An Extension to DREAM Model Based Dynamic Schema for Semi-structured Data

Mitsuru Nakata, Qi-Wei Ge
Faculty of Education, Yamaguchi University, Japan
1677-1 Yoshida, Yamaguchi-Shi Yamaguchi 753-8513, Japan

Teruhisa Hochin and Tatsuo Tsuji
Dept. of Information Science, Faculty of Engineering, Fukui University, Japan
3-9-1, Bunkyo, Fukui-Shi Fukui 910-8507, Japan

Abstract

Although database technologies have been used widely, they still can not be simply used to construct such complicated database like classical literature database or archaeological relics' database. This is because these kinds of data are semi-structured data that do not have regular structures, so that their database schema can't be defined before storing data. We have proposed DREAM model for semi-structured databases. In this model, a database consists of five elements and its operations are similar to that of set theory. And furthermore we have introduced dynamic schema "shape" showing structure of each element. Based on this model, we have realized a prototype of database management system, called DREAM DBMS, including the function of constructing shapes, called shape-function. However, shape is insufficient to describe database structures because it can't explain nested structures of elements. In this paper, we refine the concept "shape" by introducing a new concept "shape-graph" that is also a dynamic schema showing database structures more exactly. Then we describe the implementation of DREAM DBMS. Finally, we evaluate the performance of constructing shape and shape-graph in order to reveal the efficiency of DREAM DBMS.

Keywords: Database management system, Semi-structured data, Dynamic schema, Data model

1 Introduction

Although database technologies have been used widely, they still can not be simply used to construct such complicated database like classical literature database or archaeological relics' database. This is because these kinds of data are not only modified frequently but also semi-structured data that do not have regular structures, so that their database schema can't be defined before storing data. Therefore, developing new database technologies to handle such kinds of data becomes important. Recently, studies on handling various types of data have been extensively investigated [1]-[9]. In all these studies, the data structure handled cannot be decided beforehand. For such semi-structured data, some data models have been proposed [2]-[6] including our model, DREAM model [10,11,12], in each of which the data is stored without defining schema. DREAM model is based on set theory and has the operations similar to set operations. A database management system adopting DREAM model is called DREAM DBMS.

Although we can store semi-structured data by using these data models without defining schema, searching for the data from a large amount of semi-structured data does need their data structure. To solve this problem, Data Guides [9] and shape [10,11,12] have been proposed to represent the structure of the stored semi-structured database. Data Guides and shape only represent the stored data but do not have any information about deleted or non-existent data. In this meaning, the defined structures of stored data can be called *dynamic schema*.

In DREAM model, there exist five kinds of elements, data element, named element, perspective, objects and bundle, and these elements are called database elements. These elements will be explained in detail in the next section. Shape is the concept proposed trying to represent the form of database elements, but it cannot represent nested structure of database elements. For example, even if one object consists of several perspectives, a shape of this object cannot show which perspectives it is composed of. Thus, the user of our database couldn't grasp the whole structure of the database elements. Therefore, we need to refine the concept of shape to propose a new concept in order to improve our model.

In this paper, we are to refine the concept "shape" by introducing a new concept "shape-graph", trying to provide a more powerful representation for database structures. Further, we describe the implementation of DREAM DBMS and show the evaluation result on efficiency of constructing dynamic schema. Section 2 shows an outline of DREAM model and definitions of shape and shape-graph. Section 3 gives the description of the implementation of DREAM DBMS. Section 4 gives the evaluation result.

2 DREAM Model

2.1 Database elements

DREAM model is a data model that supports semi-structured database management system. A database designed by DREAM model consists of 5 kinds of database elements, "data element", "named element", "perspective", "object" and "bundle". For the operations of DREAM model, refer to [11,12]. Database elements are defined in the following Definition 1.

Definition 1. A data element represents a data by a triplet (*id, type, d*), where *id* is an identifier, *type* is data type of the data and *d* is a set of values of the data, where *d* includes only one element $|d|=1$.

A named element is to give a name to a set of data elements and/or objects. It is a triplet (*id, name, S*), where *id* is an identifier, *name* is the name named to the set of data elements (objects) and *S* is the set of data elements (objects).

A perspective represents an aspect of an object. It is a triplet (*id, name, NE*), where *id* and *name* are similar to those of named element. Further, *NE* is a set of named elements related to the aspect in which there are no same names.

An object is a unit expressing one entity by a triplet (*id, name, P*), where *id* is an identifier, *name* is the object's name and *P* is a set of perspectives of the object in which there are no same names.

A bundle represents a set of objects and/or bundles. It is a triplet (*id, name, S′*), where *id* is an identifier, *name* is the name of the bundle and *S′* is the set of objects and/or bundles. A bundle is to manage some objects that have common property.

Example 1. Consider a cup obtained from an archeological site. This cup has one line on the outside of the rim and some characters on the outside of the bottom. Height of the cup is 5.3 centimeters. And further, the cup has serial number "3310". The information of this cup is stored by the following database elements.

Data elements:
(*e*1, *int*, {3310}),
(*e*2, *string*, {*line*}),
(*e*3, *string*, {*char*}),
(*e*4, *float*, {5.3})

Named elements:
(*e*5, "*id*", {*e*1}),
(*e*6, "*out_side_of_rim*", {*e*2}),
(*e*7, "*id*", {*e*1}),
(*e*8, "*out_side_of_rim*", {*e*2}),
(*e*9, "*out_side_of_bottom*", {*e*3}),
(*e*10, "*height*", {*e*4})

Perspectives:
(*e*11, "*top*", {*e*5,*e*6}),
(*e*12, "*both*", {*e*7,*e*8,*e*9,*e*10})

An Object:
(*e*13, "*OBJ2*", {*e*11,*e*12})

The object having name "OBJ2" expresses the cup. It has two perspectives. One is "top" expressing only the upper part of the cup, and another is "both" expressing the whole cup.

Figure 1 shows an example of a database. This database is for the pieces of celadon cups and plates, that are obtained from an archeological site. The data obtained by analyzing the pieces can be put into several data elements, such as done in the above example. In Fig.1, there are two perspectives with names "top" and "both". This database stores three objects, whose names are "OBJ1", "OBJ2" and "OBJ3" respectively, and two bundles. The bundle "celadon plates " holds two objects that were considered as a plate. The bundle "celadon cups" holds two objects that were considered as a cup.

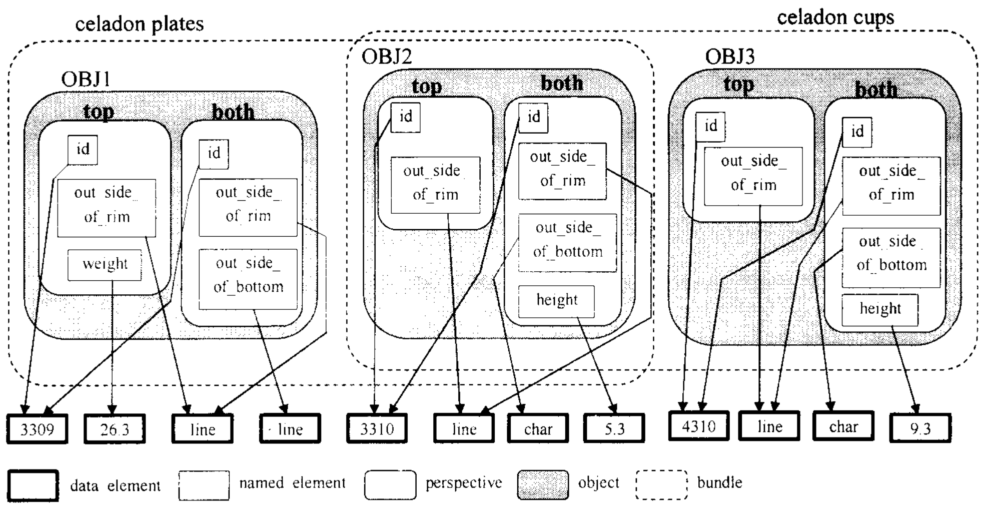


Figure 1: Example of a database

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. Furthermore, database elements can be changed easily by using the operations of DREAM model. Therefore, information about relics can be stored in a DREAM database even though they have different structures than each other. From the above discussions, it is obvious that we needn't define schema beforehand.

2.2 Shape entry and shape

Some information such as names of attributes, data types, etc. are indispensable for operations of databases. If we use conventional data model, we can find out this information from database schema. Since DREAM model does not have a database schema, we cannot do so. Therefore we have defined a concept shape [10,11,12]. However, shape is insufficient to describe database structures since it can't explain nested structures of database elements. To solve this problem, we are to refine shape by introducing a new data structure "shape-graph".

Shape is defined to describe structures of perspectives, objects and bundles. The fundamental unit of shape is a "shape entry" describing the structure of a named element.

Definition 2. A shape entry describes the structure of a named element and is represented by a triplet (*id, name, DT*), where *id* is an identifier, *name* is the name of the named element and *DT* is a set of data types of data elements related to the named element and/or shapes of objects that are included in the named element.

Example 2. The following are shape entries of the named elements of OBJ2 in Example 1.
(*se*1,"*id*", {*int*}),
(*se*2,"*out_side_of_rim*", {*string*}),
(*se*3,"*id*", {*int*}),
(*se*4,"*out_side_of_rim*", {*string*}),
(*se*5,"*out_side_of_bottom*", {*string*}),
(*se*6,"*height*", {*float*})

As Example 2 shows, different shape entries have same names. Here, "coalescent set" of shape entries is to be introduced as a preliminary to define shape. In a coalescent set of shape entries, each shape entry must have different name.

Suppose shape entries se_1 and se_2 have the same name and se_2 is included in a coalescent set, if se_1 is inserted into this coalescent set then a new shape entry must be created, which contain name of se_2 and the set of data types obtained through the union of sets of data types of se_1 and se_2.

Definition 3. A coalescent set of shape entries is defined as follows.

- An empty set is a coalescent set of shape entries.

- Let S be a coalescent set of shape entries and se_1 be a shape entry not included in S. If there is no shape entry with the same name as se_1, then union of S and $\{se_1\}$ is a coalescent set of shape entries.
- If there is a shape entry se_2 in S has the same name as se_1, then union of S and $\{se_1\}$ is not a coalescent set of shape entries. If combining se_1 to S to construct a new coalescent set of shape entries, then it is the union of the set $\{x \mid x \in S, x \neq se_2\}$ and the set that consists of only new shape entry that has the name of se_2 and a set of data types $\{dt \mid dt \in DT(se_1) \vee dt \in DT(se_2)\}$.

Definition 4. There are three kinds of shape: shape of an object, a perspective and a bundle. Each shape is a triplet (*id, name, S*), where *id* and *name* are an identifier and its name, and S is a coalescent set including shape entries with the named elements in the object, the perspective and the bundle.

Example 3 shows shapes of OBJ2 and perspectives in OBJ2. The shape entry *se*7 is a new shape entry obtained by combining *se*1 and *se*3 according to Definition 4. *se*8 is obtained similarly by combining *se*2 and *se*4.

Example 3.

Shapes of perspectives in OBJ2:
(*s*1, "*top*", {*se*1, *se*2}),
(*s*2, "*both*", {*se*3, *se*4, *se*5, *se*6})

Shapes of OBJ2 *(containing new two shape entries indicated with *1 and *2)*
(*se*7, "*id*", {*int*})) ... **1*
(*se*8, "*out_side_of_rim*", {*string*}) ... **2*
(*s*3, "*OBJ2*", {*se*7, *se*8, *se*5, *se*6})

2.3 Shape-graph

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. This information is very important in understanding construction of databases. For this, we propose shape-graph.

Definition 5. There are two kinds of shape-graphs. One is a shape-graph of an object and another is a shape-graph of a bundle.

The shape-graph of an object is a triplet (*id, s_obj,* {*s_per*}), where *id* is an identifier of the shape-graph, *s_obj* is a shape of the object and {*s_per*} is a shape set of perspectives included in the object.

The shape-graph of a bundle 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 included in the bundle.

Example 4. The shape-graph of OBJ2 is a triplet (*sg1, s3,* {*s1, s2*}). Figure 2 shows its structure. As shown in Fig.2, a shape-graph is a tree consisting of shapes that correspond to the object and the perspectives.

3 Design and Implement of DREAM DBMS

In this section, we discuss design and implementation of DREAM DBMS. We have developed a system on Compaq ProLiant ML350 server (CPU Pentium III 600MHz, 256MB Memory, Red Hat Linux release 6.1J). The DREAM DBMS has been implemented by UniSQL/X release 5.0, which is a commercial Object Relational DBMS [13].

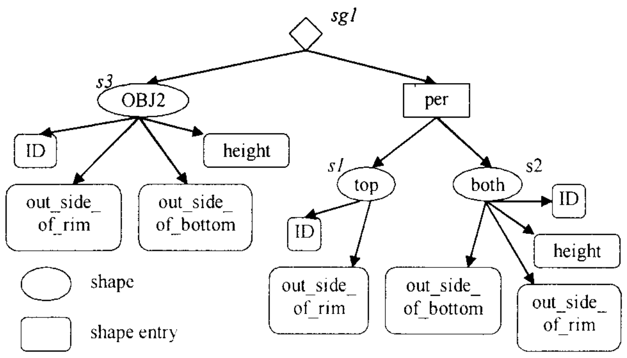


Figure 2: The shape-graph of OBJ2

3.1 Storing database elements and shapes

Each database element (such as a named element), shape, shape entry and shape-graph (we call shapes, shape entries and shape-graphs as SHAPE for short hereafter) are stored in a corresponding class (here class is represented by table in UniSQL/X database). Moreover, to realize a processing system of database operations, we represent the operations of DREAM model by the operations of UniSQL/X.

However, SHAPE and database elements, except data element, have similar structure, which is a triplet (*id*, *name*, *S*). Here *id* is an identifier, *name* is a name of the SHAPE or the database element and *S* is a set of database elements, data types or shape entries. Furthermore, database elements and SHAPE have similar operations. So it is inefficient to create classes individually for each database elements and SHAPE.

Therefore, we have introduced a kernel data model called *named set model* [11], in which so called named set is defined. It has a simple structure and is represented by a triplet (*id*, *name*, *S*), where *id* is an identifier, *name* is a name and *S* is a set of named sets and/or data. In addition, named set model has operations such as union, difference, which are similar to operations on set theory. Operations of DREAM model are based on these operations.

We have to define classes that correspond to database elements and SHAPE by UniSQL/X. Firstly we define a class of named set (*ns class*) and its operations. And then, we define classes of database elements and SHAPE with inheriting *ns class*. Data elements are stored into classes that are created corresponding to each data type. Note that these classes storing data elements have no connection with *ns class*. This method makes implementation of DREAM DBMS easy. Figure 3 shows relationships between DREAM model, named set model and UniSQL/X.

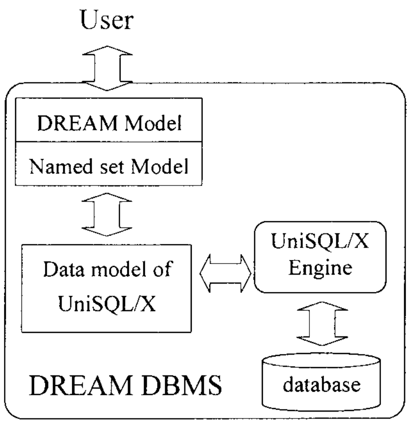


Figure 3: Relationships between DREAM model, named set model and UniSQL/X

Figure 4 shows classes of object, perspective, named element, shape graph, shape, and shape entry. Let us explain the *named element class* that store shape entries has attributes *id*, *name*, *S* and *ID*. The attribute *S* of the class is a set of identifiers of the data elements stored into the class of data elements. An attribute *ID* is a set of identifier of shape entries corresponding to the named element such as shown by the arrows in Fig.4. The attribute is to make processing time of database operations short. The attribute *ID* of other class is similar to the attribute of *named element class*. Other classes shown in Fig.4 are almost the same.

Furthermore, SHAPE 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 SHAPE, 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 SHAPE that should be reconstructed.

On the other hand, database operations in DREAM DBMS are provided by API (Application Program Interface) libraries of C language. We call these API libraries 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 by referring the database schema that consists of database name, table name, attribute name, data type and so on. But there isn't a database schema in DREAM model and further each object has different structure. So 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 (DREAM GUI) implemented by DREAM API, Java and JNI [14]. Figure 5 shows structure of DREAM DBMS & DREAM GUI. And Figure 6 shows two

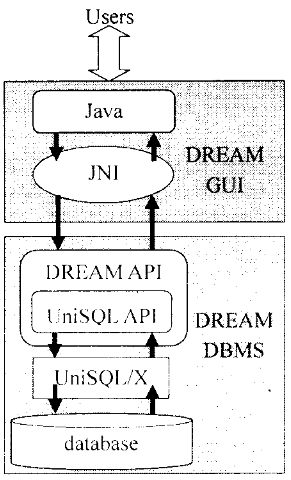


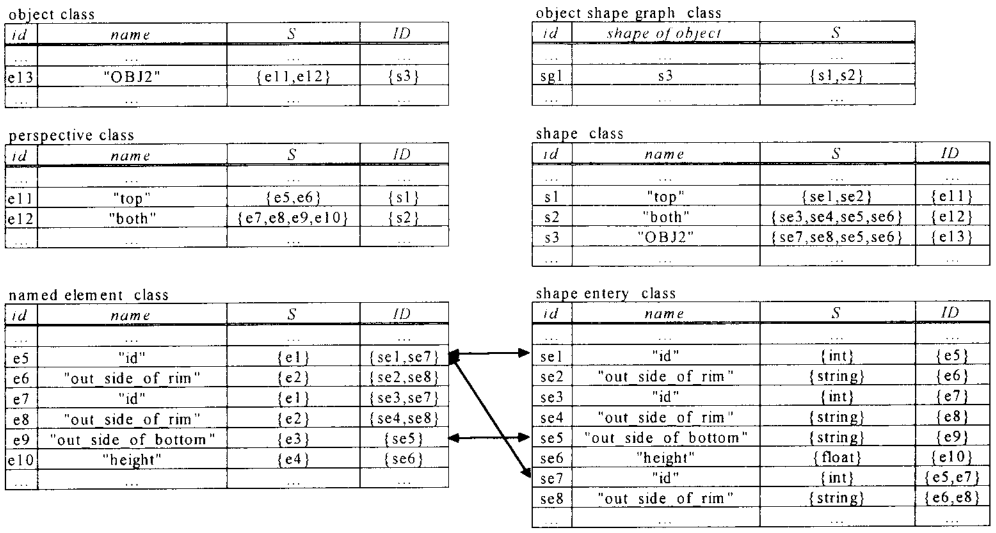
Figure 5: Structure of DREAM DBMS & DREAM GUI

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 have names "out_side_of_rim", "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 the result of the searching. As it shows, there is one object that satisfies the condition. The right window explains the detail of the object found by the searching.

4 Evaluation

We have implemented the DREAM DBMS and realized the function of constructing shapes and shape-graphs on it. To evaluate the 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 SHAPE for 200, 400, 600, 800, 1000 objects.



object class

id	*name*	*S*	*ID*
...	...	...	...
e13	"OBJ2"	{e11,e12}	{s3}
...	...	...	...

perspective class

id	*name*	*S*	*ID*
...	...	...	...
e11	"top"	{e5,e6}	{s1}
e12	"both"	{e7,e8,e9,e10}	{s2}
...	...	...	...

named element class

id	*name*	*S*	*ID*
...	...	...	...
e5	"id"	{e1}	{se1,se7}
e6	"out side of rim"	{e2}	{se2,se8}
e7	"id"	{e1}	{se3,se7}
e8	"out side of rim"	{e2}	{se4,se8}
e9	"out side of bottom"	{e3}	{se5}
e10	"height"	{e4}	{se6}
...	...	...	...

object shape graph class

id	*shape of object*	*S*
...	...	...
sg1	s3	{s1,s2}
...	...	...

shape class

id	*name*	*S*	*ID*
...	...	...	...
s1	"top"	{se1,se2}	{e11}
s2	"both"	{se3,se4,se5,se6}	{e12}
s3	"OBJ2"	{se7,se8,se5,se6}	{e13}
...	...	...	...

shape entery class

id	*name*	*S*	*ID*
...	...	...	...
se1	"id"	{int}	{e5}
se2	"out side of rim"	{string}	{e6}
se3	"id"	{int}	{e7}
se4	"out side of rim"	{string}	{e8}
se5	"out side of bottom"	{string}	{e9}
se6	"height"	{float}	{e10}
se7	"id"	{int}	{e5,e7}
se8	"out side of rim"	{string}	{e6,e8}
...	...	...	...

Figure 4: Classes of database elements

Condition 2 To reconstruct a part of SHAPE 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 7 shows the times to reconstruct SHAPE 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 these two classes to reconstruct SHAPE is an efficient method.

5. Concluding remarks

We have refined the previously proposed concept "shape" by introducing a new concept "shape-graph". Implementing DREAM DBMS as well as DREAM GUI, we have evaluated the performances of our system by measuring its runtime taken in constructing shape and shape-graph for new inserted data. As the results, (i) nested structures can be represented; (ii) by the GUI showing shape-graphs, users can operate databases even if they don't grasp the schema; and (iii)

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

the evaluation experimental results show that DREAM DBMS can construct dynamic schema efficiently, that is our system can be developed to a practical database system handling semi-structured data.

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 SHAPE; and (iii) provide the database query and manipulation language such as SQL.

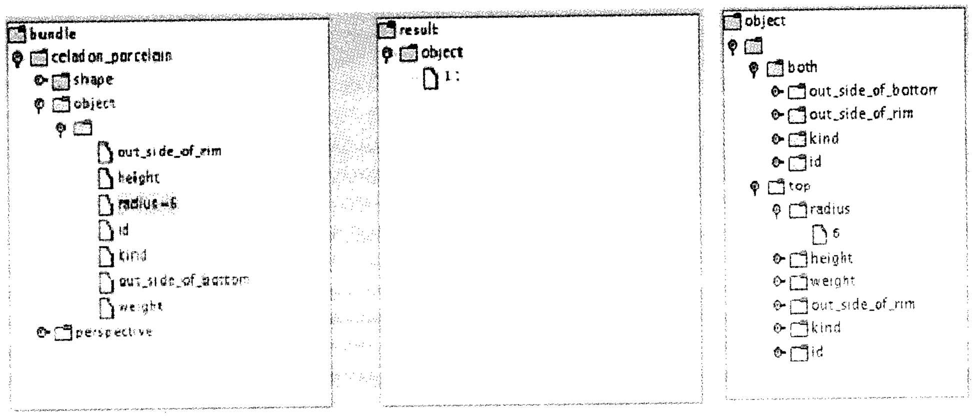


Figure 6: Windows of DREAM GUI

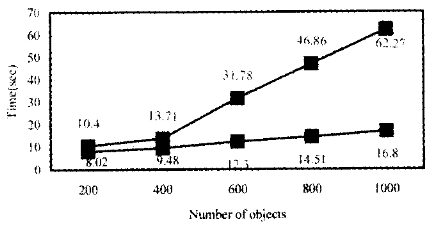


Figure 7: Times to construct SHAPE

References

[1] Zdonik, S. B., Incremental Database Systems: Database from the Ground Up, Proc. of ACM SIGMOD 1993, pp.408-412, 1993.

[2] 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.

[3] Abiteboul, S., *et al*, The Lorel query language for Semistructured Data, International Journal on Digital Libraries, Vol. 1, No. 1, pp.68-88 1997.

[4] Abiteboul, S., Querying Semi-Structured Data, Proc. of the 6th Int'l Conf. on Database Theory, pp.1-18, 1997.

[5] Buneman, P., *et al*, A Query Language and Optimization Techniques for Unstructured Data, Proc. of the 1996 ACM SIGMOD Int'l Conf. on Management of Data, pp.505-516, 1996.

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

[7] Shoens, K. *et al*, The Rufus System: Information Organization for Semi-Structured Data, Proc. of the 19th VLDB Conf., pp.97-107, 1993.

[8] Pfaltz, J. L. and French, J. C., Scientific Database Management with ADAMS, Bulletin of the Technical Committee on Data Engineering, Vol. 16, No. 1, pp.14-18, 1993.

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

[10]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, 1999.

[11]Hochin, T., Nakata, M. and Tsuji, T., A Flexible Kernel Data Model for Bottom-Up Databases and Management of Relationships, Proc. of Int'l Database Engineering & Applications Symposium 98, pp.170-177, 1998.

[12]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.

[13]Stonebraker, M., OBJECT- RELATIONAL DBMSs, Morgan Kaufmann Publishers, Inc., 1996.

[14]Rob Gordon, Essential JNI: Java Native Interface (Essential Java), Prentice Hall, 1998.