A COMPARISON BETWEEN
OBJECT-ORIENTED DATABASE SYSTEMS AND INFORMATION SYSTEM SHELLS

ABSTRACT

The work makes an overview of the main characteristics of object-oriented database systems (OODBS) and information system (IS) shells. It shows the clear distinctions between the two types of systems and the practical application of each of them. As well the paper examines ten commercial object-oriented database management systems (OODBMS) and one IS shell, which has both unique characteristics, related to its limited application and characteristics common with OODBMS.

Keywords

Object-oriented databases, information systems, CASE tools

1. INTRODUCTION

The main purpose of database systems (DBS) is to handle the informational side of an application and not to deal with its functional side. On the other side OODBS deal with both sides of an application and have emerged as a universal tool for the creation of application requiring database (DB) support. Contrasting OODBS, IS shells are specialized tools with limited application only to a specific class of practical applications. They compensate their limited application with their user-friendly interface and their unique capabilities to deal with the specific area of tasks for witch they are designed. For example the system Visual Object-Oriented Shell for Information Systems (VOOSIS) [8], witch will be used as an representative of IS shells trough out this paper, has the following characteristics not common for OODBS:

1. It has visual view for browsing the object and class hierarchy.
2. Supports quantitative objects, i.e. objects that have quantitative property.
3. Allows the merging of two quantitative objects into one and the split of one quantitative object into two objects.
4. Supports an extension of SQL in order to meet the unique requirements of the class of IS that can be created with the system. In particular there are primitives for finding the objects that belong in a specified object or are part of a specified class. Since the only relationship between objects that the system supports is “physical containment”, constructions for specifying complex relationships between objects like “joins” are absent from the extended SQL supported by the system.
5. Handles strict authority of the users of the system. More precisely, the system requires the users of the system to be part of one or more groups. VOOSIS allows for every class of object and group the explicit definition of the rights of the users belonging to that group. In particular they may have rights to create, move, delete, examine or place objects in an object from the specified type.
6. Supports event handling. The events in the system can be system, class or object. An example of a system even is the login of a new user in the system. An example of a class event is the change of the global characteristics of an object. An example of an object event is the creation of a new object.

The goal of this work is to do a quick overview of the main characteristics of object-oriented databases (OODB), object oriented database management systems (OODBMS), information systems (IS) and *VOOSIS*, as its representative, and outline the points in witch they differ and agree. The overview of contemporary OODBS and IS will be done in part 2. In part 3 a short examination of *VOOSIS* will be made and then its characteristics will be compared with those of OODBS. Part 4 of the paper summarized the reached in the work results.

OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS AND INFORMATION SYSTEMS

The object-oriented databases are an extension of the traditional relational databases. Although the created so far standards for OODB continue to evolve, some basic elements have emerged. Those specific characteristics according to [6] are:

1. Each object has a unique identifier.
2. The objects have attributes and methods, which operate on the values of the attributes.

3. The attributes have a domain: simple type, composite type or an object.
4. The values of the attributes of an object are either directly accessible in the system or an indirect access to them is supported by the methods of the objects.
5. The objects that have common methods and attributes could be grouped into classes.
6. Each object can belong to at most one class
7. It is possible for the class itself to be looked at as an object of a generalized class or meta-class.
8. The classes are connected in a hierarchy in which the relation is inheritance. The classes participating in a given relation are called respectively super-class and sub-class.
9. The class hierarchy is a lettuce; i.e. it is possible for a class to inherit more than one classes.

To the basic model of an OODB the following supplementary characteristics can be added [6]:

1. Containing objects.
2. Versification of objects.

The so stated conception of an OODB accents on the characteristics that make it different from a relational DB. On the other hand the function that an OODBMS system has to support are extension of the functions of a relational DBMS. According to [4] the basic characteristics of database (DB) regardless of whether it is relational or object-oriented are:

1. Support of multiple user access to the system.
2. Restricted access to the system by the different type of users.
3. A possibility for partitioning the database in several parts.
4. A possibility of entering, querying and showing data from the DB.
5. A possibility for indexing the data in the DB.
6. Support of transactions.
7. Concurrency control.
8. Crash recovery.

OODBMS use the knowledge acquired regarding relational DBMS to implement the upper characteristics and if needed expand and change the methodology used in designing relational DBMS.

After we have outlined some of the elements of OODB and OODBMS, we will examine closer the theory behind OODB and OODBMS. In this part of the paper we will also take a quick glance at IS.

1.1. Objects in Object-Oriented Databases

The objects are the building element in an OODB and are capable to the tuples in the relational DB model. The main differences between the two are that the objects have a more complex structure and that each object has a unique for the whole database identifier.

The structure of an object is made of attributes and methods. The attributes could be simple, reference, composite or derived. The ***simple attributes*** are of standard type such as: integer, real, data, time, Boolean, currency, etc. The ***reference attributes*** have a domain of an object and are used to represent relationships between objects. The ***composite attributes*** have a domain of an array, list or a set of values. The ***derived attributes*** are attributes, which value is a function of other attributes. Most commonly those are read-only attributes. When a reading operation is performed on them the associated with them function is executed and the calculated value is returned.

After this quick review of the structure of objects in OODB we will focus on the type of relationships between objects and the way those relationships can be represented in an OODB.

1.2. Relationships between Objects in an Object-Oriented Database

In [2] two types of binary relationships between objects, in a general object oriented model, are stated: association and aggregation. The relationship ***association*** corresponds to the interrelation between two object (in particular the two objects could symmetrically exchange messages or one could be the client and the other the server). The ***aggregation*** relationship represents a connection between two objects where one is an attribute of the other. Since the first type of relationship has to do with the functionality of the object-oriented model and the second with its information structure we will examine closely only the second type of relationship and its application to OODB.

Since the entity-relationship (ER) model is a conceptual model, there should be a way to map an ER diagram into OODB. An entity in the ER model corresponds to a type of object (class). A ***one to one***, a ***one to many*** and ***many to many*** relationships between entities could be mapped in an OODB using aggregate attributes. To represent a "one to one" relationship between two entities, each object of each entity should contain a reference attribute corresponding to the object with which it is connected. A "one to many" relationship between two entities could be represented by adding composite reverence attributes to all objects on the "one" side, and reference attributes to the objects on the "many" side. Correspondingly a "many to many" relationship is represented by adding composite reference attributes to the objects on both sides of the relationship. Some OODBMS systems introduce the concept of inverse attributes when dealing with relationships between objects. For given object and a binary relationship between this object and a second object the inverse attribute for the first object is the reference attribute of the second object, which defines this relationship for the second object. After we have examined how a relationship between objects can be expressed in an OODB we will look for some characteristics of those relationships.

From the fact that the attributes of a given object could be derived, it follows that the relationships between objects could be derived as well, i.e. there could be functions which could dynamically calculate what relationships between objects hold at the present moment.

It is also worth noticing that some OODBMS look at the relationship of containment between two objects as a special kind of the relationship aggregation. According to [3] the reasons for designating this relationship are (1) the system could adequately respond on actions with such objects; (2) allows for the effective storage of the objects on the physical carrier.

Now that we have finished our overview of objects in OODB we will extend the abstraction used in OODB by going one step further and introducing the concept of a class.

1.3. Classes in Object-Oriented Database Systems

The ***classes*** in the object-oriented terminology are defined as a set of objects with common structure and behavior. In OODB a class is defined as the object's type. The classes in an OODB are similar with the relations in the relational database model with the following differences: (1) classes can have their own unique characteristics like class methods and functions, relations can not; (2) classes can contain methods, relations can not. Now, that we have defined what a class in the OODB terminology is let us look at the different classification of classes in a general object-oriented database model.

One division of classes is into base and group classes. While the ***base classes*** are the templates for creating objects, ***group classes*** are used to group classes with similar characteristics. This division is not strict because it is possible for one class to be both base and group. Another more rigid division of classes is into physical and virtual. ***Physical classes*** are those from which objects can be created and virtual classes are those that can be used only for grouping classes, but not for object creation.

After we have cited two classifications of classes let us examine the types of relationships between them. According to [2] these relationships are association, inheritance, aggregation, use, parameterization and inheritance. The most important of those relationships for OODB are aggregation and inheritance. Since aggregation was described in the previous sub-pint we will look at inheritance here.

The ***inheritance*** between classes in an OODB allows for a class to inherit the attributes and methods of its super-class. Some OODBMS allow for specification of which characteristics of the super-class to be inherited, and which not to be. Based on the relationship inheritance a graph between the classes could be formed. This graph could be a tree, a lettuce, a set of trees or a set of lettuces depending on whether multiple inheritance and multiple roots of the hierarchy are supported. The definition of a class contains three sections: the public, private and protected. The property, that not all methods and attributes of a class are directly accessible is called ***encapsulation***. While the ***private*** part of the declaration is accessible only by the methods of the class, the ***protected*** part is also accessible by the methods of the classes that inherit the given class. There are no restrictions on the access of the ***public*** elements in the definition. A deviation from the object-oriented theory is needed when applying the encapsulation principles to OODB because queries in them must have direct access of all attributes of a given object including its private and protected attributes. While some OODBMS allow for the encapsulation to be broken when using queries, other deal with the problem by allowing for indirect access to the attributes of an OODB in a query, by using derived attributes or by allowing the direct call of methods.

Except private, public and protected, the attributes in a class can be categorized as local and global. While the ***local attributes*** have a separate value for each object of a given class, the ***global attributes*** are unique for the class (all objects of the class share the same value of the global attributes). The same division applies for the methods of the classes, which are divided into global and local as well.

This ends the discussion of the characteristics of OODB. In the next sub-points some of the characteristics of OODBMS will be examined.

2.4 Queries and Programming Languages in Object-Oriented Database Management Systems

Just as in relational DBMS, in OODBMS there are a variety of data query languages and programming tools. An attempt for classification will be made in this sub-point.

Most generally there are two approaches in designing a programming tool for a DBMS. The first approach relies on the creation of a limited database manipulation language, which is to be part of a high-level language. When using the second approach a full-functional language to work with OODB is created, which should support data manipulation tools, purely programming constructions and also primitives that allow access to the operating system.

Another classification of OODBMS is based on the result that the queries in them return. Unlike relational DBMS queries, in which the result of a query is a relation, a query in an OODBMS can return as a result: a relation, a set of objects or something of user-defined type. More detailed explanation of the type of OODBMS, regarding the type of data returned by the queries in them, can be found in [3]. Other classifications of OODBMS can be based on the type of the query language of the system (procedural or functional) or on whether query optimization is made ([5]).

In the next sub-point a study of the transactions and messages in an OODBMS will be maid.

2.5 Transactions and Messages in Object-Oriented Database Management Systems

Most OODBMS extend the existing theory about transactions from relational DBMS, by adding new theory, such as object versioning and containing transactions. These new elements allow for long transactions, which could last for more than one day, to be executed. In long transactions concurrency control is generally handled by sing object versioning. ***Object versioning*** means that an object can have more than one version, and the different transactions work with their own versions of the objects. In this way there is no competition for resources and the transactions are not interrupted till their end. The compatibility between the transactions is handled by unifying the versions of the objects, which is done after the transactions have ended. Another method used for handling long transactions proposes the containment of one transaction into another. A ***composite transaction*** is a transaction that contains sub-transactions in it. The advantage of using a composite transaction is that when a resource conflict occurs, the execution of the transaction can continue, and only the data changed in the sub-transaction could be marked as "contaminated".

After we have looked at the main characteristics of OODB and OODBMS we will make a comparison between ten OODBMS based on some of the characteristics we have discussed.

2.6 Comparison between some Popular Object-Oriented Database Management Systems

The systems that will be compared are: ORION, O_2, ONTOS, ObjectStore, GemStone, ITASCA, Objective/DB, VERSANT, POET and the described in the next point system VOOSIS. The comparison between the systems is based on the answers of the following questions:

1. Does the system support an integrated programming environment?
2. What high-level language is close to the language used in the system?
3. Is the system multiple-user?
4. If yes, how is the access to the system designed (i.e. client/server, terminal emulation)
5. Does the system support data encapsulation?
6. Is this encapsulation broken when queries are used?
7. Is class inheritance supported?
8. Is a unique object identifier, accessible to the users of the system, supported?
9. Are reference attributes supported?
10. Are derived attributes supported?
11. Are composed attributes supported?
12. What popular data query language is similar to the query language used in the system?

13. Of what type is the query result in the system?
14. Are the objects in the system indexed?
15. Does the system support long transactions?
16. What concurrency control mechanisms are used in the system?
17. Does the system support composite transactions?
18. Is event handling implemented in the system?
19. Is object versioning supported?
20. Does the system support composite objects?
21. Does the system limit the access of the different users to its OODB?
22. Does the system present visual interface?
23. Does the system allow for the dynamic change of the class structure?
24. Does the system support query?

The table comparing the OODBMS is shown below. The information for filling the table is mainly from [6], [3] and [1]. The purpose of the comparison is not to emphasize the advantages of one OODBMS over another, but rather give a general idea, of the features supported by contemporary OODBMS. The author of this publication carries no responsibility for the correctness of the table.

QUESTION/ OODBMS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
ONTOS	no	C++	yes	client/server	yes	yes	yes	yes	yes	no	yes	SQL	table	no	no	object locking	no	no	yes	no	yes	no	no	yes
O2	yes	C++ and Lisp	yes	client/server	yes	yes	yes	yes	yes	yes	yes	SQL algebra	user defined	yes	yes	by object versioning	yes	yes	yes	yes	yes	yes	yes	yes
ORION	yes	Lisp	yes	client/server	no	yes	yes	yes	yes	yes	yes	SQL	object array	yes	yes	object locking	no	yes	yes	yes	yes	no	yes	yes
ObjectStore	no	C++	yes	client/server	yes	no	yes	yes	yes	no	yes	C++	object array	yes	no	by object versioning	no	yes	yes	no	no	no	no	no
GemStone	no	Pascal,C,C++ and SmallTalk	yes	client/server	yes	no	yes	yes	yes	no	yes	own language	object array	yes	no	optimistical and pessimistical methods	no	yes	no	no	yes	yes	yes	no
ITASCA	no	C++,C, CLOS,Lisp and Ada	yes	client/server	yes	yes	yes	yes	yes	no	yes	own language	table	yes	no	two phase protocol	no	no	yes	yes	yes	no	yes	no
Objective/DB	yes	C++	yes	client/server	yes	no	yes	no	yes	no	yes	C++ and SQL	table	yes	no	object locking	no	no	no	yes	yes	yes	no	yes
VERSANT	yes	C++ and Samlltalk	yes	client/server	yes	yes	yes	yes	yes	no	yes	SQL	table	no	no	two phase protocl	no	no	no	no	yes	yes	yes	yes
POET	yes	C++	yes	client/server	yes	no	yes	yes	yes	no	yes	own language	object array	no	no	object locking	yes	yes	no	no	no	no	no	no
IRIS	yes	C and Lisp	yes	client/server	no	N/A	no	yes	yes	no	yes	SQL	table	no	no	HP-SQL	no	no	yes	no	no	no	yes	yes
VOOSIS	no	user-defined	yes	termianal emmulation	no	N/A	yes	yes	no	yes	no	SQL	table	no	no	N/A	no	yes	no	yes	yes	yes	yes	yes

The reason the system VOOSIS is added to this comparison is to show its similarities with OODBMS. In the next sub-point we will do a quick review of IS.

2.7. Basic Characteristics of Information Systems

A computer IS according to [7] is made of hardware, software, DB, telecommunications, people and procedures. A IS shell is a tool for building the software and DB part of an IS. One classification of IS is into: transaction management systems, management information systems, decision support systems and systems based on artificial intelligence ([7]).

The difference between an IS shell and an OODBMS is that while the shell is a tool for creation of a particular type of IS, OODBMS are universal tool for satisfying information needs. The IS created from the described in the next point system VOOSIS can be classified as management information systems. If the created by VOOSIS IS contains methods for decision support, the system can be classified as decision support system.

In this point we have examined some characteristics of OODB, OODBMS and IS. In the next we will take a close look at the system VOOSIS and compare it with the type of systems examined so far.

2. COMPARISON BETWEEN VISUAL OBJECT-ORIENTED SHELL OF AN INFORMATION SYSTEM (VOOSIS) AND OODBS

The system VOOSIS is a tool for creation and use of IS. Although the system has some characteristics in common with OODBMS it remains an IS shell, because of its limited application. The system compensates this limited application with the unique functionality. For example it supports visual view of the objects and

classes of an IS, possibility for merging of two objects into one and the split of one objet into two, numerical measurement of the objects, etc. Those are all features not characteristic for OODBMS. It is also true that the system does not support important for OODBMS features such as reference and composite attributes. In this point we will look at the main characteristics of VOOSIS and will compare them with those of OODBMS where appropriate.

2.1. Types of Users in VOOSIS

The system supports four types of user administrators, designers, operators and analyzers. The *administrators* of the system have responsibilities similar of those of DB administrator. They administer the user of the system and distribute them among the different groups in the system. The *designers* of the system build the skeleton of the system (i.e. the class hierarchy and the static objects in the object hierarchy and the functions associated with them). The *operators* of the system are the people who map the changes the monitored real-world process to the system and the *analyzers* are those, who monitor the state of the system and build appropriate reports when needed. After we have looked at the users of VOSIS in the next sub-point the specific characteristics of the design of the system will be examined.

2.2. Structure of the Objects in VOOSIS

The objects in the system model real-world objects. The designers and operators of the system create the objects. Each object has a type – the class from which it is created. The objects are connected in an object hierarchy, corresponding to the physical containment of the modeled real-world objects. Except information about the object hierarchy the system can contain information about the relative position of each object to the object that contains it. This gives the system characteristics close to that of a Geographical Information System.

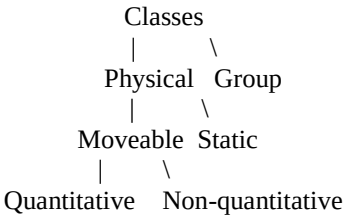
The characteristics of an object are the class to which it belongs, the coordinates of the object relative to the object that contains it and the values of the object's class local attributes. If a given local attribute value is not entered during an object's definition the default value for that attribute entered during the class's definition is used.

Similar to most OODBS the objects in VOOSIS do not have a name, but a unique for the IS identifier. The reason for this decision is that the objects themselves do not have their own identity, except for the class to which they belong and their place in the object hierarchy. If for some reason the different objects from a class should have names, this could be accomplished by adding a textual attribute "name" to that class. VOOSIS allows for the definition of key attributes but does not index on them because the system does not support the need complex storage mechanisms to do so.

The users of VOOSIS, and more particularly the operators of an IS supported by VOOSIS could be part of the object hierarchy of that IS. This makes sense because the operators in most cases are part of the modeled space and like the rest of the object in it, execute actions and actions are executed on them. The system lives the decision on whether the users of the system are displayed as objects in the object hierarchy of a particular IS to the designers of that IS. On the other hand in OODBS the users are usually not part of the DB of the system.

2.3. Class Structure in VOOSIS

The classes in the system are the templates from which objects are created. Two objects, from the same class, share the same attributes and methods. Following the object-oriented principles the system supports class inheritance, but does not support multiple inheritance for reasons of simplicity. The following tree shows a classification of the classes supported by VOOSIS.



The ***physical classes*** are the leaves of the class hierarchy tree and from them objects can be created. The ***group classes*** are used to combine classes with similar structure and behavior; as well objects can not be created directly from them. The group classes have methods and event handlers, but does not have any attributes.

The physical classes in VOOSIS are divided into moveable and static. The ***moveable classes*** are those from which objects that can be moved are created and the ***static classes*** are those from which objects that cannot be moved are created.

The moveable classes in the system can be quantitative and not quantitative. The static classes in the system are no quantitative. The ***quantitative classes*** are those, whose objects have a system attribute quantity, and non-quantitative ***classes*** are those, whose objects do not. Several objects from a quantitative class can merge into one object that will have value for quantity the sum of the quantities of those objects. As well an object belonging to a quantitative class can split into several objects from the same class and the original's object quantity will be distributed among the created objects. The system supports the existence of objects with value for the system attribute quantity equal to zero, but does not permit such objects to be divided. All those types and characteristics of classes are unusual for OODBS.

So far in this point we have looked at the users, objects and classes of an IS created with VOOSIS. The use of methods in IS created with VOOSIS will be discussed in the next sub-point.

2.4. Methods in VOOSIS

The methods in the system can be classified as global, class and object methods. The ***global methods*** are those, that are not connected with a particular class; the ***class methods*** are those connected with a particular class, without using any of its local attributes; the ***object methods*** are those that can be executed on a particular object and could use the local attributes of that object. Similar class typing exists in some OODBS as well. The choice on whether the methods should be written in the host environment of the system or be written in a an arbitrary language and imported into the system (via dynamic link libraries for examples) is independent of the type of the system. *VOOSIS* adopts the second approach.

2.5. Actions, Events and Messages in VOOSIS

Each change in the state of the object hierarchy is the result of the execution of an action. Example of actions are: the creation of a new object and its placement in the object hierarchy, the deletion of an existing object, the movement of an object, the split of an object in several parts or the merge of several objects into one, etc. Actions in the system are either triggered by user interaction with the system, or are triggered by methods in the system.

Events in IS created with VOOSIS, that can be handles by calling methods are of three types: local, class and global. The ***local events*** are triggered when something in a particular object is changed, the ***class events*** -when something in a particular class in changed and the ***global events*** - when something in a particular IS in VOOSIS is changed. As a result of the change of the local attributes of a given object - local events occur; as a result of the change of the global attributes of a given class - class events occur. The creation, deletion, merging and split of objects are local events for the objects on which the action is performed. The global events in VOOSIS can be classified as time and system. ***Time events*** could be the coming of a specified moment or the elapse of specified time. ***Global events*** could be the beginning of user session or the end of such session with a particular IS in VOOSIS. Events could be handled by methods if such are specified. The local events are handled by object methods, the class events - by class methods and the global events - by global methods.

For a method to be called in VOOSIS a message to the class to which the method is connected is send. If the method happens to be global, the message is send to the system class called ***SYSTEM*** which is the root of the class hierarchy in the IS supported by VOOSIS. If the message send to a given class in not supported by the class, it is sent to its super class. If non of the super-classes support the method the message eventually reaches the class *SYSTEM* and the default handler for a non existing method in that class is called. For message handling purposes the operators of an IS are also considered to be objects, i.e. they can send messages to objects and objects can send messages to them.

In most OODBMS actions that are performed are written in a log file, but are not handled separately. Events are handled usually, because event handling is part of the general object-oriented model. They are handled usually using messaging passing between objects. This method has the advantage that it deals not only with event handling, but with polymorphism as well.

In this sub-point we examined the events, actions and messages in VOOSIS. It remains to look at the most important element of VOOSIS – its data query language.

2.6. Data Query Language in VOOSIS

The queries of an IS in VOOSIS are written in the extended custom SQL of the system called VOOSIS SQL. The system supports two types of queries: on the current state of an IS and on its past states. This corresponds to the databases supported by VOOSIS, which are made of four parts for every IS. These parts are: (1) a section that contains the present view of the class and object hierarchy; (2) a section that contains the log of all events that have been carried on an IS from its creation to present; (3) section that contains snap-shots done at different times in the past of section 1; (4) section containing information about the administration of the particular IS (i.e. information about the users, groups and methods of the IS). At intervals specified by the designers of the IS its database is initialized (it makes sense this period to be one business day or some whole fraction of it). When an initialization occurs the information in the first part and fourth part of the DB of the IS remain unchanged, the information in the second part is erased, and a snapshot of the first part of the DB is added to the third part of the DB.

When doing queries on the present state of an IS the extended SQL supported by the system called ***VOOSIS SQL*** is used. Since VOOSIS supports only the relationship "physical containment" between objects, the SQL of the system is designed to handle only this relationship. The ways *VOOSIS SQL* works, is that it introduces its own functions that return tables. The SQL of the system is maid of ANSI SQL enriched with some functions. An exact description of the SQL of the system can be found in [8]. In the conclusion of the paper we will analyze the reached in the work goals and the possibilities for future work on the touched in the paper topics.

3. CONCLUSION

In this paper we have described some of the characteristics of OODB and OODBMS and have compared the theory on which they are based with that used in designing IS and *VOOSIS* in particular. The conclusions that can be made are that OODBMS are more functional than IS shells because they are designed to deal with greater array of problems. On the other hand IS shells are more suited for dealing with specific class of problems and all their functions are focused on dealing with them. The system *VOOSIS*, as an example of IS shell, is interesting in that, that it has taken some features of OODBMS applicable to the class of IS that can be created from it, but also introduces new concepts where needed.

REFERENCES

[1] Bancilhon, Francois, Claude Delobel and Paris Kanellakis (editors), Building an Object-Oriented Database System. The Story of O_2, Mortgan Kaufmann Publishers, ISBN 1-55860-169-4 (1992).
[2] Booch,Grady, Object-Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Company, Inc., ISBN 0-8053-5340-2 (1994).
[3] Cattell, R.G.G., Object Data Management, Revised Edition. Object-Oriented and Extended Relational Database Systems, Addison-Wesley Publishing Company, ISNB 0-201-54748-1 (1994).
[4] Elmasri Ramez and Shamkant B. Navathe, Fundamentals of Database Systems, The Benjamin/Cummings Publishing Company, ISBN 0-8053-17553-8 (1994).
[5] Freytag, Johann C, David Maier and Gottfried Vossen (editors), Query processing for advanced database systems, Morgan Kaufmann Publishers, ISBN 1-55860-271-2 (1994).
[6] Kim, Won, Introduction to Object Oriented Databases, The MIT Press, ISBN 0-262-11124-1 (1990).
[7] Stair, Ralph M., Principles of Informational Systems. A Managerial Approach, Boyd & Fraser publishing company, ISBN 0-87835-789-0 (1992).
[8] Stanchev, Lubomir. Visual Intenerated System For Object-Oriented Development And Exploitation Of A Special Class Informational Systems. The spring conference of the Union of the Mathematicians in Bulgaria (1999)