2000

ADO: An Active Distributed Object-Oriented Database Model

Ling Feng
University of Tilburg, ling@kub.nl

Allan Wong
Hong Kong Polytechnic University, csalwong@comp.polyu.edu.hk

ADO: An Active Distributed Object-Oriented Database Model

Ling Feng, Tilburg University, InfoLab, PO Box 90153, 5000 LE Tilburg, The Netherlands, ling@kub.nl
Allan Wong, Dept. of Computing, Hong Kong Polytechnic University, China, csalwong@comp.polyu.edu.hk

Abstract

Object-oriented databases are emerging as a new-generation database technology for complex applications. In this paper, we present an active distributed object-oriented (ADO) database model, which can capture not only *passive behaviors* but also *active behaviors* of complex objects. Besides, the *distribution* nature of objects can be reflected from within the model as well. Based on the model, we have developed a prototype active distributed object-oriented database management system, and applied it to a housing property management application.

Keywords: *Object-oriented database, object, active object, object distribution, and database model.*

1 Introduction

Object-oriented database systems are emerging as the new-generation database technology especially for complex applications (e.g., CAD/CAM, CASE, GIS, AI, etc.) in terms of rich data types, efficient support for complex objects, and computing power. Usually, these non-standard application areas require timely responses to critical situations, sophisticated constraint management and adaptiveness to changing business policies. It is thus desirable for the next-generation database systems to be *active* in the sense that they are able to react *automatically* rather than *passively* to certain events by means of knowledge (e.g., triggers) stored in the databases [KR98]. In addition, the rapid proliferation of computer networks has enabled users to easily access a large number of data sources scattered around different sites, making the *distributed* database management more and more important in the new era.

In this paper, we present an active distributed object-oriented (ADO) database model for our distributed object-oriented database management system. In the model, both *passive* and *active* behaviors of objects can be explicitly captured into classes. Moreover, different from previous object-oriented database models, the distribution nature of objects across sites can also be reflected in the database model. Based on the ADO model, we have successfully developed a prototype active distributed object-oriented database management system, which can manage both distributed and active objects. This system can be used in a number of complex applications, such as engineering design and manufacturing, geographic information systems, knowledge-based systems, scientific and multimedia databases, etc.

2 Related Work

Database management systems (DBMSs) have been available for three decades, originating in the form of the hierarchical and network models. In 1970, the relational data model was introduced by Codd [Cod70]. It is based on a simple and uniform data structure - the relation, and view data as collections of records in these relations. The relational data model revolutionized the database field by separating logical data representation from physical implementations. Its inherent simplicity laid a solid theoretical foundation for relational DBMSs, and led to the development of powerful, non-procedural query language [Cod71, Cod72a, Cod72b, Cod79].

While the relational data model hides many implementational details, it is nonetheless closer to how the DBMS stores data than to how a user thinks about the underlying real-world enterprise. To bridge this gap, several semantic data models have been proposed to assist in the process of database design [Abr74, AM86, HM81]. Semantic models allow the user to come up with a good initial description of the data in an enterprise. Research in semantic modeling has articulated a number of constructs which provide mechanisms for representing structurally complex interrelations between data (e.g., ISA relationships), typically arising in the real-world [HK87, HK90]. A widely used semantic data model called the *entity-relationship* model can facilitate pictorially description of entities and the rela-

tionships amongst them [Che76].

More recently, inspired by the object-oriented paradigm from programming languages, database researchers turn their attention towards incorporating the behavioral aspect of data into database models, and develop object-oriented data models [KBC+88, Kim90b, Kim90a, LRV88, PS87, MD90, Fe90, ADM+89, HtH98]. Basically, an object-oriented data model is a set of object-oriented concepts for modeling data. It allows the explicit representation of object classes (or abstract data types). Each object in a database is identified by its surrogate rather than by its value. The methods (operations) which are encapsulated within objects can be inherited from one class to another [RBP+91, ZM90, Kim93, DD95].

Besides building a completely new object-oriented data model from the ground up, database researchers pursue other alternative approaches, including the development of object-oriented middleware to provide a object programming layer on top of relational database systems, and the extension of relational data model with built-in object capabilities to object-relational data models [CD96, CCN+99, Sto96, LOL98]. Different from object-oriented data models, object-relational data models start with the relational model and SQL query language. The attractiveness of this approach is that object capabilities can be added to industrially-proven database systems that already offer many valuable data management functions [CCN+99].

Nowadays, object-based data models have strongly enhanced database support for applications where complex objects play a central role [DHR96]. In the area of standards, SQL3 is moving in the direction to incorporate objects into relational data model. The Object Database Management Group (ODMG) has drafted object-oriented database standards for an object data language (ODL), an object query language (OQL), and a C++ programming interface for manipulating and querying object databases [Cat96].

3 The ADO Database Model

The constituent components of the ADO database are objects, classes and their partial relationships. Each object is instantiated from a class. The class defines the structure of an object, as well as its dynamic behaviors. All classes in the database form an inheritance hierarchy, where a class can inherit all the structures and behaviors from its superclasses. In addition, a class can reference other classes to become an aggregate class.

We start a formal description of the ADO database model with the following notations:

- A series of basic types $\mathcal{D}_1, \ldots, \mathcal{D}_n$, whose domains are denoted as $dom(\mathcal{D}_1), \ldots, dom(\mathcal{D}_n)$.

  Let $\mathcal{DOM} = \cup_{i=1}^{n} dom(\mathcal{D}_i)$.

- A finite set of attribute names $\mathcal{ATT}$.
- A countably infinite set of identifiers $\mathcal{ID}$.

3.1 Values

Definition 3.1 *A* **value** *can be*
1) *an* **atomic value** $v \in \mathcal{DOM}$; *or*
2) *a* **constructional value** v, *where*

- $v = \{v_1, v_2, \ldots, v_m\}$, *for* $\forall j$ $(1 \leq j \leq m)$, *either* $v_j \in \mathcal{ID}$ *or* $v_j \in dom(\mathcal{D}_i)$ $(1 \leq i \leq n)$, *called a* **set value**; *or*
- $v = < v_1, v_2, \ldots, v_m >$, *for* $\forall j$ $(1 \leq j \leq m)$, *either* $v_j \in \mathcal{ID}$ *or* $v_j \in dom(\mathcal{D}_i)$ $(1 \leq i \leq n)$, *called a* **list value**; *or*
- $v = [a_1 : v_1, \ \ldots, \ a_m : v_m]$, *for* $\forall j$ $(1 \leq j \leq m)$, $a_j \in \mathcal{ATT}$ *and* $v_j \in \mathcal{DOM} \cup \mathcal{ID}$, *called a* **tuple value**.

□

An empty value is the only value that can be used as an atomic value, a set value, a list value, or a tuple value, denoted as ϕ, { }, <>, or [ ], respectively. Corresponding to atomic values, we call set, list and tuple values uniformly as constructional values. Let $\mathcal{V}$ denote the set of all possible values in the database system, including both atomic and constructional values.

3.2 Objects

Any real-world entity is an object with a system-wide unique identifier. An object encapsulates a static state and dynamic behaviors. The values of attributes of an object constitute the state of the object, and the methods associated with the object operate on its state. Since all the objects of a class have the same methods, we delegate the behavioral capabilities of objects to their classes, and simplify the object definition as follows:

Definition 3.2 *An* **object** o *is a 2-tuple* (o_{id}, o_{val}), *consisting of an identifier* o_{id} *and a value* o_{val}, *where* $o_{id} \in \mathcal{ID}$ *and* $o_{val} \in \mathcal{V}$. □

Definition 3.3 *Given two objects,* o *and* o', *they are*
1) **identical**, *if and only if* $o_{id} = o'_{id}$;
2) **equivalent**, *if and only if* $o_{val} = o'_{val}$. □

The values of two identical objects are the same. In other words, if "$o_{id} = o'_{id}$", then "$o_{val} = o'_{val}$". However, "$o_{val} = o'_{val}$" does not imply "$o_{id} = o'_{id}$".

Assume function $ref(o)$ return a set of identifiers, referenced within the value of object o.

Definition 3.4 *A set of objects O is* **closed** *if and only if the following conditions hold:*

1) *O is finite;*
2) $\forall o, o' \in O,\ o_{id} \neq o'_{id}$;
3) $\forall o \in O,\ ref(o) \subseteq \{x_{id} \mid x \in O\}$. □

3.3 Structures

Values are instances of types, which can be either basic types (e.g., *integer, float, char, boolean,* etc.), or constructional types. We call the structure of a basic type a basic structure, and the structure of a constructional type a constructional structure. A constructional structure can be built using the *set, list* and *tuple* constructors. Let $\mathcal{B}$ and $\mathcal{C}$ denote the set of basic and constructional structures, respectively. The whole set of structures in the system $\mathcal{S}$ ($= \mathcal{B} \cup \mathcal{C}$) can be recursively defined as follows:

Definition 3.5 *A* **structure** *s can be*

1) *a basic structure; or*
2) *a constructional structure, where*

- *$s = \{x\}$, where x is a structure ($x \in \mathcal{S}$), called* **set structure**; *or*
- *$s =< x >$, where x is a structure ($x \in \mathcal{S}$), called* **list structure**; *or*
- *$s = [a_1 : x_1,\ \ldots,\ a_m : x_m]$, for $\forall j\ (1 \leq j \leq m)$, $a_j \in \mathcal{ATT}$, and x_j is a structure ($x_j \in \mathcal{S}$), called* **tuple structure**.

□

Definition 3.6 *Given two tuple structures in $\mathcal{C}$, $s = [a_1 : s_1,\ \ldots,\ a_m : s_m,\ \ldots,\ a_n : s_n]$ and $s' = [a_1 : s_1,\ \ldots,\ a_m : s_m]$, s is a* **sub-structure** *of s', denoted as $s \leq_{st} s'$, if and only if s has more attributes than s', i.e., $m \leq n$. In other words, the sub-structure s' of s is more specific, while structure s is more general.* □

Assume function $refer(s)$ return the set of structures, referenced within the structure s.

Definition 3.7 *A set of structures Δ is called a* **schema**, *if and only if the following conditions hold:*

1) *Δ is finite;*
2) *Each structure has a unique structure name;*
3) $\forall s \in \Delta,\ refer(s) \subseteq \Delta$. □

Definition 3.8 *Let Δ be a schema and O be a closed object set. An* **interpretation** *of structure $s \in \Delta$ on O can be defined using a mapping function $I : \Delta \rightarrow O$ as follows:*

1) *when s is a basic structure (assume s is the structure of the basic type D_i without loss of generality), then $I(s) = \{o \mid (o \in O) \wedge (o_{val} \in dom(\mathcal{D}_i))\}$;*
2) *when s is a constructional structure,*

- *if $s =< x >$ ($x \in \Delta$), then $I(s) = \{o \mid (o \in O) \wedge (o_{val}$ is a list value$) \wedge$ (for any list element e_i in o_{val}, $e_i \in I(x))\}$;*
- *if $s = \{x\}$ ($x \in \Delta$), then $I(s) = \{o \mid (o \in O) \wedge (o_{val}$ is a set value$) \wedge$ (for any set element e_i in o_{val}, $e_i \in I(x))\}$;*
- *if $s = [a_1 : x_1,\ \ldots,\ a_m : x_m]$ ($x_1 \in \Delta, \ldots, x_m \in \Delta$), then $I(s) = \{o \mid (o \in O) \wedge (o_{val}$ is a tuple value$)$ $\wedge$ (for any attribute value e_{a_i} in o_{val}, $e_{a_i} \in I(x_i))\}$.*

□

Definition 3.9 *Let Δ be a schema and O be a closed object set.*

1) *An interpretation $I(s)$ of structure $s \in \Delta$ on O is* **smaller** *than another interpretation $I'(s)$, if and only if $I(s) \subseteq I'(s)$.*

2) *The model of schema Δ on O is the maximum interpretation I^*, where for $\forall s \in \Delta$, $I^*(s) = Max(I(s))$.* □

Theorem 3.1 *The model of a schema Δ on a closed object set O always exists.*

Proof 3.1 *Since O is a closed object set, according to Definition 3.4, there exists a finite set of interpretations on O. The theroem holds if we can prove that the union of two interpretations is also an interpretation, as the maximum interpretation is just the union of all the interpretations.*

Without loss of generality, let I and I' be any two interpretations, and $I'' = I \cup I'$. For $\forall s \in \Delta$, $I''(s)$ satisfies the property 1) in Definition 3.8 when s is a basic structure.

In the case that s is a set structure ($s = \{x\}, x \in \Delta$), according to Definition 3.8, any element in $I(s)$ or $I'(s)$ is in $I(x)$, thus, any element in $I''(s)(= I(s) \cup I'(s))$ is also in $I(x)$, satisfying the property 2) in Definition 3.8. Similarly, we can prove the tenability of the property 2) for $I''(s)$ when s is a list or tuple structure. □

3.4 Passive Behaviors

We use *methods* to describe the passive behaviors of objects. On receipt of a message consisting of a name and some arguments, the object will search the matching method, and then execute the corresponding program code of the method.

Definition 3.10 *Let Δ be a schema. A* **signature** *on Δ is a mapping $s_1 \times \ldots \times s_n \rightarrow s$, where $s_1, \ldots, s_n$, $s \in \Delta$.* □

Definition 3.11 *A* **method** *m is a 3-tuple (m_{name}, m_{sig}, m_{code}), consisting of a method name m_{name}, a signature m_{sig}, and a program code m_{code}.* □

3.5 Active Behaviors

The active behaviors of objects are expressed by means of triggers, i.e., event/condition/action (ECA) rules [BM91]. When a certain event arises and the condition holds, the action will be triggered for execution.

Definition 3.12 *A* **trigger** g *is a 5-tuple* $(g_{name}, g_{event}, g_{cond}, g_{action}, g_{time})$, *consisting of a trigger name* g_{name}, *an event in terms of a method* g_{event}, *a predicate* g_{cond} *defined on attributes, a triggered method* g_{action}, *and the time* g_{time} *to execute* g_{action}. *The time* g_{time} *can be either immediately following the triggering method* g_{event}, *or before the commit of* g_{event}. □

3.6 Classes

A class is used to create a set of objects that share the same structure, and the same passive and active behaviors.

Definition 3.13 *Let* Δ *be a schema. A* **class** c *is a 4-tuple* $(c_{name}, c_{struct}, c_{methods}, c_{triggers})$, *consisting of a class name* c_{name}, *a tuple structure* $c_{struct} \in \Delta$, *a set of methods* $c_{methods}$, *and a set of triggers* $c_{triggers}$ *associated with class* c. □

We define two partial relationships (i.e., *is-a* and *part-of*) of classes as follows:

Definition 3.14 *Class* c *is a* **subclass** *of class* c', *denoted as* $c \leq_{is-a} c'$, *if and only if the following conditions hold:*

1) $c_{struct} \leq_{st} c'_{struct}$;
2) $\forall m' \in c'_{methods}, \exists m \in c_{methods}$, *such that* $m_{name} = m'_{name}$;
3) $\forall g' \in c'_{triggers}, \exists g \in c_{triggers}$, *such that* $g_{name} = g'_{name}$; □

Definition 3.15 *Let* $[a_1 : s_1, \ldots, a_n : s_n]$ *be the structure of class* c'. *Class* c *is an* **aggregate class** *of* c', *denoted as* $c \leq_{part-of} c'$, *if and only if* $\exists s_k (1 \leq k \leq n)$ $(s_k = c_{struct})$. □

Definition 3.16 *Two classes* c *and* c' *are* **relevant**, *denoted as* $c \overset{rel}{\longleftrightarrow} c'$, *if and only if either of the following conditions holds:*

1) $c \leq_{is-a} c'$, *or* $c' \leq_{is-a} c$;
2) $c \leq_{part-of} c'$, *or* $c' \leq_{part-of} c$. □

Definition 3.17 *A* **class lattice** Γ *is a set of classes, if and only if the following conditions hold:*

1) Γ *is finite;*
2) *Each class has a unique class name. That is,* $\forall c, c' \in \Gamma$, $c_{name} \neq c'_{name}$.
3) $\forall c \in \Gamma$, $\{c' | c' \leftrightarrow_{class} c\} \subseteq \Gamma$;
4) *From a unique root class* $c_{root} \in \Gamma$, *all other classes in* $\Gamma - \{c_{root}\}$ *can be traced. That is, for* $\forall c \in \Gamma$, *either* $c \overset{rel}{\longleftrightarrow} c_{root}$, *or* $\exists c_1, \ldots, c_n \in \Gamma$, *such that* $c \overset{rel}{\longleftrightarrow} c_1 \overset{rel}{\longleftrightarrow} \ldots \overset{rel}{\longleftrightarrow} c_n \overset{rel}{\longleftrightarrow} c_{root}$. □

3.7 Distributed Databases

A database schema is a class lattice, whose instances are objects belonging to the classes in this lattice. The value of each such object, in terms of its attributes, matches the structure of the associated class. In other words, the object value must be within the maximum interpretation of the tuple structure of the class.

In our system, objects of a class can be horizontally partitioned based on either the class itself or its relevant class. The former *self-class-based partition* is similar to the traditional horizontal method used in relational databases, where partitioning is performed according to a predicate defined on attributes of this class. For the latter *relevant-class-based partition*, partitioning of a class arises from the fragmentation of its relevant classes. Thus, the partitioning predicate is defined on the attributes of its relevant class instead.

Definition 3.18 *Let* Γ *denote a class lattice. A* **fragmentation** f *of class* $c \in \Gamma$ *is a tuple* $(f_{Pred}, f_{node_1}, \ldots, f_{node_{f_n}})$, *specifying the partitioning predicate* f_{Pred} *and the nodes* $f_{node_1}, \ldots, f_{node_{f_n}}$ *to which the corresponding partition is distributed.* □

Assume function $Object(c)$ return the whole set of objects in class c, and function $FObject(c, f_i)$ return a set of objects in the fragment f_i of class c.

Definition 3.19 *Let* Γ *denote a class lattice. A* **fragmentation schema** *of class* $c \in \Gamma$ *is a tuple* $\Lambda = (\Lambda_{type}, \Lambda_{ref}, \Lambda_{f_1}, \ldots, \Lambda_{f_{n_c}})$, *where*

1) Λ_{type} *is the partitioning method, which can be either* "self-class-based" *or* "relevant-class-based";

2) *if* Λ_{type} = *"relevant-class-based", then* Λ_{ref} *is the relevant class where the partitioning predicates apply;*

3) $\Lambda_{f_1}, \ldots, \Lambda_{f_{n_c}}$ *is a list of fragments of class* c, *subject to the following constraints:*

- *For* $\forall o \in FObject(c, f_i)$ $(1 \leq i \leq n_c)$, *it satisfies the predicate of fragmentation* Λ_{f_i} *if* Λ_{type} = *"self-class-based"; Otherwise, its relevant object* o' *(where* $o' \in FObject(c', f_i)$ *and* $o \overset{rel}{\longleftrightarrow} o'$*) satisfies the predicate of fragmentation* Λ_{f_i}.
- *For* $\forall o \in Object(c)$, *there exists only one fragment* f_i $(1 \leq i \leq n_c)$, *such that* $o \in FObject(c, f_i)$.
- $Object(c) = \cup_{i=1}^{n_c} FObject(c, f_i)$
- $\forall f_i, f_j \in \{f_1, \ldots, f_{n_c}\}$, $FObject(c, f_i) \cap FObject(c, f_j) = \phi$

□

Definition 3.20 *A database is a 4-tuple* $DB = (N_{DB}, \Gamma, O, \Upsilon)$, *consisting of a database name* N_{DB}, *a class lattice* Γ, *a closed object set* O, *and a class fragmentation schema set* Υ *in* Γ, *where for* $\forall o \in O$, *there exists a class* $c \in \Gamma$, *such that* $o \in I^*(c_{struct})$. □

4 A Housing Property Management Application

Based on the ADO database model, we have built up a housing property management application for a newly developed region. The core functions of the system include:

- effectively storing all the housing information, such as housing type (state, private, commercial), location, construction company, time of completion, architecture, number of rooms, decoration, etc., in the area.
- multidimensionally viewing all the transportation facilities, such as roads, railways, gas stations, etc., in the area.
- demonstrating all services, such as shops, hotels, etc., in the area.
- easily querying and managing housing information and surrounding facilities.

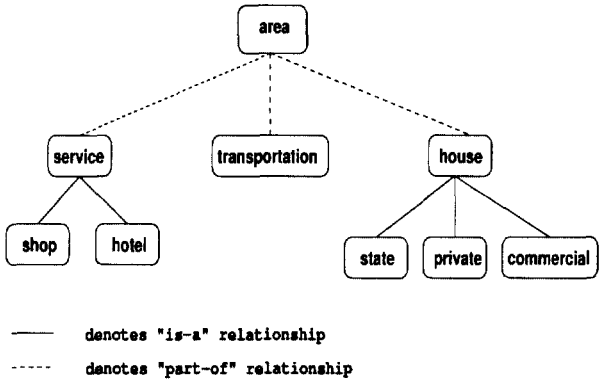

Figure 1: One portion of the application database schema

Figure 1 shows part of the application database schema. Each entity in the schema is represented by a class.

```
1) class-name:      AREA
   class-structure:
      [  areaNo:      integer;
         name:        char[12];
         position:    char[4];
         picture:     IMAGE;
         sound:       VOICE;
         transport:   Struct(TRANSPORT);
         service:     Struct(SERVICE);
         living:      Struct(HOUSE);
         .....    ]

   class-method:
     void showPicture (IMAGE picture) {...}
     void soundIntroduce (VOICE sound) {...}
     .....

   class-trigger:
      ( name:       synBroadcast
        event:      showPicture (IMAGE picture)
        condition:  TRUE
        time:       IMMEDIATE-AFTER
        action:     soundIntroduce (VOICE sound)
      )
      .....
```

AREA is the class name. IMAGE and VOICE are two additional data types introduced for handling image and audio multimediate objects in the system. As their values are actually big sequences of characters, we view them as a special kind of basic data types. Inside the AREA class structure, the attributes *transport, service, living* get values from the structures of class TRANSPORT, SERVICE, HOUSE, respectively. Two methods of the class AREA, *showPicture* and *soundIntroduce*, introduce the area in visual-audio ways. One trigger of the class AREA is named *synBroadcast*. Its active function is that once the *showPicture* method is called and executed, immediately following it, the *soundIntroduce* method will be automatically invoked.

```
2) class-name:      TRANSPORT
   class-structure:
      [  trafficMap:  IMAGE;
         road:        ROAD-ST;
         railway:     RAILWAY-ST;
         gasStation:  STATION-ST;
         .....    ]

   class-method:
     void displayRoad (ROAD-ST road) {...}
     void displayRail (RAILWAY-ST railway) {...}
     void displayGasStation (STATION-ST gasStation)
       {...}
       .....
```

ROAD-ST, RAILWAY-ST, and STATION-ST are three constructional data structures, defined as follows:

```
ROAD-ST =
   { [roadName: char[12], roadPosi: { <float, float> } ] }.
RAILWAY-ST =
   { [railName: char[12], railPosi: { <float, float> } ] }.
STATION-ST =
   { [stationName: char[12], stationPosi: { <float, float> } ] }.
```

```
3) class-name:      SERVICE
   class-structure:
      [  name:        char[24];
         kind:        char[12];
```

location: char[36];
payMethod: char[24];
serviceScope: TEXT;
.....]
.....

TEXT is another special basic data type introduced for handling strings of variable lengths.

4) class-name: HOTEL
superclass: SERVICE
class-structure:
[star: integer;
entertain: char[36];
breakfast: char[36];
diet: char[24];
facility: TEXT;
.....]
.....

5) class-name: SHOP
superclass: SERVICE
class-structure:
[kind: char[24];
goodsOnSale: TEXT;
guide: TEXT;
.....]
.....

6) class-name: HOUSE
class-structure:
[constructCompany: char[36];
completeTime: DATE;
architecture: char[12];
roomNumber: integer;
type: char[12];
decorate: TEXT;
location: char[24];
.....]
.....

7) class-name: PRIVATE
superclass: HOUSE
class-structure:
[ownerName: char[24];
ownerAddr: char[36];
.....]
.....

8) class-name: STATE
superclass: HOUSE
class-structure:
[organization: char[36];
purpose: TEXT;
.....]
.....

9) class-name: COMMERCIAL
superclass: HOUSE
class-structure:
[price: float;
deal: BOOLEAN;
payMethod: char[24];
.....]

class-method:
int DealStatistic (int price) {...}
/* count the number of houses above a certain price that have been dealt. */
.....

Due to space limitation, we illustrate the fragmentation design of class HOUSE and its subclasses, and omit the details of other classes' fragmentation schemas. The objects of class HOUSE, STATE, PRIVATE, COMMERCIAL are distributed based on the following schemas.

fragment-of-class: HOUSE
fragment-type : *self-class-based*
reference-class : nil
partitions: (. location = "*north*", $node_{1,4}$;
. location = "*south*", $node_2$;
. location = "*east*", $node_3$;
. location = "*west*", $node_4$;
)

fragment-of-class: STATE/PRIVATE/COMMERCIAL
fragment-type : *relevant-class-based*
reference-class : HOUSE
partitions: (. HOUSE.location = "*north*", $node_{1,4}$;
. HOUSE.location = "*south*", $node_2$;
. HOUSE.location = "*east*", $node_3$;
. HOUSE.location = "*west*", $node_4$;
)

5 Conclusion

In this paper, we present an active distributed object-oriented (ADO) database model, based on which a prototype active distributed object-oriented database management system has been developed. We have applied the system to a housing property management application, which can handle complex objects including texts, graphics, images, audio and video objects.

References

[Abr74] J.R. Abrial. Data semantics. In *Data Base Management, North-Holland*, 1974.

[ADM+89] M. Atkinson, D. DeWitt, D. Maier, F. Bancilhon, K. Dittrich, and S.B. Zdonik. The object-oriented database system manifesto. In *Proc. of the 1st Intl.*

Conf. on Deductive and Object-Oriented Databases, pages 40–57, Kyoto, Japan, 1989.

[AM86] H. Afsarmanesh and D. McLeod. A framework for semantic database models. In *New Directions for Database Systems, (G. Ariav and J. Clifford (ed.), Ablex Publishing Company*, 1986.

[BM91] C. Beeri and T. Milo. A model for active object-oriented database. In *Proc. of the 17th Intl. Conf. on Very Large Data Bases*, pages 337–349, Barcelona, Spain, September 1991.

[Cat96] R. Cattell. *The Object Database Standard: ODMG-93 (Release 1.2)*. Morgan Kaufmann Publishers, 1996.

[CCN+99] M. Carey, D. Chamberlin, S. Narayanan, B. Vance, D. Doole, S. Rielau, R. Swagerma, and N. Mattos. O-O, what's happening to DB2. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 511–512, Philadelphia, USA, June 1999.

[CD96] M. Carey and D. DeWitt. Of objects and databases: A decade of turmoil. In *Proc. of the Intl. Conf. on Very Large Data Bases*, pages 3–14, Bombay, India, September 1996.

[Che76] P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, January 1976.

[Cod70] E. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[Cod71] E. Codd. A data base sublanguage founded on the relational calculus. In *Proc. of the ACM SIGFIDET Workshop on Data Description, Access, and Control*, pages 35–68, November 1971.

[Cod72a] E. Codd. Further normalization of the data base relational model. In *Data Base Systems, R. Rustin (ed.), Prentice-Hall*, 1972.

[Cod72b] E. Codd. Relational completeness of data base sublanguages. In *Data Base Systems, R. Rustin (ed.), Prentice-Hall*, 1972.

[Cod79] E. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, December 1979.

[DD95] H. Darwen and C. Date. The third manifesto. *SIGMOD Record*, 24(1):39–49, March 1995.

[DHR96] M. Doherty, R. Hull, and M. Rupawalla. Structures for manipulating proposed updates in object-oriented databases. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 306–317, Montreal, Canada, June 1996.

[Fe90] D.H. Fishman and etc. Iris: An object-oriented database management system. In *Readings in Object-Oriented Database Systems, (S.B. Zdonik and D. Maier (ed.), Morgan Kaufmann, Inc.*, 1990.

[HK87] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[HK90] R. Hull and R. King. A tutorial on semantic database modeling. In *Research Foundations in Object-Oriented and Semantic DataBase Systems, A.F. Cárdenas and D. McLeod (ed.), Prentice-Hall*, 1990.

[HM81] M. Hammer and D. McLeod. Database description with SDM: A semantic data model. *IEEE Transactions on Database Systems*, 6(3):351–386, September 1981.

[HtH98] J.W.G.M. Hubbers and A.H.M. ter Hofstede. Exploring the jungle of object-oriented conceptual data modeling. In *Proc. of the 9th Australian Database Conference*, pages 65–76, Perth, Australia, February 1998.

[KBC+88] W. Kim, N. Ballou, H.T. Chou, J.F. Garza, D. Woelk, and J. Banerjee. Integrating an object-oriented programming system with a database system. In *Proc. of the 3rd Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications*, pages 142–152, San Diego, California, September 1988.

[Kim90a] W. Kim. Architecture of the ORION next-generation database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):109–124, March 1990.

[Kim90b] W. Kim. *Introduction to Object-Oriented Databases*. The MIT Press, 1990.

[Kim93] W. Kim. Object-oriented database systems: Promises, reality, and future. In *Proc. of the Intl. Conf. on Very Large Data Bases*, pages 676–687, Dublin, Ireland, August 1993.

[KR98] G. Kappel and W. Retschitzegger. The TriGS active object-oriented database system - an overview. *SIGMOD Record*, 27(3):36–41, September 1998.

[LOL98] H. Li, M.E. Orlowska, and C. Liu. A query system for object-relational databases. In *Proc. of the 9th Australian Database Conference*, pages 39–50, Perth, Australia, February 1998.

[LRV88] C. Lecluse, P. Richard, and F. Velez. O2, an object-oriented data model. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 424–433, Chicago, June 1988.

[MD90] F. Manola and U. Dayal. PDM: an object-oriented data model. In *Readings in Object-Oriented Database Systems, (S.B. Zdonik and D. Maier (ed.), Morgan Kaufmann, Inc.*, 1990.

[PS87] J. Penney and J. Stein. Class modification in the GemStone object-oriented DBMS. In *Proc. of the Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications*, pages 111–117, Florida, October 1987.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[Sto96] M. Stonebraker. *Object-Relational Database Systems: The Next Great Wave*. Morgan Kaufmann Publishers, 1996.

[ZM90] Z. Zdonik and D. Maier. *Readings in Object-Oriented Database Systems*. Morgan Kaufmann Publishers, 1990.