データベース・システム 57－9
（１９８７．１．１９）

SODB: An Object Oriented Database in Smalltalk-80

Mojtaba Mozaffari
Yuzuru Tanaka

Department of Electrical Engineering
Hokkaido University
Sapporo 060, Japan

Abstract: 'Object Oriented Data Model' (**ODM**) is a new data model based on object oriented programming concepts. This model extends the data structures and operations of Relational Data Model (RDM). SODB that is described here is an implementation of ODM in Smalltalk-80 environment.

In SODB, a database is defined as a set whose elements are named uniform sets. The uniform set concept is an extension of relation of RDM. All of the elements of a uniform set are instances of the same class. However, in contrast to the relations in RDM, the elements of a uniform set are not necessarily tuples. We have also extended operations such as 'join' and 'project' of RDM.

SODB:Smalltalk-80に基づいたオブジェクト指向データベースシステム

モジィタバ・モザファリ　田中譲
北海道大学工学部

「オブジェクト指向データモデル」(ODM)はオブジェクト指向プログラミングの概念に基づいた新しいデータモデルである。このモデルは関係データモデル(RDM)のデータ構造と演算を拡張している。本論文で示すSODBはODMをSmalltalk-80によって実現したものである。

SODBは、データベースは定型集合と名付けられたオブジェクトを要素として持つ集合と定義される。定型集合の概念はRDMにおける関係の拡張である。定型集合のすべての要素は同じクラスのインスタンスである。しかし、RDMにおける関係とは異なり、定型集合の要素はタプルでなくてもよい。RDMにおける「ジョイン」や「プロジェクト」のような演算の拡張についても示す。

1. Introduction

Relational Data Model (RDM) was proposed by E. F. Codd [CODD70] in 1970 based on sound ideas and provides the basis for many Database Management Systems (DBMS). However these Relational Database Management Systems (RDBMS) and RDM itself have various limitations. We propose **SODB** (Object oriented Database in Smalltalk-80) as a prototype system that improves these limitations.

During 1980s the abundance of RDBMS that operate efficiently in data processing environments has suggested their use for other applications such as engineering design. This has enlightened some limitations of these RDBMS and also provided improvements that make a particular RDBMS suitable for a specific application area [GUTT82, KLAU85, LORI83, STON83]. It has already been pointed out that the requirements of different applications could be different [CHOL83, FOIS82, HASK82]. Clearly creation of a new DBMS for each application area is not a good strategy. On the other hand such improvements to tune up a RDBMS for a specific application area have been ad hoc; they improve functionality for some environments while they increase complexity and can be useless or even harmful in other environments. Therefore *we need DBMS that are general purpose and more flexible than current RDBMS.*

The above limitations are not confined to the implementations of RDM; they also exist in the data model itself. For example the tuples are not suitable for representation of inhomogeneous data [KENT79]. Moreover the operations of RDM such as 'join' are inherently defined in terms of tuples; while tuples are not suitable to represent all data that we may need. Thus the operations may be useless. For example what does the 'project' operation mean in a database whose data is mainly pictures?

In RDM a domain is defined to be a set of atomic values. In practice the data types supported by the Data Definition Language (DDL) restricts the domains that we can define for the attributes of a relation. While the Data Manipulation Language (DML) defines the operations. The data structures and operations defined in these languages are not sufficient and the user appeals to a programming language such as PL/1 to provide the required functionality. There are also database programming languages [JARK82, ROWE79, SCHM77, SHOP79, WASS79] that include the data structure and operations of RDM. These database programming languages somewhat improve the situation [REIN81]. For example optimization of queries can be left to these languages which simplifies the DBMS. However these improvements are not sufficient; more flexible data type facility is needed to cope with various applications. For example PLAIN [WASS81] provides useful constructors to define new data types in terms of existing data types. These constructors enhance the data type facilities of PLAIN but they are high level and more flexible low level facilities are also needed. In other words PLAIN allows us to define new molecular types from existing types but does not allow us to define new atomic types. The problem is that in contrast to atoms in chemistry the required atomic types in computer depend on the application and are not known a priori. We conclude that *a DBMS that supports different applications should provide facilities to define new atomic and molecular types when they are needed.*

We propose SODB as a prototype DBMS that has more powerful data structures and operations than current RDBMS. It is an attempt to provide the flexibility needed for various applications in one system. SODB is based on a new data model called ODM (Object oriented Data Model). This data model is based on the ideas of Object oriented Programming and RDM. This model is also useful in

environments with object oriented features but some modifications may be needed. Some features of SODB are inherited from Smalltalk-80 whence they are not necessarily a characteristic of a database system based on ODM in other environments. Our main purpose in developing SODB was to provide a concrete example based on ODM to experiment and examine the advantages of the model itself.

2. Smalltalk-80 and RDM

Our goal in SODB is to bring together the advantages of Smalltalk-80 and RDM in a uniform system. We shall continue this section with a brief description of pertinent Smalltalk-80 features. Then we highlight some extensions to Smalltalk-80 that we need. Our description of Smalltalk-80 is not complete and the reader may consult the references [BYTE81, GOLD83, GOLD84] for further information.

2.1 Smalltalk-80 features

To begin with Smalltalk-80 is not only a programming language but a system that can work independently without any requests to operating system. It provides a friendly user-interface and allows graphics and animation. All these come in a uniform environment based on the concept of object. Since Smalltalk-80 is a system we do not need separate DDL, DML and host languages. In SODB everything is done in one environment namely Smalltalk-80 with addition (or modification) of some classes and messages. This uniform environment is an advantages of SODB to current DBMS.

Smalltalk-80 has its origin in the simulation language Simula-67 and artificial intelligence. It is suitable for modelling complex systems. The objects represent real world entities with instance variables that represent their state. The objects communicate by sending and receiving messages. This is a powerful paradigm and makes it easy to design an interface between Smalltalk-80 and other systems such as a banking system.

In Smalltalk-80 each object is an instance of a class. The only way to modify an object is by sending a message to it. While the response to a message is defined via the class of the receiver. Thus modularity is preserved. The classes form a hierarchical structure which greatly increases functionality and simplifies design of complex systems. For example if we want a message to be understood by all objects in the system we just need to define that message in 'Object' that is the root of this hierarchy. The class of an object corresponds to its data type in conventional languages. This hierarchical type structure provides reusability of code. It is a powerful tool since it allows us to do a lot of work with minor changes to existing code.

The standard Smalltalk-80 system supports many classes and the user can easily define a new class dynamically or modify an existing class to match an application. When a class is no longer needed the user can remove it from the system. The data structure of an object and the operations performed on it are defined via the class of that object. Therefore the separation of data structure and operations that is a common drawback of some systems disappears. The modification of a class can modify both the data structure and operations done on the instances. These modifications are noticed by the instances and subclasses of the class automatically. This powerful and flexible type structure of Smalltalk-80 is inherited to SODB.

There are also limitations in Smalltalk-80. One drawback is that the information that can be specified in a class is rather limited. As an example a class definition provides no mechanism (such as type declaration of conventional programming languages) to define the class of the instance variables. The class of an object is defined when it is created but we also need some type mechanism for the

instance variable. Therefore in SODB we defined conversion messages to support data types for the instance variables of an object. Generally speaking, conversion of an object x to a class C will result in an instance of C that is considered to be equal to x. However we still feel that Smalltalk-80 classes need more expressive power.

2.2 Comparison of RDM and Smalltalk-80

In this section we compare RDM with Smalltalk-80 to see what extensions to the latter are needed.

In RDM a database is a set of relations where each relation is a set of tuples. Moreover each tuple is a set of pairs (ai, vi) where ai is an attribute name and vi is the corresponding attribute value. We see that in RDM the database, relation and tuple are in fact 'set of similar objects'. Therefore 'set of similar objects' is a fundamental concept in RDM. In contrast the elements of a set in Smalltalk-80 need not be similar. In SODB we introduce the concept of **'uniform set' (u-set)** for a 'set of similar objects'. A tuple in RDM is a 'set of similar objects' but it has a more specific structure. In particular for each attribute name a domain is specified and the corresponding attribute value should belong to that domain. Smalltalk-80 does not support **tuples** or **attributes** hence we introduce them as new concepts in SODB.

We clarify the term 'similar objects' to mean objects that are instances of the same class. Then a u-set is a set all of whose elements are instances of a fixed class namely the 'element class' of the u-set. We show a u-set and its elements in Fig. 2.1. In this paper arrows with dashed lines connect instances to their classes. In SODB tuple and attribute are defined as extensions of similar terms in RDM. Domain concept of RDM corresponds to class concept in SODB.

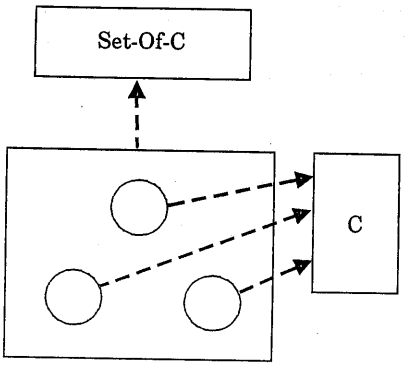

Fig. 2.1. A u-set whose element class is C.

Now we consider the operations. RDM defines two kinds of operations on relations. These are the usual set operations such as 'union' and relational operations such as 'join'. The usual set operations are supported by all u-sets but the relational operations are not. The reason is that the relational operations are not always necessary as explained in section 1. However the concept of **'relational u-set'** has been introduced to support relational operations.

We conclude that we need the following new concepts for SODB.

1) attribute
2) tuple
3) u-set
4) relational u-set
5) database

We shall continue with an overview of SODB in section 3 while section 4 provides further details of the above concepts.

3. Overview of SODB

In this section we provide an overview of SODB. We shall define the database and its elements but leave further details to section 4. We have to omit many details to stress the main points.

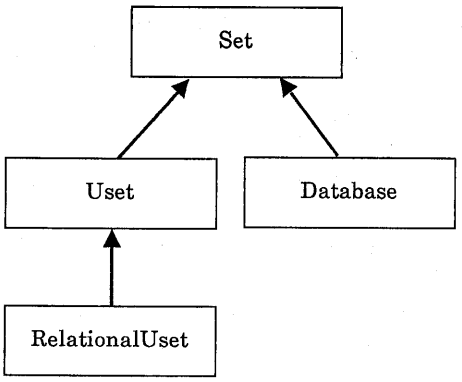

Fig. 3.1 An overview of the structure of SODB.

A sketch of SODB is given in Fig. 3.1. In this paper arrows with solid lines connect classes to their superclass. Here 'Set' is the Smalltalk-80 class named Set with some modifications. While Uset, RelationalUset and Database are new classes defined for SODB. A u-set is an instance of a subclass of 'Uset'. Similarly a relational u-set is an instance of a subclass of 'RelationalUset'. A database is an instance of 'Database'. The user can define subclasses for any of these classes to increase functionality.

More specifically sets in SODB provide for operations such as union and intersection. Each element of a u-set is an instance of the element class of the u-set. The relational u-sets are u-sets that support the relational operations of SODB.

An example database named Music is shown in Fig. 3.2. In SODB a database is a set which has a unique name that globally identifies it. Moreover each element of a database is a pair (s, u) where u is a u-set and s is the unique name of u within the database.

4. Extensions to Smalltalk-80

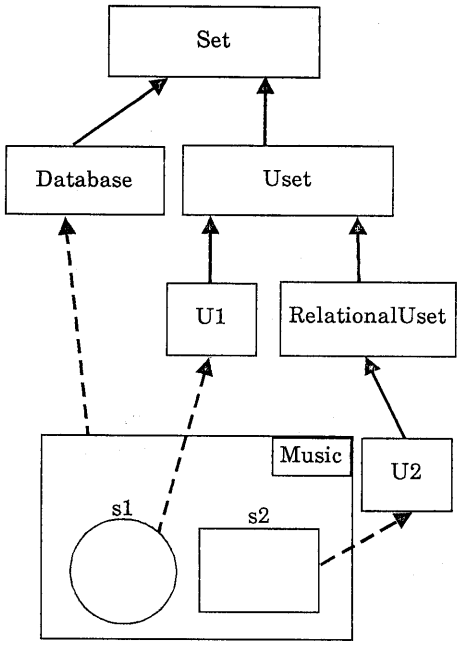

Fig. 3.2 A database whose name is Music and some of its elements.

In this section we explain the extensions of Smalltalk-80 that were mentioned in section 2.2.

4.1 Attributes

We define an attribute of an object, say x as a triple (n, v, V) where n is the attribute name, v is the attribute value and V is the attribute class. See Fig. 4.1 where an attribute of an instance of class X is shown.
We assume that:

1) An attribute name of x is a 'unary selector' (as explained in Smalltalk-80) for x and its class X. Briefly a 'unary selector' is a message selector without parameters.

2) The attribute value is the object that is returned when the corresponding attribute name is sent as a message to x.

3) The attribute class is the object returned when the attribute name is sent as a message to X.

4) The attribute value v is an instance of the corresponding attribute class V.

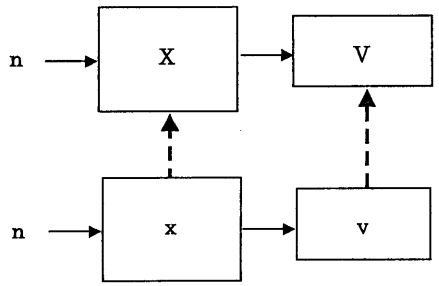

Fig. 4.1 The triple (n, v, V) is an attribute of x where x is an instance of X.

In SODB attributes are defined for all objects so that any object can have zero or more attributes. Moreover an attribute value can be any object.

4.2 Tuples

A tuple is an object that is determined by its attributes. More precisely a tuple is an instance of a tuple class; a tuple class is a subclass of a system defined class named Tuple. Moreover a tuple class is created by a message such as:

typed: pairSet named: aSymbol

sent to Tuple. Here pairSet is a set of pairs (ni, Vi) where ni is an attribute name and Vi is the corresponding attribute class. Moreover a Symbol will be the name of the newly created tuple class. In general the attribute names of a tuple are also its instance variable names; these instance variables can be assigned a value by a message such as:

ni: value

This message will convert value into Vi and then assigns the result to the instance variable ni.

Example 1

We define a tuple class named 'E' with 'namePart', 'salary' and 'manager' as the attribute names that refer to the name, salary and manager of the instances. Then we create some instances of 'E' and finally make a query.

Tuple typed: #((manager E) (namePart Symbol) (salary Money)) named: #E.

e1 ← (E new) namePart: #Smith; salary: 1100.

e2 ← (E new) namePart: #Brown; salary: 1000; manager: e1.

Here 'Money' is a user defined subclass of integer to improve semantics. Now the message:

e2 manager salary

returns 1100 that is the salary of Brown's manager.

Now we explain a message that defines a tuple from the attributes of an object. This is called the 'extract' message and is defined in 'Object'. The message is written as:

extract: nameSet

Here nameSet is a set whose elements are some of the attribute names of the receiver. The message returns a tuple whose instance variable names are all the elements of nameSet. The value (or class) of each of the instance variables will be the corresponding attribute value (or class) of the receiver of the 'extract' message. The required tuple class will be defined by the system automatically.

4.3 Uniform sets (U-sets)

A u-set is an instance of a 'u-set class' while a u-set class is a subclass of Uset. Moreover a u-set class is created by a message such as:

of: aClass named: aSymbol

that is sent to Uset. Here aClass is the element class and aSymbol is the name of the newly created class.

Uset redefines some of the messages of Set and also defines new messages. In particulr the message:

basedOn

is defined for all u-sets and returns the element class of the receiver.

4.4 Relational u-sets

A relational u-set is defined similar to a u-set. It is an instance of a 'relational u-set class' and the latter is a subclass of RelationalUset created by a message such as:

of: aClass named: aSymbol

similar to that explained above.

RelationalUset is a subclass of Uset therefore the relational u-sets support the messages for u-sets. In addition RelationalUset defines messages that correspond to the relational operations of RDM. Among these messages we briefly explain the 'join' and 'project' messages and note that 'select' message is inherited from Smalltalk-80.

The 'join' message is written as:

join: aUset by: aBlock2

Here aUset is a uniform set and aBlock2 is a block with two arguments. The message returns a relational u-set whose elements are all possible (x, y) pairs where x is an element of the receiver and y is an element of aUset and aBlock2 returns 'true' when evaluated with x, y as its arguments respectively. The required classes will be defined by the system automatically. The above pairs support the messages:

first

This message will return the x component of the pair.

second

This message will return the y component of the pair.

Now we consider the project message that is written as:

project: nameSet

Here nameSet is a set whose elements are attribute names of the elements of the receiver. The message will return a relational u-set whose elements are all the objects created by sending the message:

extract: nameSet

to each element of the receiver of the project message. The required classes will be defined by the system automatically.

4.5 Databases

A database is an instance of Database; while Database is a subclass of Set. A database is created by a message such as:

new: anInteger named: aSymbol

sent to Database. Here a Symbol is the name of the newly created class and anInteger is the initial number of elements allowed in the database. A database will provide space for more elements automatically.

Example 2

We define a new database and a relational u-set class for a set of rectangles. Then we create a set of rectangles and investigate which rectangles intersect. Finally we put the pairs of intersecting rectangles into the database.

```
Database new: 5 named: #Graph.
RelationalUset of: Rectangle named: #Rect.
s1 ← Rect new: 20.
s1 add: (Rectangle origin: 100@100 corner: 200@200).
s1 add: (Rectangle origin: 150@100 corner: 200@200).
```

other rectangles can be added to s1 similarly.

```
s2 ← s1 join: s1 by: [ :x :y | x ~~ y & (x intersects: y)].
Graph at: #Intersecting put: s2.
```

Note that even if Smalltalk-80 had not implemented the 'intersects:' message the user

could define it separate from the above computation. Therefore SODB supports high level operations.

5. Comparison of RDM and SODB

Now that we have explained the basic features of SODB we can compare it with RDM to highlight the differences. RDM defines specific data structures and operations and enforces the integrity and referential rules [CODD79]. We shall compare each of these aspects with SODB below.

The user can define subclasses of Database, Uset, RelationalUset and tuple classes to further tune up the system for a specific application area. Moreover SODB concepts of attribute, tuple and u-set are more general than attribute, tuple and relation in RDM as explained below.

1) In contrast to RDM the definition of attributes does not rely on tuples. Attributes are defined for all objects while in RDM only tuples can have attributes.

2) In contrast to RDM an attribute value need not be atomic. It can be any object including sets or u-set. This provides for repeating fields.

3) In contrast to RDM an attribute value need not be some part of another object. The attribute value can be somehow computed by the object whose attribute is sought. This provides for derived attributes.

4) The generalization of attribute automatically generalizes tuples in SODB. Similarly a result of the generalization of tuples is that u-sets or relational u-sets (which may be set of tuples) are more general than relations of RDM. Moreover the elements of a u-set need not be tuples whence the u-sets are further enhanced.

The improvement of the data structure in SODB results in an improvement of the operations. The definition of these operations in SODB provides further improvements. For example the blocks used in 'join' and 'select' messages can evaluate complex expressions to decide inclusion of an element in the result.

The integrity and referential rules of RDM are not directly supported by SODB. These rules are based on the assumption that each relation is a set of tuples and a fixed attribute of a tuple is its primary key. These assumptions do not hold in SODB whence we do not support the rules. The primary keys provide a mechanism to access a tuple within a relation. However such a mechanism already exists in Smalltalk-80; since each object has an object pointer that uniquely identifies it in the system. Therefore in SODB primary keys are not so essential as in RDM. Of course the user is free to enforce these rules for some applications and SODB does not prevent it.

6. Conclusion

SODB is proposed as a prototype to open a new direction for database systems. It runs in a uniform environment and provides the flexibility of Smalltalk-80. It encapsulates data and operation and improves semantics. It allows modularity and high level operations. In a sense it is simpler than RDM since the user is not obliged to use primary keys or tuples. Moreover the user does not need to learn both query and host languages. SODB is a single user system but we think that its concepts provides a basis for an object oriented system with database facilities.

References

[BYTE81] Byte Vol. 6., No. 8, (Aug. 1981).

[CHOL83] Cholvy, L. and Foisseau, J.: 'ROSALIE: A C.A.D. Object oriented and Rule-based System', Information Processing 83, (Elsevier Science Publishers) pp. 501-505.

[CODD70] Codd, E.F.: 'A Relational Model of Data for Large Shared Data Banks', Communications of the ACM Vol. 13, No. 6, (June 1970), pp. 377-387.

[CODD79] Codd, E.F.: 'Extending the Database Relational Model to Capture More Meaning', ACM Transactions on Database Systems Vol. 4, No. 4, (Dec. 1979), pp. 397-434.

[FOIS82] Foisseau, J. and Valette, F.R.: 'A Computer Aided Design Data Model: FLOREAL', File Structures and Data Bases for CAD, (North-Holand Publishing Co. 1982), pp. 315-334.

[GOLD83] Goldberg, A. and Robson, D.: 'Smalltalk-80: The language and its Implementation', (Addison Wesley Publishing Co. 1983).

[GOLD84] Goldberg, A.: 'Smalltalk-80: The Interactive Programming Environment', (Addison Wesley Publishing Co. 1984).

[GUTT82] Guttman, A. and Stonebraker, M.: 'Using a Relational Database Management System for Computer Aided Design Data', Database Engineering June '82, pp. 155-162.

[HASK82] Haskin, R. and Lorie, R.: 'On Extending the Functions of a Relational Database System', Proc. 1982 ACM SIGMOD Int. Conf. on Management of Data, (Orlando, FL, June 82), pp. 207-212.

[JARK82] Jarke, M. and Schmidt J.W.: 'Query Processing Strategies in the Pascal/R Relational Database Management System', Proc. 1982 ACM SIGMOD Int. Conf. on Management of Data, (Orlando, FL, June '82), pp. 256-264.

[KENT79] Kent, W.: 'Limitations of Record-Based Information Models', ACM Transactions on Database Systems Vol. 4, No. 1, (March 1979), pp. 107-131.

[KLAU85] Klaus, R.D. and Lorie, A.R.: 'Vision Support for Engineering Database Systems', IBM Research Laboratory, San Jose, CA 95193.

[LORIE83] Lorie, R. and Plouffe, W.: 'Complex Objects and Their Use in Design Transactions', Proc. ACM SIGMOD 1983 Engineering Design Applications, pp. 115-121.

[REIN81] Reind P., Wasserman A.I.: ' High Level Programming Features for Improving the Efficiency of a Relational Database System' ACM Transactions on Database Systems Vol. 6, No. 3, (Sept. 1981), pp. 464-485.

[ROWE79] Rowe, L.A. and Shoens K.A.: 'Data Abstraction, Views and Ups in RIGEL', Proc. 1979 ACM SIGMOD Int. Conf. on Management of Data, (Boston, Mass. May '79), pp. 71-81.

[SCHM77] Schmidt, J.W.: ' Some High Level Constructs for Data of Type Relation', ACM Transactions on Database Systems Vol. 2, No. 3, (Sept. 1977), pp. 247-261.

[SHOP79] Shopiro, J.E: 'Theseus __ A Programming Language for Relational Databases, ACM Transactions on Database Systems Vol. 4, No. 4, (Dec. 1979), pp. 493-517.

[STON83] Stonebraker, M,, Rubenstein, B. and Guttman, A.: 'Application of Abstract Data Types and Abstract Indices to CAD Data Bases', Proc. Engineering Design Applications of ACM-IEEE Database Week, (San Jose, CA. May '83), pp. 107-113.

[WASS79] Wasserman, A.I.: 'The Data Management Facilities of PLAIN', Proc. 1979 ACM SIGMOD Int. Conf. on Management of Data, (Boston, Mass. May '79), pp. 60-70.

[WASS81] Wasserman, A.I., et al.: 'Revised Report On Programming Language PLAIN', ACM SIGPLAN Notices Vol. 16, No. 5, (May 1981), pp. 59-80.