In Proceedings of the International Conference on Image Science, Systems, and Technology (CISST '97), Las Vegas, USA, June 30 - July 3, 1997. CSREA Press

Storing Graphic Data in Databases

Mengchi Liu
Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada S4S 0A2

Abstract *This paper investigates how new database technologies, in particular, complex object databases, object-oriented databases, and deductive databases, can be used to store object-oriented graphic data and discusses the related problems.*

Keywords: graphic data structures and databases, scene and object modeling

1 Introduction

Graphic data is generally divided into two kinds: bit-mapped graphics and object-oriented graphics [18]. Bit-mapped graphics are usually generated with scanning or painting software, while object-oriented graphics are usually created with drawing software using primitive objects such as points, lines, shapes, texts etc. How to effectively store graphic data in a database system presents a challenge to the traditional relational database technology. Graphic data is usually handled by various file systems, rather than database systems. As a result, the graphics can only be displayed or printed rather than be queried. For example, we cannot ask where a specific object is located, which object is at a certain location, how many same kind objects there are in the scene, etc. Besides, we cannot update objects systematically. For example, change the shape of all identical objects in the scene.

As the relational database technology is inadequate for many advanced applications, it has been extended significantly during the past decade. These extensions can be classified into three kinds: complex object databases, object-oriented databases and deductive databases. Complex object databases [2, 3, 6, 15, 20, 32, 27, 35, 38, 39] relax the first normal form constraint imposed on the relational databases by allowing complex objects to be formed using the tuple and set constructors. Object-oriented databases [7, 11, 12, 16, 17, 21, 24, 26] extends data modeling power of the relational databases by means of object identity, complex objects, classes, class hierarchies, and inheritance. Deductive databases [1, 5, 9, 10, 13, 19, 23, 28, 31, 33, 34, 37, 36, 40, 41, 42] extend the expressive power of the relational databases by means of rules for deduction, recursion and declarative querying.

In this paper, we examine how these new database technologies can be used to store object-oriented graphic data and discuss the related problems. We use the following scene to illustrate.

In the scene, there are two kinds of trees: pine and maple, two kinds of people: standing and walking, two kinds of houses: bungalow and two storey, some roads, a garden of maple trees, a mountain and the Sun. The trees, people, and houses appear in the scene at fixed intervals. These objects are formed with circles and/or lines.

As a database usually consists of two parts: the schema and the instance, we will discuss how to define the schema for this particular graphic database and how to represent the instance using different kinds of databases.

This paper is organized as follows: Section 2 investigates complex object databases. Section 3 examines object-oriented databases. Section 4 looks into deductive databases. For simplicity, we shall use a virtual language for each kind of database to illustrate the ideas rather than real languages supported by various database management systems.

2 Complex Object Databases

Complex object databases relax the first normal form constraint imposed on traditional relational databases. The tuple components of a relation can be complex objects, such as tuples, sets or even relations themselves. Complex objects are formed from atomic values such as integers, reals, strings, characters, etc. using the tuple and set constructors.

A type is associated with each value in the obvious way and each value is an instance of the type. The types for atomic values (atomic types) such as Int, Real, String, Char, etc. are usually built-in. The types for complex objects (complex types) have to be explicitly defined by the user, using the tuple and set constructors, as follows:

1. If $\mathsf{T}_1, ..., \mathsf{T}_n$ are types, and $\mathsf{A}_1, ..., \mathsf{A}_n$ are distinct attributes, then $[\mathsf{A}_1{:}\mathsf{T}_1, ..., \mathsf{A}_n{:}\mathsf{T}_n]$ is a tuple type.

2. If T is a type, then {T} is a set type.

Complex types and values can be viewed as trees. In a typed tree, leaves are atomic types, and in a value tree, leaves are atomic values. Internal nodes are labeled in both cases by constructors. A type is tuple or set, respectively, according to the constructor at the root of its tree.

Complex types can be named by using type equations of the following form:

T = T'

where T is a type name and T' is a type.

The schema of a complex object database is a collection of type equations such that there is no cyclicity in the type equations.

Consider the scene in Section 1. If we ignore the objects formed with circles such as people and the Sun, we can define the schema with the following type equations:

```
InScene= {[Object: {Line}, Loc: Point]}
Line   =  [From: Point, To: Point]
Point  =  [X: Int, Y: Int]
```

where Line and Point are named complex types formed with the tuple constructor, {Lines} is an unnamed complex type formed with the set constructor, and InScene is a named complex type formed with both the tuple and set constructors. Indeed, InScene is a set (nested relation) of tuples.

The database instance, without people and the Sun, for the scene in Section 1, is a set of tuples in the relation InScene as follows:

```
InScene:
  {[Object: {[From: [X:18,Y: 0], TO: [X:18,Y:20]],
             [From: [X:36,Y:20],TO: [X: 0,Y:20]],
             [From: [X: 0,Y:20],TO: [X:18,Y:55]],
             [From: [X:18,Y:55],TO: [X:36,Y:20]]},
    Loc: [X: 10,Y:235]],
   [Object: {[From: [X:18,Y: 0], TO: [X:18,Y:20]],
             [From: [X:36,Y:20],TO: [X: 0,Y:20]],
             [From: [X: 0,Y:20],TO: [X:18,Y:55]],
             [From: [X:18,Y:55],TO: [X:36,Y:20]]},
    Loc: [X: 130,Y:235]],
   [Object: {[From: [X:18,Y: 0], TO: [X:18,Y:20]],
             [From: [X:36,Y:20],TO: [X: 0,Y:20]],
             [From: [X: 0,Y:20],TO: [X:18,Y:55]],
             [From: [X:18,Y:55],TO: [X:36,Y:20]]},
    Loc: [X: 250,Y:235]],
   [Object: {[From: [X:18,Y: 0], TO: [X:18,Y:20]],
             [From: [X:36,Y:20],TO: [X: 0,Y:20]],
             [From: [X: 0,Y:20],TO: [X:18,Y:55]],
             [From: [X:18,Y:55],TO: [X:36,Y:20]]},
    Loc: [X: 370,Y:235]],
    ...
  }
```

Note that with this approach we can only store primitive objects such as points and lines, rather than abstract objects such as trees, houses, people, etc. and we have to explicitly duplicate the structure of the same object for different occurrences in the scene. As a result, if we want to change the structure of a particular kind of object such as the pine tree systematically in the scene, we have to modify all of its occurrences in the database. Besides, the query asking for which object is at a certain location, such as Loc: [X:10,Y:235], won't have meaningful result in such a database.

A solution to this problem is to decompose the InScene relation into two relations: InScene and LineObjects with the schema and instance as follows.

```
Schema:
    InScene     = {[Object: String, Loc: Point]}
    LineObjects= {[Name: String, Lines: {Line}]
    Line        =  [From: Point, To: Point]
    Point       =  [X: Int, Y: Int]
InScene:
    {[Object: Pine,   Loc: [X: 10,  Y:235]],
     [Object: Pine,   Loc: [X: 130,Y:235]],
     [Object: Pine,   Loc: [X: 250,Y:235]],
     [Object: Pine,   Loc: [X: 370,Y:235]],
     [Object: Pine,   Loc: [X: 490,Y:235]],
     [Object: Pine,   Loc: [X: 610,Y:235]],
     [Object: Maple, Loc: [X: 90,  Y:110]],
     [Object: Maple, Loc: [X: 140,Y:110]],
     [Object: Maple, Loc: [X: 190,Y:110]],
     [Object: Maple, Loc: [X: 240,Y:110]],
      ...
    }
LineObjects:
   {[Name: Pine,
     Lines: {[From: [X:18,Y: 0], TO: [X:18,Y:20]],
             [From: [X:36,Y:20],TO: [X: 0, Y:20]],
             [From: [X: 0, Y:20],TO: [X:18,Y:55]],
             [From: [X:18,Y:55],TO: [X:36,Y:20]]}],
    [Name: Maple,
     Lines: {[From: [X:10,Y: 0], TO: [X:10,Y:30]],
             [From: [X: 0, Y:25],TO: [X:10,Y:15]],
             [From: [X:10,Y:15],TO: [X:20,Y:25]]}],
     ...
   }
```

In this database, the objects in the relation LineObjects can only be formed with lines, rather than circles and lines. A natural question is how to add people and the Sun in the scene which are formed with circles and/or lines into the database. One solution is to modify both the schema and instance of the relation LineObjects so that objects can be formed with circles and lines. However, such an update in a database system is quite expensive and should be avoided. Another solution is to add a new relation CLineObjects with the schema and an additional instance for the relation CLineObjects as follows.

```
Schema:
    InScene      = {[Object: String, Loc: Point]}
    LineObjects  = {[Name: String, Lines: {Line}]}
    CLineObjects= {[Name: String, Lines: {Line}
                     Circle:[Center:Point, Radius:Int]}
    Line         =  [From: Point, To: Point]
    Point        =  [X: Int, Y: Int]

CLineObjects:
    {[Name: Sun,
      Lines: {},
      Circle: [Center: [X:10,Y:20], Radius: 20]],
     [Name: Standman,
      Lines: {[From: [X:10,Y:20],TO: [X:10,Y:10]],
              [From: [X:10,Y:15],TO: [X: 0, Y:10]],
              [From: [X:10,Y:15],TO: [X:20,Y:10]],
              [From: [X: 5, Y: 0], TO: [X:10,Y:10]],
              [From: [X:10,Y:10],TO: [X:15,Y: 0]]},
      Circle: [Center: [X:10,Y:25], Radius: 5]],
      ...
    }
```

However, this solution still suffers from several problems. Firstly, the object names such as Pine, Maple, Sun, Standman, etc. are used to connect the relation InScene to the relations LineObject and CLineObject, but the schema provides no accounts for such connections. We cannot say that the object names used in the relation InScene must appear in one of the two relations: LineObjects and CLineObjects. As a result, inconsistency may easily occur. Secondly, these object names are used to identify the corresponding graphic objects in the database but subject to updates which may cause inconsistency as well. Finally, all the information about objects in the scene is expressed explicitly even though we know that some of them appear at fixed intervals. As a result, if we want to increase or decrease the distance between these objects, we have to modify all of them.

3 Object-Oriented Databases

In an object-oriented database, each real world object is directly modeled as an object. Each object has an identifier and is associated with a complex object as its value or state. An object identifier uniquely identifies the object in the database and is not subject to changes while the value associated with the object is subject to changes. An object in an object-oriented database is referenced through its identifier rather than the complex object associated with it. Object identity provides better support for object sharing and simplifies update management.

Objects often share common properties and are grouped together into classes. Each class has a name and is associated with a complex type which specifies common properties of all objects in the class. Each object in the database belongs to (is an instance of) some class.

Classes are organized into class hierarchies by defining subclass (isa) relationships between classes. Instances of a subclass are also instances of its superclasses. Thus an object can be an instance of several classes. However, every object should have a primary class (the unique, lowest class in the class hierarchy where the object is an instance). Properties defined on the superclasses are inherited by their subclasses.

Inheritance supported by object-oriented paradigm provides a powerful mechanism for organizing data as it allows the user to define classes in an incremental way by refining already existing ones.

Each class in a database has to be explicitly defined by the user using an expression of the following form:

```
CLASS C isa C1, ..., Cn T
```

where C, C1, ..., Cn are classes and T is a complex type. It defines the class C as a subclass of the classes C1,...,Cn with a directly associated complex type T. If C is not a subclass of any classes, then it can be defined using another form:

```
CLASS C T
```

The schema of an object-oriented database is a set of class definitions.

The schema for the scene in Section 1 is defined as follows:

```
CLASS InScene [Object: LineObject, Loc: Point]
CLASS LineObject [Name: String, Lines: {Line}]
CLASS CLineObject isa LineObject
        [Circle: [Center: Point, Radius: Int ]]
Line    = [ From: Point, TO: Point ]
Point   = [ X: Int, Y: Int ]
```

In the schema, we have three classes InScene, LineObject, CLineObject and two named complex types Line, Point. The classes InScene, LineObject are the roots of two class hierarchies. The class CLineObject is a subclass of LineObject. Thus the instances of CLineObject are also instances of LineObject. The class InScene is directly associated with a complex type

[Object: LineObject, Loc: Point],

which is similar to the complex type InScene in the last section except that the class LineObject is used in place of the atomic type String. Therefore, only an instance of LineObject (which can also be the instance of CLineObject) can be used as the value of the attribute Object, rather than an arbitrary string. The class LineObject is directly associated with a complex type

[Name: String, Lines: {Line}]

which is the same as the complex type LineObject in the last section. The class CLineObject is directly associated a complex type

[Circle: [Center: Point, Radius: Int]]

and also inherits the complex type

[Name: String, Lines: {Line}]

from the class LineObject. Thus the complex type associated with the class CLineObject is

```
[Name: String, Lines: {Line},
 Circle: [Center: Point, Radius: Int]]
```

which is also the same as the complex type CLineObject in the last section.

Each object in a database has to be explicitly defined by the user using an expression of the following form:

```
C: Oid V
```

where C is a class, Oid is an object identifier, and V is a complex object. It defines an object consisting of an identifier Oid and a complex object V with C as its primary class.

The database instance for the scene in Section 1 consists of a set of objects as follows.

```
InScene: ObjIn1
    [Object: ObjLO1, Loc: Point [X: 10, Y:235]]

InScene: ObjIn2
    [Object: ObjLO1, Loc: Point [X: 130, Y:235]]

InScene: ObjIn3
    [Object: ObjLO1, Loc: Point [X:250, Y:235]]

InScene: ObjIn4
    [Object: ObjLO1, Loc: Point [X:250, Y:235]]

InScene: ObjIn5
    [Object: ObjLO1, Loc: Point [X:250, Y:235]]
InScene: ObjIn6
    [Object: ObjLO1, Loc: Point [X:250, Y:235]]

InScene: ObjIn7
    [Object: ObjLO2, Loc: Point [X:250, Y:235]]

InScene: ObjIn8
    [Object: ObjLO2, Loc: Point [X:250, Y:235]]
  ...

LineObjects: ObjLO1
    [Name: Pine,
     Lines: {[From: [X:18,Y: 0], TO: [X:18,Y:20]],
             [From: [X:36,Y:20],TO: [X: 0, Y:20]],
             [From: [X: 0, Y:20],TO: [X:18,Y:55]],
             [From: [X:18,Y:55],TO: [X:36,Y:20]]}]

LineObjects: ObjLO2
    [Name: Maple,
     Lines: {[From: [X:10,Y: 0], TO: [X:10,Y:30]],
             [From: [X: 0, Y:25],TO: [X:10,Y:15]],
             [From: [X:10,Y:15],TO: [X:20,Y:25]]}]
  ...

CLineObjects: ObjCLO1
    [Name:Sun,
     Lines: {},
     Circle:[Center: [X:10,Y:20], Radius: 20]]

CLineObjects: ObjCLO2
    [Name: Standman,
     Lines: {[From: [X:10,Y:20],TO: [X:10,Y:10]],
             [From: [X:10,Y:15],TO: [X: 0, Y:10]],
             [From: [X:10,Y:15],TO: [X:20,Y:10]],
             [From: [X: 5, Y: 0], TO: [X:10,Y:10]],
             [From: [X:10,Y:10],TO: [X:15,Y: 0]]},
     Circle: [Center: [X:10,Y:25], Radius: 5]]
  ...
```

A fundamental feature of object-oriented databases is that only objects can exists in the database. Complex objects alone are not objects and cannot exist independently. In other words, complex objects are not treated as first-class citizen. This causes some problems. For the above example, the object identifiers ObjIn1, ObjIn2, ... are not really needed for any purpose. In fact, it is burdensome to use them here.

Let us see, if complex objects were supported as first-class citizens as objects, how could we benefit with our graphic database application.

First, we could define the schema as follows.

```
CLASS LineObject [Name: String, Lines: {Line}]
CLASS CLineObject isa LineObject
        [Circle: [Center: Point, Radius: Int ]]
InScene= [ Object: Object, Loc: Point ]
Line    = [ From: Point, TO: Point ]
Point   = [ X: Int, Y: Int ]
```

Then we could use pure tuples in the complex object relation InScene to represent individual objects in the scene as follows.

```
InScene:
    {[Object: ObjLO1,Loc: [X: 10, Y:235]],
     [Object: ObjLO1,Loc: [X: 130,Y:235]],
     [Object: ObjLO1,Loc: [X: 250,Y:235]],
     [Object: ObjLO1,Loc: [X: 370,Y:235]],
     [Object: ObjLO1,Loc: [X: 490,Y:235]],
     [Object: ObjLO1,Loc: [X: 610,Y:235]],
     ...
    }
```

The problem with object-oriented databases is that even though we know that some of the objects appear at fixed intervals, we cannot use such information in the database. We have to explicitly specify where an object is located in the scene using a tuple as in the complex object database. Therefore, if we want to increase or decrease the distance between them, we have to modify all of them.

4 Deductive Databases

Deductive databases stem from the integration of logic programming and relational database techniques [13, 19, 37, 41]. It combines the benefits of the two approaches, such as representational and operational uniformity, reasoning capabilities, recursion, declarative querying, efficient secondary storage access, etc. Datalog, a restricted form of logic programming language Prolog without functors, has been widely accepted as the standard deductive database language [13, 41]. However, Datalog is based on the traditional relational data model which imposes first normal form constraint and is found inadequate for many advanced database applications. In recent years, deductive databases have been extended to subsume complex object databases [4, 8, 14, 25, 28, 34] and object-oriented databases [1, 5, 9, 10, 23, 22, 29, 31, 40]. For simplicity, we only discuss deductive complex object databases in this section. Extension of the discussion to deductive object-oriented databases is straightforward.

In a deductive database, we can represent data either extensionally using facts or intensionally using rules.

To specify that [A1:O1,...An:On] is a tuple in the relation R in a deductive database, a fact of the following form can be used:

R [A1:O1,...,An:On]

Consider the following example:

InScene [Object: Pine, Loc: [X: 10, Y:235]]

This fact specifies that the tuple

[Object: Pine, Loc: [X: 10, Y:235]]

is in the relation InScene.

A rule is an expression of the form

A :- L1, ..., Ln

where L1, ..., Ln is the body of the rule and used for premises and A is head of the rule and used for conclusions. Rules allow new facts to be derived from existing ones.

Consider the following rule:

```
InScene [Object: Pine,
         Loc: [X:Xvalue,Y:Yvalue]] :–
    InScene [Object: Pine,
             Loc: [X:Xvalue1,Y:Yvalue]],
    Xvalue =Xvalue1 + 120, Xvalue < 650
```

where Xvalue, Xvalue1, and Yvalue, are logical variables used as place holders. This rule states that if there exist Xvalue1 and Yvalue such that the following tuple is in the relation InScene:

[Object: Pine, Loc: [X: Xvalue1, Y: Yvalue]]

and Xvalue satisfies the following conditions:

Xvalue = Xvalue1 + 120, Xvalue < 650

then we can derive the following tuple in the relation InScene.

[Object: Pine, Loc: [X: Xvalue, Y: Yvalue]]

Therefore, based on the fact and the rule above, we can infer the following facts:

InScene [Object: Pine, Loc: [X: 130, Y:235]]
InScene [Object: Pine, Loc: [X: 250, Y:235]]
InScene [Object: Pine, Loc: [X: 370, Y:235]]
InScene [Object: Pine, Loc: [X: 490, Y:235]]
InScene [Object: Pine, Loc: [X: 610, Y:235]]

In deductive databases, the user can directly update the rules but not the facts derived with the rules. Changes to the rules will result in changes to the derived facts automatically. Therefore, if we want to increase or decrease the distance between the pine trees, we just need to modify the rule.

A deductive database instance consists of an extensional database (EDB) which is a set of facts and an intensional database (IDB) which is a set of rules.

The extensional and intensional database for the scene in Section 1 are shown as follows.

EDB

```
InScene[Object:Pine,        Loc:[X:130,Y:235]]
InScene[Object:Maple,       Loc:[X: 90,Y:110]]
InScene[Object:Maple,       Loc:[X:120,Y:140]]
InScene[Object:Standman,    Loc:[X: 50,Y:215]]
InScene[Object:Walkman,     Loc:[X:110,Y:215]]
InScene[Object:Bungalow,    Loc:[X: 50,Y:220]]
InScene[Object:Twostorey   ,Loc:[X:400,Y:220]]
InScene[Object:Sun,         Loc:[X:500,Y:500]]
...
LineObjects: ObjLO1
 [Name: Pine,
  Lines: {[From: [X:18,Y: 0], TO: [X:18,Y:20]],
          [From: [X:36,Y:20],TO: [X: 0,Y:20]],
          [From: [X: 0,Y:20],TO: [X:18,Y:55]],
          [From: [X:18,Y:55],TO: [X:36,Y:20]]}]
LineObjects: ObjLO2
 [Name: Maple,
  Lines: {[From: [X:10,Y: 0], TO: [X:10,Y:30]],
          [From: [X: 0,Y:25],TO: [X:10,Y:15]],
          [From: [X:10,Y:15],TO: [X:20,Y:25]]}]
CLineObjects: ObjCLO1
 [Name:Sun,
  Lines: {},
  Circle:[Center: [X:10,Y:20], Radius: 20]]
CLineObjects: ObjCLO2
 [Name: Standman,
  Lines: {[From: [X:10,Y:20],TO: [X:10,Y:10]],
          [From: [X:10,Y:15],TO: [X: 0,Y:10]],
          [From: [X:10,Y:15],TO: [X:20,Y:10]],
          [From: [X: 5,Y: 0], TO: [X:10,Y:10]],
          [From: [X:10,Y:10],TO: [X:15,Y: 0]]},
  Circle:  [Center: [X:10,Y:25], Radius: 5]]
CLineObjects: ObjCLO3
 [Name: Walkman,
  Lines: {[From: [X:10,Y:20],TO: [X:10,Y:10]],
          [From: [X:10,Y:15],TO: [X: 0,Y:10]],
          [From: [X:10,Y:15],TO: [X:20,Y:10]],
          [From: [X: 5,Y: 0], TO: [X:10,Y:10]],
          [From: [X:10,Y:10],TO: [X:15,Y: 0]]},
  Circle:  [Center: [X:10,Y:25], Radius: 5]]
...
```

IDB

```
InScene [Object: Pine,
        Loc: [X:Xvalue,Y:Yvalue]] :–
    InScene [Object: Pine,
            Loc: [X:Xvalue1,Y:Yvalue]],
    Xvalue =Xvalue1 + 120, Xvalue < 650
InScene [Object: Maple,
        Loc: [X:Xvalue,Y:Yvalue]] :–
    InScene [Object: Maple,
            Loc: [X:Xvalue1,Y:Yvalue]],
    Xvalue =Xvalue1 + 50, Xvalue < 550
InScene [Object: Bungalow,
        Loc: [X:Xvalue,Y:Yvalue]] :–
    InScene [Object: Bungalow,
            Loc: [X:Xvalue1,Y:Yvalue]],
    Xvalue =Xvalue1 + 120, Xvalue < 400
InScene [Object: Twostorey,
        Loc: [X:Xvalue,Y:Yvalue]] :–
    InScene [Object: Twostorey,
            Loc: [X:Xvalue1,Y:Yvalue]],
    Xvalue =Xvalue1 + 120, Xvalue < 550
InScene [Object: Walkman,
        Loc: [X:Xvalue,Y:Yvalue]] :–
    InScene [Object: Walkman,
            Loc: [X:Xvalue1,Y:Yvalue]],
    Xvalue =Xvalue1 + 180, Xvalue < 650
InScene [Object: Standman,
        Loc: [X:Xvalue,Y:Yvalue]] :–
    InScene [Object: Standman,
            Loc: [X:Xvalue1,Y:Yvalue]],
    Xvalue =Xvalue1 + 180, Xvalue < 650
```

Current deductive database systems will evaluate the rule and derive tuples once they are queried and the results are not stored into the database. For static graphic data, this may be reasonable. However, for dynamic graphic data such as animation, this may be a problem as query process may take some time. In this case, intensional data should be materialized so that fast response is possible. The problem with this method is that of update propagation. That is, if we modify an extensional tuple or a rule, then intensional tuples derived with it must be updated. Unfortunately, not much work has been down in this aspect in deductive database area.

5 Conclusion

In this paper, we have investigated how complex object databases, object-oriented databases, and deductive databases can be used to store graphic data and discussed the related problems. Object-oriented databases provide better support for object sharing and simplify update management with object identity than complex object databases. Their support for inheritance allows data to be defined in an incremental way. Deductive databases allow data to be represented extensionally using facts and intensionally using rules. It is obvious that deductive object-oriented databases are the most promising for storing graphic data based on the discussions.

In the past several years, we have design and developed such kind deductive object-oriented database system called Rule-based Object Language (abbreviated ROL) [29, 30] with object-oriented graphic databases in mind. It is available over the Internet for anonymous ftp from *ftp.cs.uregina.ca/pub/rol*. The ROL system directly supports object-oriented graphics as discussed in this paper and provides a graphic user interface called drawROL which allows the user to create, modify, display, and query graphic data stored in a ROL database. The system has been downloaded by thousands of users world-wide and used for various data and knowledge-based applications since it's first release.

Currently, we are extending the ROL system and drawROL to support animation. We are also investigating how to use ROL for 3-D graphic data, CAD databases, and multimedia databases which involve extensive graphic data of high complexity and developing corresponding tools for them.

Acknowledgments

This work has been partly supported by the Natural Sciences and Engineering Research Council of Canada.

References

[1] S. Abiteboul. Towards a deductive object-oriented database language. *Data and Knowledge Engineering*, 5(2):263–287, 1990.

[2] S. Abiteboul and N. Bidoit. Non first normal form relations: An algebra allowing data restructuring. *J. Computer and System Sciences*, 33(3):361–393, 1986.

[3] S. Abiteboul, P. C. Fischer, and H.J. Schek, editors. *Proceedings of the International Workshop on Theory and Applications of Nested Relations and Complex Objects in Databases*, Darmstadt, Germany, 1987. Springer-Verlag LNCS 361.

[4] S. Abiteboul and S. Grumbach. COL: A logic-based language for complex objects. *ACM TODS*, 16(1):1–30, 1991.

[5] S. Abiteboul and P. C. Kanellakis. Object identity as a query language primitive. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 159–173, Portland, Oregon, 1989.

[6] Serge Abiteboul and Catriel Beeri. The power of languages for the manipulation of complex values. *VLDB Journal*, 4(4):727–794, 1995.

[7] A. Albano, G. Ghelli, and R. Orsini. Fibonacci: A programming language for object databases. *VLDB Journal*, 4(3):403–444, 1995.

[8] F. Bancilhon and S. Khoshafian. A calculus for complex objects. *J. Computer and System Sciences*, 38(2):326–340, 1989.

[9] M. L. Barja, A. A. A. Fernandes, N. W. Paton, M. H. Williams, A. Dinn, and A. I. Abdelmoty. Design and implementation of ROCK & ROLL: a deductive object-oriented database system. *Information Systems*, 20(3):185–211, 1995.

[10] F. Cacace, S. Ceri, S. Crepi-Reghizzi, L. Tanca, and R. Zicari. Integrating object-oriented data modelling with a rule-based programming paradigm. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 225–236, 1990.

[11] M. Carey, D. DeWitt, and S. Vanderberg. A data model and query language for EXODUS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 413–423, Chicago, Illinois, 1988.

[12] R.G.G. Cattell, editor. *The Object Database Standard: ODMG-93, Release 1.2.* Morgan Kaufmann, 1996.

[13] S. Ceri, G. Gottlob, and T. Tanca. *Logic Programming and Databases.* Springer-Verlag, 1990.

[14] Q. Chen and W. Chu. Hilog: A high-order logic programming language for non-1nf deductive databases. In W. Kim, J.M. Nicolas, and S. Nishio, editors, *Proceedings of the International Conference on Deductive and Object-Oriented Databases*, pages 431–452, Kyoto, Japan, 1989. North-Holland.

[15] L. Colby. A recursive algebra and query optimization for nested relations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 124–138, Portland, Oregon, 1989.

[16] O. Deux and others. The story of O_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, 1990.

[17] D. H. Fishman, B. B., H. P. Cate, E. C. Chow, T. Connors, J. W. Davis, N. Derrett, C. G. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M. A. Neimat, T. A. Ryan, and M. C. Shan. Iris: An object-oriented database management system. *ACM Trans. on Office Information Systems*, 5(1):48–69, 1987.

[18] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison Wesley, 2 edition, 1990.

[19] H. Gallaire, J. Minker, and J. M. Nicolas. Logic and databases: A deductive approach. *ACM Computing Surveys*, 16(2):153–186, 1984.

[20] R. Hull. A survey of theoretic research on typed complex database objects. In J. Pardaens, editor, *Databases*, pages 193–256. Academic Press, 1987.

[21] H. Ishikawa, F. Suzuki, F. Kozakura, A. Makinouchi, M. Miyagishima, Y. Izumida, M. Aoshima, and Y. Yamane. The model, language, and implementation of an object-oriented multimedia knowledge base management system. *ACM TODS*, 18(1):1–50, 1993.

[22] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42(4):741–843, 1995.

[23] M. Kifer and J. Wu. A logic for programming with complex objects. *J. Computer and System Sciences*, 47(1):77–120, 1993.

[24] Won Kim. *Introduction to Object-Oriented Databases.* The MIT Press, 1990.

[25] G. M. Kuper. Logic programming with sets. *J. Computer and System Sciences*, 41(1):44–64, 1990.

[26] C. Lecluse and P. Richard. The O_2 database programming language. In *Proceedings of the International Conference on Very Large Data Bases*, pages 411–422, Amsterdam, The Netherlands, 1989. Morgan Kaufmann Publishers, Inc.

[27] Mark Levene and George Loizou. Semantics for null extended nested relations. *ACM TODS*, 18(3):414–459, 1993.

[28] M. Liu. Relationlog: A typed extension to datalog with sets and tuples (extended abstract). In *Proc. Intl. Logic Programming Symp. (ILPS '95)*, pages 83–97, Portland, Oregon, U.S.A., December 4-7 1995. MIT Press.

[29] M. Liu. ROL: A deductive object base language. *Information Systems*, 21(5):431 – 457, 1996.

[30] M. Liu. Storing graphic data in databases. In H. R. Arabnia, editor, *Proc. Intl. Conf. on Image Science, Systems, and Technology (CISST '97)*, pages 197–206, Las Vegas, USA, June 30-July 3 1997. CSREA Press.

[31] D. Maier. A logic for objects. Technical Report CS/E-86-012, Oregon Graduate Center, Beaverton, Oregon, 1986.

[32] W. Y. Mok, Y. NG, and D. W. Embley. A normal form for precisely charaterizing redundancy in nested relations. *ACM TODS*, 21(1):77–106, 1996.

[33] K. Morris, J. D. Ullman, , and A. V. Gelder. Design overview of the Nail! system. In *Proceedings of the International Conference on Logic Programming*, pages 554–568, London, England, 1986. MIT Press.

[34] Shamim Naqvi and Shalom Tsur. *A Logical Language for Data and Knowledge Bases*. Computer Science Press, 1989.

[35] Z. Meral Ozsoyogly and Li-Yan Yuan. A new normal form for nested relations. *ACM Trans. on Database Systems*, 12(1):111–136, 1987.

[36] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. CORAL: Control, relations and logic. In *Proceedings of the International Conference on Very Large Data Bases*, pages 238–250, Vancouver, British Columbia, Canada, 1992. Morgan Kaufmann Publishers, Inc.

[37] R. Reiter. Towards a logical reconstruction of relational database theory. In M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling*, pages 191–233. Springer-Verlag, 1984.

[38] M. A. Roth, H. F. Korth, and D. S. Batory. SQL/NF: A query language for ¬1NF relational databases. *Information Systems*, 12(1):99–114, 1987.

[39] M. A. Roth, H. F. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *ACM TODS*, 13(4):389–417, 1988.

[40] D. Srivastava, R. Ramakrishnan, D. Srivastava, and S. Sudarshan. Coral++: Adding object-orientation to a logic database language. In *Proceedings of the International Conference on Very Large Data Bases*, pages 158–170, Dublin, Ireland, 1993. Morgan Kaufmann Publishers, Inc.

[41] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, 1989.

[42] J. Vahani, K. Ramamohanarao, K. Kemp, D. B. Somogyi, and P. J. Stucky. Design overview of the Aditi deductive database system. In *Proceedings of the International Conference on Data Engineering*, pages 240–247, Kobe, Japan, 1991. IEEE Computer Society.