XML instance prior to the creation of database schema. They analyzed a graph that represents XML elements and attributes, and edges which represents relationship between parent and children elements.

In this paper, we propose an algorithm to mine the structural similarity and an advanced mapping rule from the algorithm. Object-relational mapping technique is used for extracting advanced storage design schema for microarray data in our method, but unlike previous works structural similarity of elements is evaluated for efficient storage construction. Using the proposed algorithm, we designed a MAGE-compliant database system and evaluated its performance.

2. Background Study

For the purpose of managing semi-structured data, commercial RDBMSs provide the object-relational mapping methods [4, 7, 8]. Commonly, these methods treat XML documents as a tree form of objects and transform it into several relational tables. This makes queries to XML data set have a number of joins. As a result, the performance of a database adopting this method is potentially deteriorated [4]. To overcome this problem, several studies have proposed alternative XML storage mapping techniques: DTD-dependent [4], Edge-dependent [5], and data mining-dependent [6]. The research [5] stores graph and edges in a single Edge table to handle all graph and edge information of XML data. The research [6] proposed STORED system adopted a data-mining algorithm [9] to extract relations from XML data and then transform it to relational database schema. These approaches [5, 6] remarkably improved the database performance but only one structure can be handled because they require only an instance of XML data in the transformation process. In managing the structural variety of an XML instance, simplifying DTD would be the fundamental measure. The research [4] proposed in-line technique focusing on simplifying DTD prior to object-relational mapping. This technique eliminated omissible elements from DTD. Where, the omissible elements are the elements with only a role of a linker between an ancestor element and a descendant one. By removing such elements, ancestors can be directly linked to descendants.

Many MGED (Microarray Gene Expression Data) standard-compliant repositories have adopted RDBMS, which enables the repositories to take advantage of the RDBMS engine for its stability, convenient management, and good performance [1, 2]. The goal of the research is to implement storage following the MGED standard. Although the research [1] improved performance of common queries through local modification of object model, it is difficult to expect the efficient database design scheme from it because its database design scheme adheres to the straightforward mapping technique. Similarly, the research [2] did not provide an answer for understanding the object-relational impedance mismatch [10], which occurs when tree-based XML documents are stored in the standardized relational database tables. Although the research [3] presented the concrete mapping rule to transform MAGE objects into XML documents, it aimed to export XML documents from relational tables rather than to consider the advances in the efficient design of relational database for microarray data set.

2.1. Microarray Data and MAGE (Microarray Gene Expression)

By sharing microarray experimental information, research groups can acquire the various experimental techniques performed by others and use them to solve their own biological questions. For necessity, the MGED society [11] has been leading several working groups to develop standards. MIAME (Minimum Information About a Microarray Experiment), one of the workgroups, has defined the necessary data for analyzing microarray experimental. Another workgroup, MAGE (Microarray Gene Expression) has announced the standard protocols for the exchange of MIAME data: MAGE-OM/ML (Object Model/Markup Language) [12]. The MAGE-OM is an object-oriented class diagram stated by UML (Unified Modeling Language). MAGE-OM has a complex and huge hierarchy consisting of 132 classes, 123 kinds of attributes, and 223 kinds of associations. MAGE-ML also has a structural complexity that has made its format impractical and its creation impossible by hand [13, 14]. The structures of MAGE-ML defined in DTD have similar structure patterns. This feature allowed us to establish an efficient storage construction scheme for microarray data sets.

2.2. Decision Tree

The decision tree that makes rules for classifying the patterns is to be determined in a set of attributes. The expression of the decision tree is very simple but its classifier has good accuracy [15]. The decision tree is constructed by calculating information gain of each attribute. For example, if a training set of classes, D is $\{C_1, C_2, \ldots, C_m\}$, the information gain of the training set is given by

$$Info_{(D)} = -\sum_{i=1}^{m} P_i \log_2(P_i) \qquad (1)$$

where, p_i is the probability that an arbitrary tuple in D belongs to class C_i.

The information gain of an attribute with different values such as $\{a_1, a_2, \ldots, a_v\}$ is given by

$$Gain_{(A)} = Info_{(D)} - \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info_{(D_j)} \qquad (2)$$

The decision tree has been widely adopted in various research fields such as document classification, detection of illegal intrusion, and classification of protein and diseases [16, 17].

3. Design of Decision Tree for Establishing Classification Rules

Processes show how classification rules are established. First, we define the structural similarity among elements. Terminology is defined as well. Second, we select the splitting criterion for constructing the decision tree. Finally, we construct the decision tree from which classification rules are extracted. We state algorithms for realizing those rules as well.

3.1. The Basic Object-Relational Mapping Technique for XML Storage

For illustration purpose, we introduce an example XML schema that shows how the classification rules apply, as shown in Figure 1. The classification rules also perform well on MAGE-compliant data.

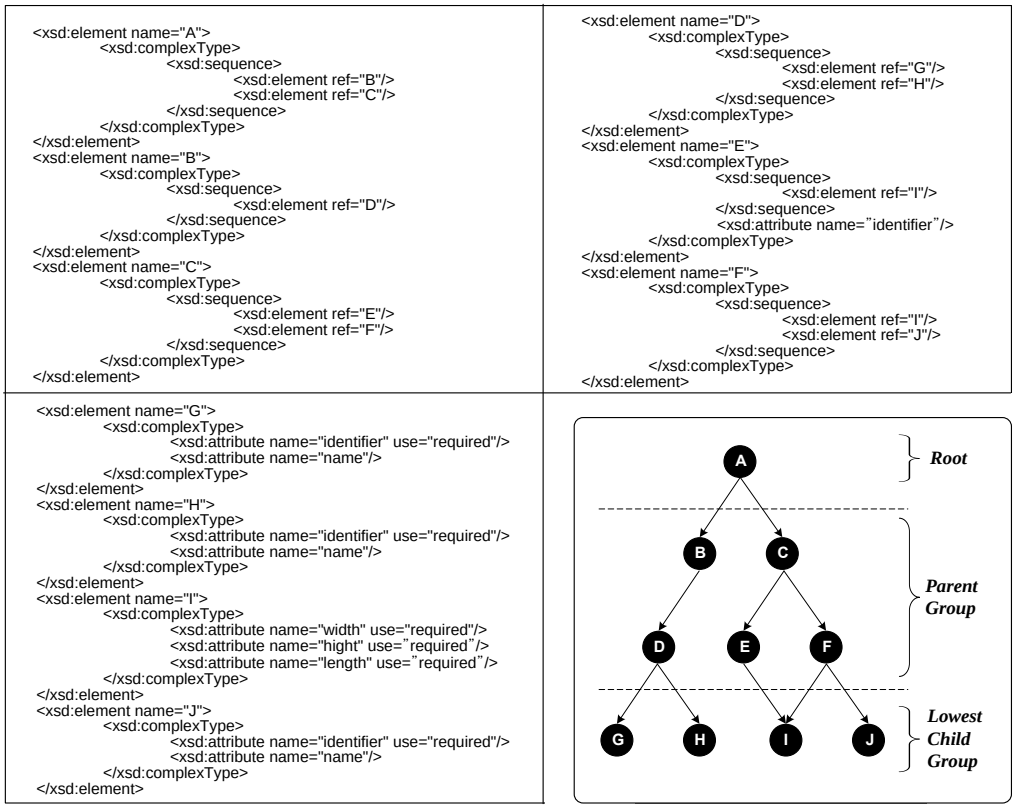


Figure 1. The Structural Expression using Tree Nodes and Its Example XML Schema

In the example, the basic principle of object-relational mapping technique is briefly represented. All elements are converted to classes. Among them, certain classes create its instance and have a child element or attribute that is transformed into relational tables. Figure 2 shows a part of such tables created from elements A to J. All fields of class A have a value referencing an external object. This shows the relation between classes A and J. To maintain the hierarchy of the example schema, each table exploits a primary key and several foreign keys.

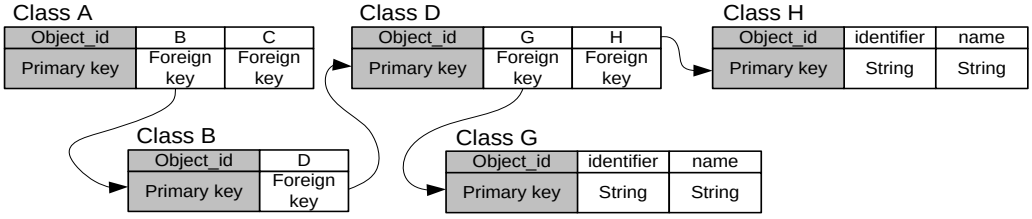


Figure 2. Tables Derived from complexType from A to H

3.2. The Definition of Structural Similarity among Elements

As shown in Figure 1, every element has a *complexType* that defines components such as sub-element(s) and attribute(s). Each component can be expressed as a part of a set with properties. Assuming that a certain element *x* has a *complexType*, each complement set of *x* can be defined as follows:

$$SE_x = \{e_1, e_2, \ldots, e_n\},\ SA_x = \{A_{e1}, A_{e2}, \ldots, A_{en}\}$$
$$complexType(x) = \{SE_x, SA_x\}$$

To help to understand the above summary, we defined some terminologies as follows:

- *e*: an element defined in XML schema
- *E*: an elements set of *e*
- *SE*: a sub-elements set of *e*
- *a*: an attribute of *e*
- *A*: an attributes set of *e*
- *SA*: an attributes set for all sub-elements of *e*
- *complexType*: Structural information that consists of *SE* and (or) *A* of *e*.
- *Lowest Child*: an element without any sub-element.
- *LCG (The Lowest Child Group)*: a set of lowest child elements
- *PG (Parent Group)*: a set of elements with sub-element.
- *Root*: an element without parent element.

Whether one *complexType* is similar to another *complexType* will depend on the comparison of sequence and attribute set between two complexType. For example, when *SEx* and (or) *SAx* in the *complexType* of a certain element *x* exactly match *SEy* and (or) *SAy* in the *complexType* of another element *y*, *x* and *y* have the structural similarity. Note that in evaluating structural similarity between a *complexType(x)* and a *complexType(y)*, the criterion of the similarity is data type not the name of each comparison object. Where the comparison object is each factor arranged in sub-elements set or attributes set.

3.3. Selection of Splitting Criterion

Element name, attribute, and child element are the factors of a *complexType* that define the structure of an element. In evaluating the structural similarity of elements, the most important factors are data types which are arranged in individual set of attributes sub-elements. We selected *hasParent*, *hasChild*, and *PG* as the test predicates for each *elementName*. These predicates are used in comparing structural information among *complexTypes*. The predicate *PG* decides whether an element belongs to *PG*

(Parent Group) class. For the construction of the decision tree, the training set of these items needs to be organized (Table 1). Table 1 presents the training set, *D*, of class-labeled tuples parsed from all *complexTypes* in the example schema of Figure 1.

Table 1. Class-labeled Training Tuples from an Example XML Schema

|Element|	hasParent	hasChild	Class: PG
A	False	True	Yes
B	True	True	Yes
C	True	True	Yes
D	True	True	Yes
E	True	True	Yes
F	True	True	Yes
G	True	False	No
H	True	False	No
I	True	False	No
J	True	False	No

In Table1, *PG* has two distinct values ({*yes*, *no*}); therefore, there are two distinct classes (that is, *m=2*). Let class C_1 correspond to *yes* and class *C2* correspond to *no* (refer to equation (1)). There are six tuples of class *yes* and four tuples of class *no*. A (root) node *N* is created for the tuples in *D*. To find the splitting criterion for these tuples, we compute the information gain of each attribute. We first use Equation (1) to compute the expected information needed to classify a tuple in *D*:

$$Info_{(D)} = -\frac{6}{10}\log_2\left(\frac{6}{10}\right) - \frac{4}{10}\log_2\left(\frac{4}{10}\right) = 0.971 \text{ bits}$$

Next, we need to compute the expected information requirement for each attribute: *hasParent*. We need to look at the distribution of *yes* and *no* tuples for each category of *hasParent* and *hasChild*. For the *hasParent*, category *false*, there is one *yes* tuple and zero *no* tuples. For the category *true*, there are five *yes* tuples and four *no* tuples. Using equation (2) the expected information needed to classify a tuple in *D* if the tuples are portioned according to *hasParent* is

$$Info_{hasParent(D)} = \frac{1}{10}\times\left(-\frac{1}{1}\log_2\frac{1}{1} - \frac{0}{1}\log_2\frac{0}{1}\right) + \frac{9}{10}\times\left(-\frac{5}{9}\log_2\frac{5}{9} - \frac{4}{9}\log_2\frac{4}{9}\right) = 0.891 \text{ bits}$$

Similarly, we can compute $InfohasChild_{(D)}=0$ bits. Hence, each of gains in information from such partitioning would be

- $Gain_{(hasParent)} = Info_{(D)} - Info_{hasParent(D)} = 0.971 - 0.891 = 0.08$ bits
- $Gain_{(hasChild)} = Info_{(D)} - Info_{hasChild(D)} = 0.971 - 0 = 0.971$ bits

Since hasChild has the highest information gain between the attributes, it is selected as the splitting attribute. Tuples in Table 1 can be partitioned as shown in Figure 3.

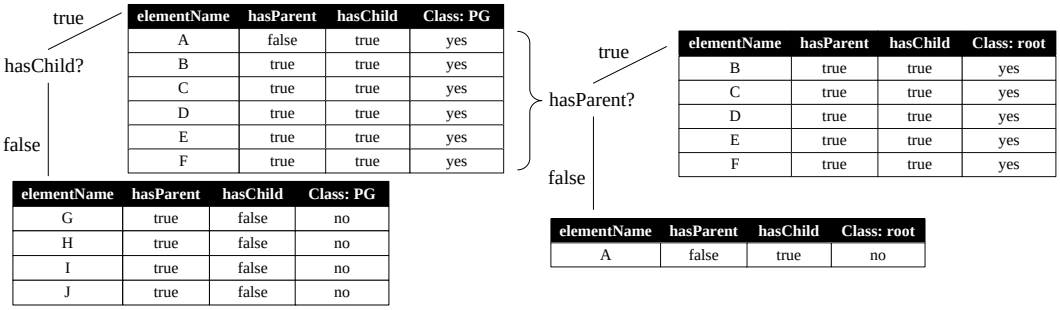


Figure 3. The Splitting Attribute at the Root Node of the Decision Tree

A further branching will be done in each partitioned class when an element is a parent element. Hence, a new splitting criterion for PG class needs to be calculated recursively. We can compute gains of *hasParent* and *hasChild* predicates: $Gain_{(hasParent)}$=0.971-0=0.971 bits and $Gain_{(hasChild)}$=0.971-0=0.971. Since they have the same information gain, one or the other can be selected as the splitting attribute. Briefly, this implies that the partitioned results must be same regardless of what the splitting attribute is. So we selected hasParent as the splitting attribute.

3.4. Classification Rules from the Decision Tree

Figure 4 shows the decision tree shaping idea for classifying elements with structural similarity. As an individual set of similar elements, each branching point has a condition by which an element is classified into a specific target node.

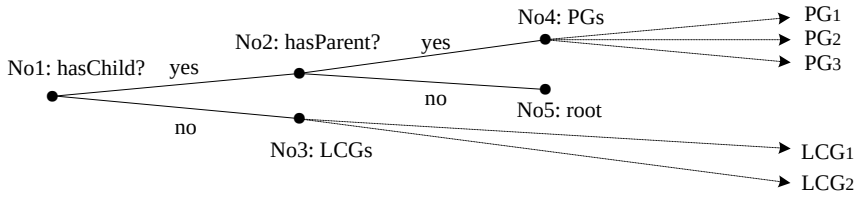


Figure 4. The Decision Tree from the Training Set of the XML Schema for Example

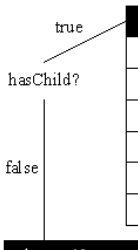
From each of branching points in the decision tree, we extracted rules for classifying elements with the structural similarity. Each of rules is expressed into *IF-THEN* statements respectively. Table 2 presents *IF-THEN* statements at each node.

Table 2. The Rules for Classifying LCGs, PGs, and Root Element

|No|	If	Then
2	hasChild=yes	Go to node No. 4
3	hasChild=no	LCGs
4	hasChild=yes AND hasParent=yes	PGs
5	hasChild = yes AND hasParent = no	root

Note that Node No. 3 and 4 are leaf nodes, which are sets of elements classified by

the conditions at No. 1 and 2, respectively. Each set of elements might be separated into several groups for parent elements and lowest child elements with structural similarity. Thus, it is required rules to evaluate the structural similarity of elements in No. 3 and No. 4. To simplify the comparison of structural similarity, we expressed *complexTypes* of elements to *Classification_codes* with string value.



elementName	hasParent	hasChild	Class:PG	Classification_code
A	false	true	yes	{[Obj, Obj], []}
B	true	true	yes	{[Obj, []}
C	true	true	yes	{[Obj, Obj], []}
D	true	true	yes	{[Obj, Obj], []}
E	true	true	yes	{[Obj], [@String]}
F	true	true	yes	{[Obj, Obj], []}

elementName	hasParent	hasChild	Class:PG	Classification_code
G	false	true	no	{[], [@String, @String]}
H	false	true	no	{[], [@String, @String]}
I	false	true	no	{[], [@String, @String, @String]}
J	false	true	no	{[], [@String, @String]}

Figure 5. Classification Code with the Decision Tree Example

A *Classification_code* shows a structure of an element as shown in Figure 5, where we attached the classification code to each training set tuple in the example decision tree of Figure 4. An element might have several sub-elements and attributes. Under this assumption, a *Classification_code* has two factors, one for sub-elements and the other for attributes. We can analyze the structural similarity among elements using these factors. The first factor arranges sub-elements set and has the fixed string value "Object". The value "Object" means a sub-element is managed in an external relational table. In other words, the first factor is a set of foreign keys referencing external relational tables. The second factor arranges data types built in XML schema, where, the symbol '@' used for indicating data type of the attribute. The following is the summary of a *complexType* of the element E (see Section 3.1).

- SE_E*={I},* SA_E*={identifier}*
- *complexType*$_{(E)}$*={[*SE_E*], [*SA_E*]}={[I], [identifier]}*

The above summary is simplified as follow:

- *Classification_code*$_{(E)}$*={[Object], [@String]}*

Note that, the sameness of *Classification_code* means that the compared elements have the structural similarity.

Using the *Classification_code*, we created *IF-THEN* statements at node No3 and No4 in Table 3. As shown in Table 3, at node No3 in Figure 4, *LPGs* might be divided into several groups according to the *Classification_Code*. Therefore, an algorithm to exactly expect the number of *LPGs* is required. Similarly, an algorithm for *PGs* is required at node No. 4.

Table 3. The Rules for Evaluating the Structural Similarity among Elements in Each Group

|No|	If	Then
3	hasChild=no AND hasParent=yes AND Classification_Code={@String, @String}	LCG1
	hasChild=no AND hasParent=yes AND Classification_Code={@String, @String, @String}	LCG2
4	hasChild=yes AND hasParent=no AND Classification_Code={Obj}	PG1
	hasChild=yes AND hasParent=no AND Classification_Code={Obj, Obj}	PG2

The following four rules are the comprehensive algorithms to seek the root element and expect the number of *LCGs* and *PGs*. For the implementation of the algorithms, all statements in Table 2 and 3 are generalized into four rules: *Rule1*, *Rule2*, *Rule3*, and *Rule4*. These rules are performed at node No. 1, No. 2, No. 3, and No. 4 respectively.

***Rule1*:** if an element has no sub-elements then the element is classified into *LCG*. Otherwise, the element is classified into *PG*. That is, *Rule1* decides that an element should belong to group *LCG* or *PG*. The following pseudo code expresses this rule.

```
For each e_i ∈ E {
    if(number of elements in SEe_i == 0){
        e_i is classified into LCG;
    }else{
        e_i is classified into PG;
     }
}
```

***Rule2*:** If a certain element in *PGs* has the parent element, then the element is classified into *PGs*. Otherwise, the element is the root element. That is, *Rule2* seeks the root element in *PGs*.

```
For each e_i ∈ PGs {
    if (e_i has parent) {
        e_i is classified into  PG_s;
    }else{
        e_i is Root
     }
}
```

Rule3: Let *PG* from Rule1 be PG_0. When several elements in PG_0 have *complexType* in which one or more attributes are defined, the elements that have the same *complexType* are separated into a new group PG_p $(p>0)$. If there is already a group PG_p with the same *complexType*, the element comes to the group PG_p. That is, *Rule2* classifies multiple sets of *PG*: $PG_1 \ldots PG_n$

```
p = 0;
For each e_i ∈ PG_0 {
  Flag=0;
If (p>0) {
   For q=1 to p
         If (complexType(e_i) == complexType(element in PG_q)) {
            e_i is classified into PG_q;
            Flag=1;
         }
   }
   If (Flag==0) {
      For each e_j ∈ PG_0
         if(complexType(e_i) == complexType(e_j) {
            p=p+1;
     e_i and e_j are classified into a new group of PG_p;
         }
    }
}
```

Rule4: Let *LCG* from *Rule1* be LCG_0. When several elements in LCG_0 have *complexType* in which one or more attributes are defined, the elements, which have the same *complexType*, are separated into a new group LCG_p $(p>0)$. If there is already a group LCG_p with the same *complexType*, the element comes to the group LCG_p. That is, *Rule2* classifies multiple sets of *LCG*: $LCG_1 \ldots LCG_n$

```
p = 0;
For each e_i ∈ LCG_0 {
  Flag=0;
If (p>0) {
   For q=1 to p
         If (complexType(e_i) == complexType(element in LCG_q)) {
            e_i is classified into LCG_q;
            Flag=1;
         }
   }
   If (Flag==0) {
      For each e_j ∈ LCG_0 {
         if(complexType(e_i) == complexType(e_j) {
            p=p+1;
            e_i and e_j are classified into a new group of LCG_p;
      }
    }
}
```

4. Database Implementation using the Proposed Decision Tree

Using the decision tree illustrated in the previous section, we reduced the complexity of XML schema for MAGE-ML and implemented database tables from it. The database

consists of three levels as the tree node in Figure 1: *root*, *PGs*, and *LCGs*. Although the associations among *LCGs* are clear, *PGs* have associations too complex to complete the relations among parent elements and lowest child elements. To simplify the complexity, we created the *BigbrotherType* table with 15 columns for *PGs* and *LCGs*. Since each of columns is allowed to have null value, the table can handle all relations possible to be occurred under the root element. In addition, it is possible to cover the mutual references between parent elements because all *PGs* have a foreign key referencing the *BigbrotherType* table. All columns in each table are for the attribute value except columns for foreign keys.

Table 4 shows the comparisons of the number of classes, tables, table joins, and database sizes among three schemas: *Raw Schema* from direct object-relational mapping, *Optimized Schema* from our proposed algorithm, *In-Lined Schema* from [4]. The number of tables produced by *Optimized Schema* is only 3% of the number of tables produced by *Raw Schema*. From comparison of the number of records, table join and database size between *Raw Schema* and *Optimized Schema*, we know that the number of tables affects performance of the database.

Table 4. The Complexity of XML Documents

	Raw schema	Optimized schema	In-Lined schema
Class	455	15	203
Table	455	15	203
Record	2012	156	-
Table join	1033	24	-
Database size (KB)	4270	192	-

As the mapping technique for the comparison with our algorithm, we chose the in-line technique [4], because it has provided related research results on the efficient storage approaches. We presented only the number of classes and tables created from in-lined schema in Table 4, since we can calculate them for the literature. The comparison of two techniques proves that the consideration of the structural similarity is very effective as a scheme for acquiring the optimized database design against complex and huge hierarchy such as microarray data set.

5. Conclusion

This paper proposed a new approach for the efficient XML storage structure for microarray data, exploiting the structural similarity of elements that usually belong to a complex type and comprise of sub-elements and attributes. Unlike previous works, which focused on finding omissible elements between ancestor and descendant elements and removing them from DTD or XML schema, we paid attention to the structural similarity of elements in MAGE-ML schema. Microarray data have very complex hierarchies that are organically linked to other hierarchies, and just a few specific core values repeatedly occur at similar structures in such hierarchies. We suggested an algorithm to extract core features that repeatedly occur in an XML schema for biological information and explained how to improve classification speed and efficiency by using decision tree algorithm. The results of the performance evaluation proved from our scheme show the proper way to construct XML storage for microarray data. Additionally, the constructed database and middleware supports transformation

from MAGE-ML file into the storage and vice versa by importing and exporting. Our scheme comply MAGE standard and improves relational database performance for microarray data.

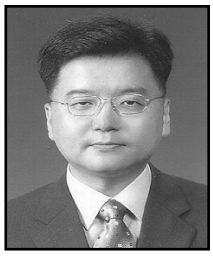
Acknowledgements

This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under IT/SW Creative research program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2012-H0502-12-1011

References

[1] U. Sarkans, H. Parkinson, G. G. Lara, A. Oezcimen, A. Sharma, N. Abeygunawardena, S. Contrino, E. Holloway, P. Rocca-Serra, G. Mukherjee, M. Shojatalab, M. Kapushesky, S. A. Sansone, A. Farne, T. Rayner and A. Brazma, "The ArrayExpress gene expression database: a software engineering and implementation perspective", Bioinformatics, vol. 21, no. 8, **(2005)**, pp. 1495-1501.
[2] C. A. Ball, I. A. B. Awad, J. Demeter, J. Gollub, J. M. Hebert, T. Hern, H. Jin, J. C. Matese, M. Nitzberg, F. Wymore, Z. K. Zachariah, P. O. Brown and G. Sherlock, "The Stanford Microarray Database accommodates additional microarray platforms and data formats", Nucleic Acids Research, vol. 33, **(2005)**, pp. 580-582.
[3] W. Martin and R. M. Horton, "Magebuilder: a schema translation tool for generating MAGE-ML from tabular microarray data", In 2003 IEEE Internation Conference on Computational System Bioinformatics Conference (CSB), **(2003)**, pp. 431-432.
[4] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. Detwitz and J. Naughton, "Relational databases for querying xml documents: Limitations and opportunities", In 25th International Conference on Very Large Data Bases (VLDB), **(1999)**, pp. 302-314.
[5] A. Schmidt, M. L. Kersten, M. Windhouwer and F. Waas, "Efficient relational storage and retrieval of XML documents", In ACM SIGMOD Workshop on the Web and Database (WebDB), **(2000)**, pp. 47-52.
[6] A. Schdmidt, M. F. Fernandez and D. Suciu, "Storing Semistructured Data with STORED", In ACM SIGMOD International Conference on Management of Data, **(1999)**, pp. 431-442.
[7] H. Schoning, "Tamino - A DBMS designed for XML", In 17th IEEE International Conference on Data Engineering (ICDE), **(2001)**, pp. 149-154.
[8] I. Tatarinov and S. D. Viglas, "Storing and Querying Ordered XML Using a Relational Database System", In ACM SIGMOD International Conference on Management of Data, **(2002)**, pp. 204-215.
[9] K. Wang and H. Liu, "Discovering typical structures of documents: a road map approach", In ACM SIGIR Conference on Research and Development in Information Retrieval, **(1998)** August, pp. 146-154.
[10] C. Ireland, D. Bowers, M. Newton and K. Waugh, "A Classification of Object-Relational Impedance Mismatch", In First International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA), **(2009)**, pp. 36-43.
[11] MGED – Microarray Gene Expression Data Society, http://www.mged.org.
[12] P. T. Spellman, *et al.*, "Design and implementation of microarray gene expression markup language (MAGE-ML)", Genome Biology, vol. 3, Issue 9, **(2002)**.
[13] T. Rayner, P. Rocca-Serra, P. T. Spellman, H. C. Causton, A. Farne, E. Holloway, J. Liu, D. S. Maier, M. Miller and K. Petersen, "A simple spreadsheet-based, MIAME-supportive format for microarray data: MAGE-TAB", BMC Bioinformatics, vol. 7, **(2006)**, pp. 489.
[14] http://www.mged.org/Workgroups/MAGE/MAGEdescription2.pdf.
[15] I. H. Witten and E. Frank, “Data Mining-Practical Machine Learning Tools and Techniques (Third Edition)”, Morgan Kaufmann Publishers, **(2011)**.
[16] R. C. Barros, M. P. Basgalupp, A. C. P. L. F. de Carvalho and A. A. Freitas, "A Survey of Evolutionary Algorithms for Decision-Tree Induction", IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. PP, Issue 99, **(2011)**, pp. 1-10.
[17] R. Hu and Y. Zhao, "Knowledge-Based Adaptive Decision Tree State Tying for Conversational Speech Recognition", IEEE Transactions on Audio, Speech, and Language Processing, vol. 15, Issue 7, **(2007)**, pp. 2160- 2168.

Authors



Dongkyoo Shin received a B.S. in Computer Science & Statistics from Seoul National University, Korea, in 1986, an M.S. in Computer Science from Illinois Institute of Technology, Chicago, Illinois, in 1992, and a Ph.D. in Computer Science from Texas A&M University, College Station, Texas, in 1997. He is currently a Professor in the Department of Computer Science & Engineering at Sejong University in Korea. From 1986 to 1991, he worked in Korea Institute of Defense Analyses, where he developed database application software. From 1997 to 1998, he worked in the Multimedia Research Institute of Hyundai Electronics Co., Korea as a Principal Researcher. His research interests include XML Security, XML based middleware, multimedia application, biological database, mobile Internet and ubiquitous computing.

Dongil Shin received a B.S. in Computer Science from Yonsei University, Seoul, Korea, in 1988. He received an M.S. in Computer Science from Washington State University, Pullman, Washington, U.S.A., in 1993, and a Ph.D. from University of North Texas, Denton Texas, U.S.A., in 1997. He was a senior researcher at System Engineering Research Institute, Deajun, Korea, in 1997. Since 1998, he has been with the Department of Computer Science & Engineering at Sejong University in Korea where he is currently a Professor. His research interests include Mobile Internet, Computer Supported Cooperative Work, Object-Oriented Database, Distributed Database, Data Mining and Machine Learning.