An Extended Dynamic Schema for Storing Semi-structured Data

Mitsuru NAKATA†, Qi-Wei GE†, Teruhisa HOCHIN†† and Tatsuo TSUJI††

†Faculty of Education, Yamaguchi University, Japan
1677-1 Yoshida, Yamaguchi-Shi Yamaguchi 753-8513, Japan
††Dept. of Information Science, Faculty of Engineering, Fukui University, Japan
3-9-1, Bunkyo, Fukui-Shi Fukui 910-8507, Japan
†Tel: +81-83-933-5402, Fax: +81-83-933-5402
E-mail: †{nakata, gqw}@inf.edu.yamaguchi-u.ac.jp, ††{hochin, tsuji}@pear.fuis.fukui-u.ac.jp

Abstract: Recently, database technologies have been used commonly. But, ordinary technologies aren't suitable to construct a complicated database such as a classical literature database or an archaeological relic's database. Because this kinds of data are semi-structured data that doesn't have regular structures, database schema can't be defined before data storing. We have proposed DREAM model for semi-structured databases. In this model, a database consists of five elements and the model has operations similar to operations of set theory. And further we have introduced dynamic schema "shape" showing structure of each element. We have already realized a prototype of DBMS adopting DREAM model (DREAM DBMS) and constructing function of shapes. However, shape is imperfect to describe database structures because it can't explain nested structures of elements. In this paper, we will propose a "shape graph" that is a dynamic schema showing database structures more exactly and extend the DREAM DBMS. Further we will evaluate the performance of constructing function of shapes and shape graphs.

1. Introduction

This paper proposes an extension of a dynamic schema, shape[1, 2], which represents semi-structured data and describes the implementation and evaluation of a function to construct dynamic schema. Database technologies have been widely used in many fields. However, ordinary technologies aren't suitable to construct a complicated database such as a classical literature database or an archaeological relic's database, because this kinds of data are semi-structured data that doesn't have regular structures and these structures are modified frequently. In other words, defining database schema (database structure) for these data is quite difficult before storing.

Some data models handling semi-structured data have been proposed[3, 4]. DREAM model[1, 2] is one of them. It is based on set theory and has operations similar to set operations. Although we can manage semi-structured data by using these data model without schema definition, knowing the database structure is required in order to search for information in a large amount of semi-structured data. Data Guides[5] and shape have been proposed to represent the structure of the semi-structured database. Data Guides and shape do not include the information on the data that does not exist in a database. These schemas are changed according to creation, modification and deletion of objects and are called dynamic schema here.

In DREAM model, there exist five kind of elements, such as named elements, perspectives and objects, called database elements. These elements will be explained in next section. Shape is the information about structure of each element and shows attributes of a corresponding element. However, the shape doesn't explain nested construction of each element. For example, even if one object consists of several perspectives, a shape of the object doesn't show which perspectives compose of it. Therefore, database users couldn't see general structure of database elements.

In this paper, we propose a data structure, called shape graph, trying to provide a more powerful representation of database structures. A shape graph is a tree consisting of shapes. Section 2. shows an outline of DREAM model and definitions of shape and shape graph. Section 3. gives extension of the original DREAM DBMS to manage shape graphs. And further we will evaluate the constructing function of shapes as well as shape graphs through experiments in Section 4..

2. DREAM Model

2.1 Database elements

DREAM model is a data model that supports semi-structured database management system (DBMS). A database consists of **data elements**, **named elements**, **perspectives**, **objects** and **bundles**. These are referred to as database elements. A data element is an element storing a data value. It is represented with a triplet $(id, type, d)$, where id is an identifier, $type$ is data type of the data value and d is a set of data values that has only one element. The named element is a triplet of its identifier, name and a set of data elements and/or objects. The object is a unit expressing an entity. It's represented with a triplet $(id, name, P)$, where id is an identifier, $name$ is a object's name and P is a set of perspectives. A perspective represents an aspect of an object. It's a triplet of its identifier, its name and a set of named elements. A set of objects can be managed in a bundle. The bundle is a triplet of its identifier, its name and a set of objects and/or bundles.

Figure 1 shows an example of a database. This is for

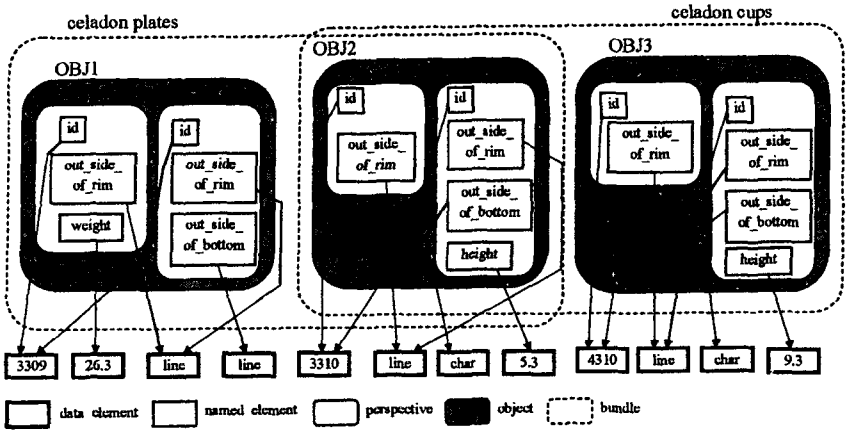

Figure 1. Example of a database

the pieces of the celadon cups and plates obtained from an archeological site. Each of the data values obtained by analyzing a data file can be put into a data element. A named element is created by giving a name to it. In Fig.1, there are two kinds of perspectives. Their names are "top (view)" and "both (view)". Three objects are created and there are two bundles in Fig.1. The bundle "celadon plates" holds two objects that were considered a plate. The bundle "celadon cups" holds two objects that were considered a cup. The OBJ2 couldn't be classified clearly, so it belongs to both bundles. Each object can consist of different perspectives and each perspective can hold different named elements. And furthermore, structure of these elements can be changed easily by using operations of DREAM model. Therefore, information of relics can be stored in a DREAM database even though they have different construction each other. And advance schema definition isn't needed.

2.2 Shape and shape graph

Shape is the information describing the form of data. This information includes names and data types. The fundamental unit of shape is a **shape entry** describing the structure of a named element. A shape entry is a triplet $(id, name, DT)$, where id is an identifier, $name$ is a name of a named element and DT is a set of data type names. Consider a named element $ne\_1 = (id\_1, "nm", N)$. The name of the shape entry for ne_1 is "nm". When N has data elements composed of primitive data such as "integer" and "float", these are elements of DT of the shape entry.

Shape describes the structure of perspectives, objects and bundles. It is a triplet $(id, name, S)$, where id and $name$ are an identifier and its name, and S is a set of shape entries for the named elements in those elements. Figure 2 shows the shape entries for the named elements of OBJ2 in Fig.1. And it also shows the shapes of OBJ2 and perspectives in OBJ2. The o38 is a new shape entry obtained from o30 and o32. Because there are two shape entries of which the name is "ID" in the OBJ2, these shape entries are merged to o38. The o39 is obtained from o31 and o33 similarly.

Shape entries of OBJ2
(o30, "ID", {int})
(o31, "out_side_of_rim", {string})
(o32, "ID", {int})
(o33, "out_side_of_rim", {string})
(o34, "out_side_of_bottom", {string})
(o35, "height", {float})

Shapes of perspectives in OBJ2
(o36, "top", {o30, o31})
(o37, "both", {o32, o33, o34, o35})

*Shape of OBJ2 (containing new two shape entries indicated with *1 and *2)*
(o38, "ID", {int}) $\cdots$ **1*
(o39, "out_side_of_rim", {string}) $\cdots$ **2*
(o40, "OBJ2", {o38, o39, o34, o35})

Figure 2. Shape entries and shape of OBJ2 and shapes of perspectives in OBJ2

Shape provides us information about attributes included in perspective, object and bundle. But it doesn't give us information about other elements composing the database element: for example, a bundle is composed of many objects and an object is composed of some perspectives. The information is very important to understand construction of databases. If users don't catch construction of their database, they might not be able to utilize theirs. Therefore, we propose **shape graph**. There are two kinds of shape graphs. One is the shape graph of an object that is a triplet $(id, s\_obj, \{s\_per\})$, where id is an identifier of a shape graph, s_obj is a shape of the object and $\{s\_per\}$ is a shape set of perspectives in the object. Another is the shape graph of a bundle that is a four-piece set $(id, s\_bndl, \{s\_per\}, \{s\_obj\})$, where id is an identifier, s_bndl is a shape of the bundle, $\{s\_per\}$ is a shape set of perspectives and $\{s\_obj\}$ is a shape set of objects in the bundle. For example, the shape graph of OBJ2 is a triplet $(o41, o40, \{o36, o37\})$. Figure 3 shows its structure. As showing in Fig.3, a shape graph is a tree consisting of shapes that correspond to the object and the perspectives.

3. Design and Implement

3.1 Storing database elements and shapes

Apply shape graph, we extend the original DREAM DBMS. The extended DREAM DBMS has been implemented by commercial Object Oriented Relational DBMS UniSQL release 5.0 on Compaq ProLiant ML350 server (CPU PentiumIII 600MHz, 256MB Memory, RedHat Linux release 6.1J).

Each database element such as a named element, shape, shape entry and shape graph (we call shapes, shape entries and shape graphs as SHAPES for short hereafter) are stored in a corresponding class (here class is table in an UniSQL database). Figure 4 shows classes of "named element" and "shape entry". A class "shape entry" to store shape entries has attributes "id", "name", "S" and "ID". An attribute "ID" is a set of identifier of database elements corresponding to the shape entry. The attribute "ID" of class of "Named element" is a similar attribute of "ID" of "Shape entry".

On the other hand, database operations in DREAM DBMS are provided as API(Application Program Interface) libraries of C language. These API libraries is referred to as **DREAM API**.

3.2 Utilizing shape graphs

Next, we describe how the shapes and shape graphs are used. In general database systems, users search and operate data on the basis of a database schema that consists of database name, table name, attribute names and data types and so on. But there isn't a database schema in DREAM model and each object has different structure. Users might know only database name and some bundle's name and might not know structure of each object. To manage database elements easily even if users don't know the details of database components, we provide graphical user interface implemented by DREAM API, Java and JNI[6]. Figure 5 shows structure of DREAM DBMS. And Figure 6 shows two windows of GUI. The left window is to explain a shape graph of a bundle "celadon_porcelain". As the figure shows, the bundle has named elements, which has name "out_side_of_rem", "height", "radius" and so on. The condition to search objects, of which radius is 6cm ($radius = 6$), is indicated on the left side tree of the window. And the right side tree is a result of the searching. As it shows, there is one object, which satisfies the condition. The right window explain the detail of the object looked by the searching.

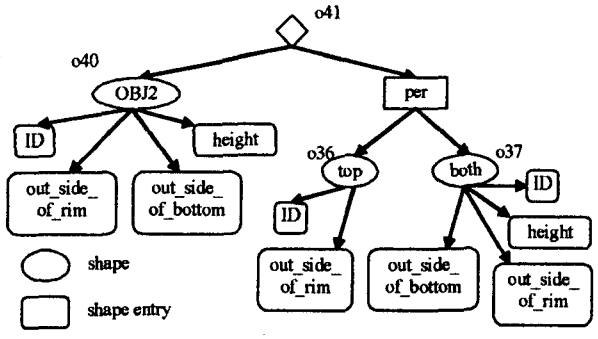

Figure 3. The shape graph of OBJ2

3.3 Updating shapes and shape graphs

SHAPES should be reconstructed when an update operation, such as insertion, modification or deletion, is executed. For example, a shape and a shape graph of a bundle should be reconstructed when an object is added into a bundle. Because it takes long time in reconstructing all SHAPES, we introduce two new classes. The first is a class "update_db_element" storing histories of operations. The second is a class "remake_shape" storing the database elements related to shape entry, shape and shape graph that should be reconstructed. By these two classes, the system needs only to change SHAPES that should be reconstructed.

4. Evaluation

We have implemented the extended DREAM DBMS and constructing function of shapes and shape graphs on it. To evaluate the constructing function, we have measured run time taking in constructing shapes and shape graphs for the Web contents database of Yamaguchi University, Japan. Because web contents data can be collected automatically and easily, we use these data as the sample of semi-structured data. The run time is measured every 200 objects under the following two conditions.

Condition 1 To totally reconstruct SHAPES for 200, 400, 600, 800, 1000 objects.

Condition 2 To reconstruct a part of SHAPES that should be reconstructed, when 150, 350, 550, 750, 950 objects have been stored and 50 objects are inserted.

In the experiment under condition 2, information stored in the classes "update_db_element" and "remake_shape" that is used to identify shapes should be reconstructed. Table 1 and Figure 5 show the times to reconstruct SHAPES every 200 objects. The data in Table 1 show the average of 10 times' executions. When stored 1000 objects, constructing all shapes takes 62.27 seconds. On the other hand, constructing partial shapes that should be reconstructed takes only 16.8 seconds. From these results, it is clear that adopting the data of

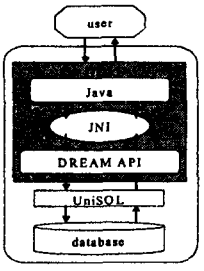

Figure 5. Structure of DREAM

Named element

id	name	S	ID
...	...	...	...
o100	"ID"	{o1}	o30
o101	"out side of rim"	{o2}	o31
o203	"ID"	{o3}	o30
...	...	...	...

Shape entry

id	name	S	ID
...	...	...	...
o30	"ID"	{int}	{o100, o203}
o31	"out_side_of_rim"	{string}	{o101, o204}
...	...	...	...

Figure 4. Class "named element" and class "shape entry"

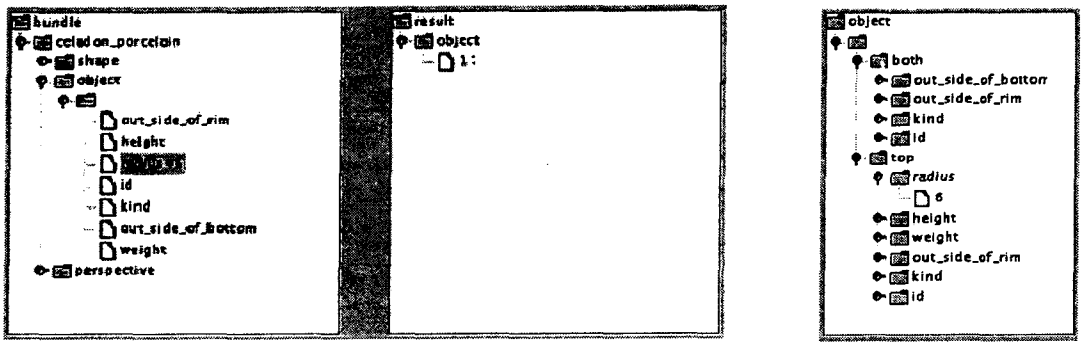

Figure 6. Windows of DREAM GUI

Table 1. Results of measurements

Condition 1		Condition 2	
Number of objects	Time to derive shape (sec)	Number of objects	Time to derive shape (sec)
200	10.4	150 → 200	8.02
400	13.71	350 → 400	9.48
600	31.78	550 → 600	12.3
800	46.86	750 → 800	14.51
1000	62.27	950 → 1000	16.8

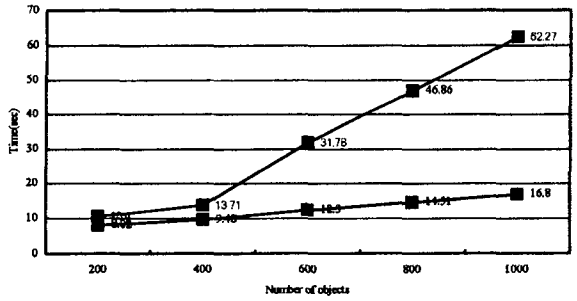

Figure 7. Times to construct SHAPES

these two classes to reconstruct SHAPES is an efficient method.

5. Concluding remarks

We have proposed shape graph that can express nested structure of database elements. It consists of a shape of a corresponding database element and one or two sets of shapes of nested elements. By the GUI showing shape graphs, users can operate databases even if they don't grasp a database schema. Further we have evaluated constructing function of SHAPES by measuring run time of constructing SHAPES. The experimental results show that our method can reconstruct SHAPES efficiently.

As the future works related to realization of the DREAM, we need to (i) complete the API libraries and GUI of DREAM; (ii) improve run time for updating operations and constructing SHAPES; and (iii) provide the database query and manipulation language such as SQL.

References

[1] Hochin, T. and Tsuji, T., "A Method of Constructing Dynamic Schema Representing the Structure of Semistructured Data", *Proc. of Int'l Database Engineering & Applications Symposium 99*, pp.103-108, 1998.

[2] Nakata, M., Hochin, T., and Tsuji, T., "Bottom-up Scientific Databases Based on Sets and Their Top-down Usage", *Proc. of Int'l Database Engineering & Applications Symposium 97*, pp.171-179, 1997.

[3] Buneman, P., et al, "Adding Structure for Unstructured Data", *Proc. of the 6th Int'l Conf. on Database Theory*, pp.336-350, 1997.

[4] Papakonstantinou, Y., Garcia-Molina, H., and Widom, J., "Object Exchange Across Heterogeneous Information Sources", *Proc. of 11th International Conference on Data Eng.*, pp.251-260, 1995.

[5] Goldman, R. and Widom, J., "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases", *Proc. of the 23rd VLDB Conf.*, pp. 436-445, 1997.

[6] Rob Gordon, *Essential Jni: Java Native Interface (Essential Java)*, Prentice Hall, 1998.