PAMUKKALE ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
PAMUKKALE UNIVERSITY ENGINEERING COLLEGE
MÜHENDİSLİK BİLİMLERİ DERGİSİ
JOURNAL OF ENGINEERING SCIENCES
YIL : 2000
CİLT : 6
SAYI : 1
SAYFA : 57-63

xPROT : OBJECT-ORIENTED PROTOTYPING MODEL

A. Nuri BAŞOĞLU
Boğaziçi Üniversity, Dept. of Management Information Systems, İstanbul

Geliş Tarihi : 06.05.1998

ABSTRACT

Even relational database management systems have now become the standard for data processing applications; object oriented database systems are being developed to meet the complex data modeling requirements. In this paper an object-based system, xPROT, is introduced which could be used for rapid program development. System includes some pre-defined methods and classes as Menu, Report, View, Table. In xPROT the schema as well as the data manipulation is expressed by the same command-based language. Expressiveness, easiness and flexibility in terms of data modeling power is investigated.

Key Words : Data modelling, Software engineering, Prototyping, Object-Orientation, Database systems

xPROT : NESNE-YÖNELİMLİ PROTOTİPLEME MODELİ

ÖZET

İlişkisel veri tabanları bilgiişlem uygulamalarında standart olmaya başlamasına rağmen, karmaşık veri modellemesinin gereklerini yerine getirmek için nesne yönelimli sistemler geliştirilmektedir. Bu makalede, hızlı program geliştirmek için kullanılabilecek, nesne-tabanlı bir sistem, xPROT tanıtılmaktadır. Sistem bazı önceden tanımlanmış metod ve Menu, Report, View, Table gibi sınıfları içermektedir. xPROT'da veri şeması olduğu kadar, veri işleme de aynı komut bazlı dil ile ifade edilmektedir. Sistemin ifade yeterliliği, kolaylığı ve esnekliği, veri modelleme gücü açısından incelenmektedir.

Anahtar Kelimeler : Veri modelleme, Yazılım mühendisliği, Prototip, Nesne-yönelim, Veri tabanı sistemleri

1. INTRODUCTION

Relational database management systems (RDBMS) have been the standard for business data processing. Its success has greatly been due to flexibility and ease of use (Ullman, 1988, Date, 1990). However an application of medium size may contain tens of tables and hundreds of columns. Data modelling capabilities of RDBMS are too limited. It's difficult to build complex, large-scale and data intensive applications using RDBMS.

Relational systems are mainly based on calculus-based languages (SQL, QUEL) or other non-procedural languages that can define and manipulate data (Ahad and Yao, 1993). Besides in some cases procedural languages are in use. Various extensions of SQL and Quel-Ingres are reported by Dar, Snodgrass and Su: SQL/TC, Quel*, POSTQUEL, TQuel, Traversal Recursion, μ Calculus, α-extended Relational Algebra, Aggregate Closure (Dar and Agrawal, 1993; Snodgrass et all., 1993; Su and Lam 1993; Ahad and Yao, 1993).

The object-oriented concept was first introduced in SIMULA, and made it very popular through the language SMALLTALK. They allow richer structural constructs and behavioural properties of objects to be specified at the logical level independent of their physical implementations (Hughes, 1991; Bekke, 1992). There is a strong trend in the research community toward extending object-oriented languages in the direction of databases (Dewitz, 1996). More recently several object-oriented databases are proposed. The partial list compiled from Mohan, Su, Yoo, Gray and Hughes includes GEMSTONE, GEM, Iris, Ariel, ODE, EXODUS, ALLTALK, POSTGRES,

PGRAPHITE, Trellis/Owl, Vbase, OQL, O_2, ONTOS, ORION, OASIS, PROBE, ENCORE, OSAM*, SSONET, FORM (Parsaye et all., 1989; Hughes, 1991; Gray et all., 1992; Mohan and Kashyap 1993, Su and Lam, 1993, Yoo and Sheu, 1993).

In this paper an object-based system is introduced which could be used for rapid program development. System includes some pre-defined classes as Menu, Report, View, Table, etc., and pre-defined methods as AddNewRecord, DeleteRecord, SearchByIndex, Filter, ReportGenerate,etc. Formal prototyping languages with a fair degree of client orientation have been developed: JSD, ERAE, INFOLOG, USE, SF, PGRAPHITE, VIZLA, CQL (Hughes, 1991; Berztiss, 1993). Major motivation behind this project is to develop a system that will meet the below and other requirements: (Lieberherr and Xio,1993; Gyssens, 1994).

- Easy to write and read
- Flexible and adaptive to changes
- Rapid program development
- Power to express a variety of different situations.

2. OBJECTS

Object is a real-world entity that includes a unique identifier. The components of an object are attributes and methods, where attributes describe the object and methods represent the operations that can be applied to an object. Class is collection objects with same attributes and methods. Classes are hierarchically structured based on the is_a association. Subclasses inherit attributes and methods from their superclasses. When a class is permitted to have more than one superclass, we speak of multiple inheritance. The Classes Menu, Report, View, Table, Composite and Atomic are introduced below, where some people may prefer using the word metaclass instead of class (Parsaye et all., 1989).

Table 1 shows the system pre-defined standard classes and Table 2 presents related built-in methods and properties.

Table 1. Standard Database Classes

Atomic Objects	:	Non-decomposable unit of objects
Composite Objects	:	Composition of atomic and composite objets
Table Objects	:	Physical data object, composed of columns
View Objects	:	A joined and/or filtered set of table objects
Report Objects	:	Reporting objects
Menu Objects	:	Branching mechanisms

2. 1. Abstract Data Objects (ADO)

Abstract Data objects are the simplest objects where basic data structures and their internal methods are defined using the system primitive types (char, num, date, ...) and user-defined data objects. Value of these objects is volatile and they are used to model the complex data. Each object may have some pre-defined properties and a free vector. There are two kinds of ADO's. Atomic (AO), Composite(CO).

2. 2. Atomic Objects (AO)

AO is a data object with a non-decomposable data unit and some properties. An atomic object may also be defined referring to an existing AO where properties may inherit from ancestor (Figure1). For definition of an AO, a unique name as an identifier and Type are mandatory. This hierarchy may repeat many times.

```
AO Comm TYPE char LENGTH 10 ;
                                PICTURE ''@R (###)### ####"
AO Phone TYPE Comm EXP "Phone"
AO Fax    TYPE Comm EXP "Fax"
AO Shift TYPE num EXP "Working Shift" ;
                              POSTACT (1 ≤ Shift .and. Shift ≤ 3)
AO Time TYPE Char LENGTH 4 ;
                    PICTURE "@R ##:##" POSTACT ValidTime(x)
AO xTime TYPE Time HEAD "Start Time"
AO Pcode TYPE Char LENGTH 7 PICTURE ;
                "P!!####"  EXP "Product Code" HEAD "P code "
AO Ccode TYPE Char LENGTH 5 PICTURE "@!" ;
                        EXP "Customer Code" HEAD "C code "
```

There are several data objects defined above. Objects Phone and Fax are of type object Comm. Object xTime is of type Time where also the behavioural features inherit. Object Shift's value is restricted to keep between 1 and 3 by a post edit action method.

2. 3. Composite Objects (CO)

CO is a composition of many atomic AOs or COs. While defining an atomic or composite object, inherited properties may be modified, dropped or pre-defined properties may be assigned an expression. For the terminals of object hierarchy tree of AO and CO, Length (+Decimal) is mandatory for objects whose type is character or numeric. This hierarchy may repeat many times (Figure 1). Object Process is a composition of objects sDate, Stime, Etime, Shift. Columns which are common in different tables may be also be defined as AO. Objects Pcode and Ccode will be refered from many objects.

```
CO Process
    COMPOSITION
        sDate TYPE date INITVALUE Today()
        STime TYPE xTime POSTACT sVAR[Shift0] ≤ Stime ;
                                .and. Stime ≤ eVAR[Shift0]
```

```
        Etime TYPE  xTime POSTACT STime ≤ Etime .and. ;
                                        Etime ≤ eVAR[wShift]
        Shift0 TYPE Shift  EXP "Current Shift " HEAD "Shift"
    METHOD
        - - - -
ENDOBJECT
```

Property Process has four embedded AO, where their name is not declared explicitly. One component of Object production is STime that is defined as member of the CO-Process. Property of Stime inherits from object xTime that borrows those from object Time. Reference name to any column on TO (Table Object) will be taken from last AO name while descending on the object hierarchy tree. TO and CO can not hold two member objects that has same reference name, that is, object identity.

Table 2. List of Basic Properties of AO, CO and TO's is Given Below

TYPE	Primitive types or user-defined ADO's
LENGTH	Numerical value within appropriate ranges
DECIMAL	Numerical value
PICTURE	Character or code block
EXP	Description. (character)
HEAD	Heading in a table format display. (character)
INITVALUE	İnitial value. Expression or code block which return a value of type of objects' type
PREACT	Code block which is invoked before edit of object
INACT	Code block or help routine
POSTACT	Code block which is invoked after edit of object
COLOR	Code block which determines its color
VECTOR	A list with indefinite length keeping various types of data
NONULL	Can not be left empty
FULL	For each digit of edited field, there should be non-space value
MULTIVALUE	This will give a lists of possible item, among which user may makes a selection

Additional properties as Size, style, pattern and other specification of display may be added.

2. 4. Class Hierarchy and Inheritance

The database classes have different paths of inheritance, that are demonstrated below (Figure 1)

class	may inherit from
primitive types : p	-
Atomic Object : AO	p
Composite Object : CO	p, AO
Physical Data Object : TO:	p, AO, CO
View Object : VO	TO
Report Object : RO	TO, VO
Menu Object : MO	TO, VO, RO

Figure 1. Class hierarchy and inheritance

In a CO, inheritance from another CO would be very useful, but it may bear some presentation problems. A different definition of object Process is displayed and discussed below.

```
CO WorkPeriod
    COMPOSITION
                STime TYPE xTime POSTACT ( sVAR[Shift0] ≤ ;
                                Stime .and. Stime ≤ eVAR[Shift])
            Etime TYPE  xTime POSTACT ( STime ≤ Etime .and.;
                                Etime ≤ eVAR[Shift0])
          Shift0 TYPE Shift  EXP "Current Shift " HEAD  "Shift"
    METHODS
        - - - -
ENDOBJECT
CO Process
    COMPOSITION
        sDate TYPE date INITVALUE Today()
        Work TYPE WorkPeriod
    METHOD
        - - - -
ENDOBJECT
```

Object Process is composed of sDate and Work where Work has a type of WorkPeriod. Some naming conflicts will arise while referring to Stime, ETime or Shift components. A way to overcome this problem is to accept the last referred atomic object name as the physical file field name. In this case programmer should follow the CO tree to get the actual reference name.

There is another way of establishing an inheritance while defining the object ProcessRec as below.

```
CO ProcessRec  inherit from Process
    COMPOSITION
        sDate TYPE date INITVALUE Today()
    METHOD
        - - - -
ENDOBJECT
```

In the former definition, methods that were defined in the METHOD section of object WorkPeriod would not be accessible. Only the methods and other properties of atomic objects Stime, Etime and Shift0 will inherit. In the latter solution all objects, their properties and methods are expected to inherit. There may be different situations, where it may be preferable to select one of these solutions.

2. 5. Table Class (TO): Physical Data Class

Physical Data Objects are the objects that keep actual data of the database where instance values will persist forever until they are edited or deleted by end-user. TO is implemented as relations (table) of a relational DBMS. Hierarchy of TO is not implemented. This class is a set of data columns. Column type, picture, pre & post edit, pre & post Delete, pre & post insert methods may be defined. Below is the list of three Physical Data Objects: Product, Customer, Production. At this level the definition of the object that resides on disk as a table is given. The object relationship model

(Yourdan, 1994) is an expressive technique for documenting object classes and their attributes and relationship, and the model will be used to present the sample (Figure 2).

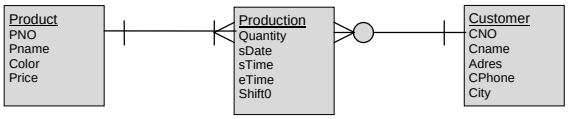

Figure 2. Object relationship model

```
TO Product
    COMPOSITION
        PNO TYPE Pcode  KEY
        Pname TYPE Char LENGTH 35
        Color TYPE Char LENGTH 2 MULTIVALUE
('RED','BLUE','WHITE')
        Price TYPE num LENGTH 5 POSTACT (Price > 0)
    RELATIONSHIP
        RELATED WITH Production DEGREE MIN 1 MAX n
    METHOD
        - - -
ENDOBJECT

TO Customer
    COMPOSITION
        CNO TYPE Ccode  KEY AUTOSEQ
        Cname TYPE Char LENGTH 30  NODUP
        Adres TYPE Char LENGTH 50
        cPhone TYPE Phone1
        City TYPE Char LENGTH 12
    RELATIONSHIP
        RELATED WITH Production DEGREE MIN 0 MAX n
ENDOBJECT
TO Production
    COMPOSITION
        PNO TYPE Pcode  KEY ;
        CNO TYPE Ccode KEY ;
        Quantity TYPE num LENGTH 6 PICTURE "###,###"
POSTACT (0<Quantity) HEAD "Q"
        Process COLOR if(Quantity> 0, CO_GREEN, CO_WHITE)
    METHOD
        preACT is KeepStatus()
        postACT is SCHEDULE()
    ACCESS
        INDEX  P1 ON Customer HEADING "..... "
        INDEX  P2 ON sDate,sTIME HEADING "Date,Time"
ENDOBJECT
```

2. 6. Relationship

There is another section, RELATIONSHIP, where it is possible to define the relationship and the degree between TOs. Minimum and maximum limits of the number of relations between instances can be declared as below. A Product may take place in many productions, that is; there is a one-to-many relationship between Product and Production.

If it is not defined Default relationship cardinality is a many-to-one (n:1). For every incoming relationship from an object an inverse relationship is defined automatically, however it can be overwritten by extra statements.

In the current sample there are three TOS's such as Product, Customer, Production. A product may be produced for many customers as well as a customer may require many products, that is; the relationship between Product and Customer is many-to-many. However this may be decomposed into two one-to-many relationships. The relationship between objects can be defined within body of objects. Physical Object Production has a special method KeepStatus() which will be invoked every time Object Production is active and method Schedule() will work automatically after user exits the object.

If property is defined as KEY or NODUP an automatic index definition statement is created.

2. 7. Alternative Definition of Table Objects

It's possible to apply two different approaches to define the object where a typical sample is given below. A TO may be defined as a composition of many AOs or a single CO. In case database administrator does not wish to allow programmer to modify the structure; a pre-defined CO is given to programmer. Then the programmer will use this pre-defined structure. When the structure should stay safe, this approach seems to be preferable. A different way is to allow programmer to define his own structure. When there is a rush or in the initial, tentative steps, the latter approach may be more preferable. Object Product is defined by ProductType object.

```
TO ProductType
    COMPOSITION
        PNO TYPE Pcode  KEY
        Pname TYPE Char LENGTH 35
        - - -
ENDOBJECT
TO Product INHERIT FROM ProductType
    RELATIONSHIP
        RELATED WITH Production DEGREE MIN 1 MAX n
ENDOBJECT
```

2. 8. View Class (VO)

View is a set of relations (tables). If it is not defined then the table at the lowest level in relation hierarchy is selected as driving table. This Class inherits properties from Tables Class (Figure 3). Basic screen attributes & modules that may be invoked by user are declared in this Class. Columns that inherit from Table may be excluded. Derived columns may be defined. Additional pre & post action of columns may be defined. Help popup lists may be filtered. Some of the properties of view class may be defined by the end user at runtime. Pre & post action of view objects may be defined. (Lee and Wiederhold 1994, Spaccapietra and Parent, 1994)

```
VO ProducPROG
    COMPOSITION
        OBJECT Customer
        OBJECT Product
        OBJECT Production
        EXPR Product->Price * Production->Quantity ;
                                    PICT "##,###,### "
```

```
     FILTER
          sDate > Today()-30
ENDOBJECT

MO Main
     OPT  "Production " MESG "...." PREACT ...
                                        POSTACT ProducPR
     OPT  "Reports  "     MESG "...." PREACT time() > "17:00"
                                              POSTACT SubMe
ENDOBJECT

MO SubMenu HORIZONTAL AT 10,12 MAXLINE 5
     EXP  "Report 1"  MESG "...." PREACT ... POSTACT Rep1
     EXP  "Report 2"  MESG "...." PREACT ... POSTACT Rep2
     ---
     ---
ENDOBJECT

RO Rep
     COMPOSITION
          OBJECT ProducPROG
     FILTER
          sDate=Today() .and. Customer->City="IST"
     ORDER
          CNO
     SUMMARY
          Sum of Quantity, Max of Quantity
ENDOBJECT
```

Figure 3. View, Report and Menu Objects definition

2. 9. Report Class (RO)

This object may inherit either from a TO or VO, but however it does not utilize many methods of its superclasses because change in persistent data is not allowed. In Report object there are many methods used for formating, filtering, ordering, aggregation (Figure 3).

2. 10. Menu Class (MO)

Menus are list of independent programs from which a user may make a selection. Menu objects are the branching mechanisms that allow user to jump to selected modules. These lists may be in different forms. Object Menu Main is the introduction menu of the system (Figure3). A menu object may invoke a View, a TO or another menu object.

View Object ProducPROG is composed of objects Customer, Product, and Production. Since the relationship between these objects had been defined, An automatic natural join will take place using the access mechanisms. View object ProductPROG is called from Main Menu object

3. METHOD DEFINITION

Methods are defined in the method section of objects. The preACT and postACT are special pre-edit and post-edit procedures. Methods may be simple equations as well as procedures coded in native language.

AO: Mainly pre-edit, post-edit, default value methods,
CO: Interaction functions between AOs which compose CO may be defined as well as pre-edit, post-edit and default value methods,
TO: this is the object where basic row operations may be defined. Key definition, duplicate check, automatic sequence generation, pre-delete and post-delete methods, data flows, referential Integrity rules,
VO: Filtering rules, computed columns, additional referential integrity rules,
RO: Computed columns, aggregate functions, referential Integrity rules may be defined.
MO : Pre and post actions of reports may be defined.

3. 1. Demons

For each object a pre and post methods may be defined. For AO and CO, these methods are simple pre and post edit procedures. Pre object method is executed before the object is active, and the post method is initiated automatically upon leaving the object. These demons are given as preACT and postACT procedures in method section of objects.

3. 2. Standard Methods

The system comes up with a pre-defined method and these methods are utilized by TOs. Some of these methods are the built-in methods of Clipper Classes and the rest is built by the programmer. It should be assumed that every TO do inherit methods from block CORE_METHOD. This block is part of initial source template. Even it's dropped a copy of these methods that is embedded in system code will be in use. However it's possible to overwrite or use them in different ways.

```
OBJECT CORE_METHOD
     KEY F9 INVOKES SearchByIndex TITLE "Arama"
     KEY alt-R INVOKES ReportGenerate TITLE "Raporlama"
     DELETE RESTRICTED IF CHILD EXISTS
     ----
ENDOBJECT
```

There are some extra pre-defined methods. Methods may be used to define some behavioral properties of objects.

KEY	PREACT isempty(...) POSTACT Dupcheck(...)
AUTOSEQ	INITVALUE AutoSEQ(...)
NODUP	POSTACT DupCheck(...)
NONULL	POSACT not isempty(...)
FULL	POSACT not isempty(atrim(..))
NOEDIT	PREACT false
MULTIVALUE <...>	PREACT Select(<...>) POSTACT isWithin (<...>)

3. 3. Standard pre-edit, pre-delete and post-delete methods

While editing a foreign key, in most cases the value should be in the set of parent tables' key. This is the

standard existence check act as a pre-edits method. In addition as a standard, a pop-list of available values is listed to select among them.

While attempting to delete row that is parent, related with its child tuples there may be different standard strategies (Table 3). However these methods may be overwritten.

Table 3. Delete strategies and related commands.

COMMAND / Strategy
DELETE RESTRICTED
Do not delete, which is rarely applied.
DELETE RESTRICTED IF CHILD EXISTS
Do not allow delete if child exists
DELETE INVOKES DELETECHILDREN
Delete all children (complete subtree) Delete Propagation
DELETE INVOKES NULLIFY
Nullify the foreign key in the children objects.

3. 4. Method Inheritance and Overwrite

If it is not defined again, the methods inherit from ancestors. Even it is not meaningful it is possible to relate same method to two or more keys (buttons). However if at lower level a different method were assigned to same buttons, the latter would be dominating. In VO, collision of methods due to multiple inheritance is possible. In case of a collision, the selection would be arbitrarily, if it were not redefined. However the preACT and postACT methods will work additive; accumulating the methods that should be applied.

4. APPLICATION

DICTPRO is dictionary-based expert system that aids designer to input the basic specifications. Depending on these specifications a module called EXEC_HOT simulates many properties of the program that will be coded (in progress). After some refinements, another module of the system may generate the source code that could be modified by programmer. The source should be compiled and linked with other modules twice.

- First, to evaluate the schema and create physical files and access mechanisms.
- Second, to arrive at the executable module.
  Link list should include kernel of the database, other objects and user-defined functions.

4. 1. Architecture

The System includes many facilities to implement the model. The list of main modules is given below.

- Class dictionary
- Object dictionary and definition
- Source code generation
- Editing Source code
- Schema generation. Physical file creation and modification. Access mechanism definition.
- Edit data
- Maintain access mechanisms

4. 2. Authorization, Recovery, Multi-user Environment

Given the TO and VO the system will add the built-in authorization module. Using this module administrator will be able to arrange the grants, and then the system will automatically apply the restrictions. For each user, his rights for each TO, VO and MO and also the buttons may be defined. The system allows creating journal files of the selected tables that keep track of every operation. When a file is destroyed, it is possible to recover the modifications starting from the last back-up. In multi-user environment, tables that face a storage operation (insert, update, and delete) are locked automatically.

4. 3. Implementation

The system is constructed by CA-Clipper DBMS. Its built-in functions, its native classes and commands translation directives and CA-Clipper Tools are utilized. Below is a typical command translation application.

```
#command DISPLAY <li1>[,<lin>][<p:LASTPAGE>] ;
          FROM  <fi>            ;
          [CONNECTION <cn>]   [FILTER <tfi>]    ;
          [AT <c1>[,<c2>[,<c3>]]]  [<b:BOX>] ;
          [HEADING <he> ]        ;
   ⇨   DISP_SUBSET({||QQout( <li1>[,<lin>] )},
       <"fi">,<"cn">,<"tfi"><.p.>,<.b.>,<c1>,<c2>,<c3>,<he
       >)
```

5. CONCLUSION

Numerous object-oriented systems have been designed and implemented over the past few years. Current research objective was to create a system that would be easily used for developing immediate applications. This objective was fulfilled in a great percentage. zPROT, an un-matured version without objects and command language, has been used in many applications and it had proved its efficiency. Still it needs many enhancements and extensions. A matrix type of report generation, a rule-base method definition and a graphical modeling interface extension is feasible and may improve the overall prototyping activity (Başoğlu, 1993, 1996). Internal algorithms may need revisions to optimize the speed.

6. ACKNOWLEDGMENT

Special thanks to Bekir Kara for his innovative approach and cooperative work while developing the zPROT system engine and testing on various applications.

7. REFERENCES

Ahad, R., Yao, B. 1993. RQL : A Recursive Query Language, IEEE Transactions on Knowledge and Data Engineering, 5 (3), 451-461.

Başoğlu, N. 1993. "İstatistik Veri Tabanları için Bir Sorgulama Sistemi", **Araştırma Sempozyumu**, DIE, Ankara, Kasım, 1993.

Başoğlu, N. 1996. "Bir Kural Tabanlı Nesne Yönelimli Üretim Çizelgeleme Sistemi Tasarımı", **1. Zeki İmalat Sistemleri Sempozyumu, Sakarya Üniversitesi**, Sapanca, Mayıs, 1996.

Bekke, J. H. 1992. Semantic Data Modeling, Trowbridge,Prentice Hall.

Berztiss, A. 1993. The Query Language Vizla, IEEE Transactions on Knowledge and Data Engineering, 5 (5), 813-825.

Dar, S., Agrawal, R. 1993. Extending SQL with Generalized Transitive Closure, IEEE Transactions on Knowledge and Data Engineering, 5 (5), 799-812.

Date, C.J. 1990. An Introduction to Database Systems, 5th ed., Addison-Wesley, Reading, MA.

Dewitz, S. D. 1996. System Analysis and Design And the Transition to Objects, McGraw-Hill.

Gray, P. M. D., Kulkarni, K. G., Paton, N. W. 1992. Object-Oriented Databases A Semantic Data Model Approach, Trowbridge, Prentice Hall.

Gyssens, M., Parenaens, J., Bussche, J. V., Gucht, D. V. 1994. A Graph-Oriented Object Database Model, IEEE Transactions on Knowledge and Data Engineering. (6), 572.

Hughes, J. 1991. Object Oriented Databases, Cambridge, Prentice Hall.

Lee, B. S., Wiederhold, G. 1994. Outer Joins and Filters for Instantiating Object from Relational Databases Through Views, IEEE Transactions on Knowledge and Data Engineering. (6), 108.

Lieberherr, K., Xio, C. 1993. Formal Foundations for Object-Oriented Data Modeling, IEEE Transactions on Knowledge and Data Engineering, 5 (3), 462-478.

Mohan, L., Kashyap, R.L. 1993. A Visual Query Language For Graphical Interaction With Schema-Intensive Databases, IEEE Transactions on Knowledge and Data Engineering, 5 (5), 843-858.

Parsaye, K., Chignell, M., Khoshafian, S., Wong, H. 1989. Inteligent Databases, Objet-Oriented, Deductive, Hypermedia Technologies, Wiley.

Snodgrass, R.T., Gomez, S., Edwin M. L. 1993. Aggregates in the Temporary Query Language Tquel, IEEE Transactions on Knowledge and Data Engineering, 5 (5), 826-842.

Spaccapietra, S., Parent, C. 1994. View Integration : A Step Forward in Solving Structural Conflicts, IEEE Transactions on Knowledge and Data Engineering. (6), 258.

Su, S.Y.W., Lam, H. 1993. Association Algebra: A Methematical Foundation for Object-Oriented Databases, IEEE Transactions on Knowledge and Data Engineering, 5 (5), pp.775-798.

Ullman, J. D. 1988. Principles of Database and Knowledge-Base Systems, Compuer Science Press.

Yoo, S. B., Sheu, P. C. Y. 1993. Evaluation and Optimization of Query Programs in an Object-Oriented and Symbolic Information System, IEEE Transactions on Knowledge and Data Engineering, 5 (3), 479-495.

Yourdan, E. 1994. Object-Oriented Systems Design: An Integrated Approach,Yourdan Press/Prentice-Hall. 69.