VISUAL INTEGREATED SYSTEM FOR OBJECT-ORIENTED DEVELOPMENT AND EXPLOITATION OF A SPECIAL CLASS INFORMATIONAL SYSTEMS

ABSTRACT

The work introduces a tool for rapid design and development of information system (IS). The name of the tool is Visual Object-Oriented Shell for Information Systems (VOOSIS). The class of IS supported by the tool is limited to systems monitoring the changing characteristics of the modeled real world objects. More precisely the system keeps track of an object hierarchy, representing the physical containment of the real world objects represented in the system and a class hierarchy, representing the inheritance between the object types. Although VOOSIS has common feature with the object-oriented database management systems (OODBMS), it presents many unique characteristics, related to the specific application of the system. A working hybrid horizontal-vertical prototype of the system has been developed. It consists of four parts*: VOOSIS ADMINISTRATOR*, *VOOSIS DESIGNER* and *VOOSIS OPERATOR* and *VOOSIS ANALYZER*. The prototype is developed using Borland Delphi 3.0 and runs under Windows 95 and Windows NT. If the prototype is extended, the system that emerges can be applied for handling the informational needs of process like monitoring the groceries in a supermarket, handling the organization in a restaurant, etc.

1. Introduction

Whit the increase use of information technologies trough out the world the need for CASE tools for fast and user friendly creation and exploitation of IS has become more evident. This paper proposes one such tool, which has limited application. The area of situations to which the proposed tool can be applied is narrowed to real-world situations where the movement of objects, related to the surrounding them objects is the process that requires computerized handling. Although the proposed tool has many elements in common with OODBMS it differs from them in some ways. The major characteristics of VOOSIS are:

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) and information systems (IS) and then propose a new tool for creating a specula class of IS. The overview of current OODB and OODBMS feature

will be done in part 2. In part 3 the main theoretical concepts behind the proposed tool will be outlined. Part 4 of the paper summarized the reached in the work results.

2. 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 [Kim90] 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 [Kim90]:

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 [ElSh94] 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.

2.1Objects 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.

2.2. Relationships between Objects in an Object-Oriented Database

In [Booc94] 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 [Catt94] 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.

2.3. 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 [Booc94] 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 [Catt94]. 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 ([FrMaGo94]).

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 optimization?

The table comparing the OODBMS is shown below. The information for filling the table is mainly from [Kim90], [Catt94] and [BaDeKa92]. 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 [Stai92] 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 ([Stai92]).

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

3.Visual Object-Oriented Shell of an Information System (VOOSIS)

The system VOOSIS is a tool for creation and use of IS. Although the system has some characteristics in common with OODBMS it remains n 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 basic 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 this of OODBMS where appropriate.

3.1. Types of Users in VOOSIS

The ***administrators*** of the system can add and delete users, assign and remove users from groups, as well as create, delete and rename groups. The users with similar responsibilities are designated into groups. Each group can be of one of three types: designers, operators and analyzers. A user could belong to more than one group. All operations done by the administrators are recorded in a special log part of the database of the particular IS. The administrators are given read rights to that log. They work with a specially created for them application ***VOOSIS ADMINISTRATOR*** which is part of the VOOSIS system.

The ***designers*** of the system build the schema of an IS. The responsibilities of the designers are: (1) to define the classes of the IS and connect them in a class hierarchy; (2) to create the static objects in an IS from those classes; (3) to determine together with the administrators which groups will have rights to create, move, delete and examine the attributes on an object from a given class; (4) to code and test the object, class and global methods of the system, as well as the methods that respond to local, class and global events. The designers perform those tasks using ***VOOIS DESIGNER***. This application provides them with visual view of the class and object hierarchy and allows for fast and easy execution of the above tasks. VOOSIS is an open system in the sense that it allows for the change of an IS schema during the exploitation of the IS.

The ***operators*** in VOOSIS are end-users of the IS created by the system. This are people that fill the created by the designers schemas of IS. It is the operator's responsibility to regularly update the changing real-world characteristics of importance into the particular IS. He does that by moving and deleting objects in the IS, or by changing their characteristics in order to notify the system for the changes that have occurred in the watched processes. Those tasks are performed using the application ***VOOSIS OPERATOR***, which provides a visual view of the object hierarchy, as well as tools for searching objects, by using their characteristics. The creation and movement of objects is performed using the "drag and drop" mechanism and the other functions on objects are made available trough "pop up" and "pull down" menus.

The ***analyzers*** in VOOSIS are end-users of the system. They are usually clerks or managers. Clerks could make standard reports for the changes in the watched process, such as monthly reports and balance sheets. The managers respectively could regularly examine the state of the process in order to have adequate and up-to-date information for managing purposes. The analyzers can perform does functions by directly browsing the object hierarchy, by using created by the designers methods for creating reports or by writing their own methods for examining the state of the system. The analyzers in VOOSIS work with ***VOOSIS ANALZER***.

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.

3.2. 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.

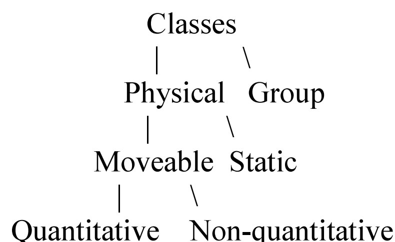
The characteristics of an object are the class to which it belongs, the coordinates of the object relative to the object that contains it and the values of the object's class local attributes. If a given local attribute value is not entered during an object's definition the default value for that attribute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

Similar to most OODMS 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.

3.3. 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.

For every class the designer of the IS could define the parameters of the class, its methods and attributes, as well as the way those methods should handle events related to the objects belonging to the class. The parameters of the class are its name, type and a picture to be used in visualizing objects from the class in the graphical view of the object hierarchy. When a class is defined the names of and the types of the global and local attributes are also defined. While for the local attributes their default value is entered, for the global attributes their initial value is specified. For every attribute restrictions on the attribute's domain can be stated. When a class is defined, the names of its local and global methods are entered, as well as the names of the dynamic link libraries (DLL) that implement those methods. The local and class events that are handled, together with the methods that do this handling, are also parts of the class definition. A detailed review of the vent handling mechanisms of the system could be found in point 3.5.

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.

3.4. 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.

The way in which the methods in the system are implemented and connected to an IS or a specific class is the following: (1) the methods are coded in a high-level programming language chosen by the designers; (2) the code is compiled to a DLL; (3) in ***VOOSIS DESIGNER*** the methods are attached to an IS or a particular class of an IS. The questions that could arise from the lack of own programming environment in VOOSIS are (1) how the written methods can have access to the database of a specific IS an the extended query language implemented in VOOSIS and (2) how the written methods can be tested. As a solution to the first problem three DLLs are made available. The libraries contain functions that respectively support access to the extended SQL of the system, allow object and class manipulation and allow the examination of the current or past states of the object and class hierarchy of an IS. The three function libraries are described in details in the file ***appendix1.doc***, which could be found at the stated at the first page of the paper web address. The problem with testing a written DLL could be solved by doing all the testing in the environment where the DLL is written. For this to be done special methods for calling the method and checking the method's results should be implemented in the programming environment of the DLL.

This concludes our look at the methods in VOOSIS. IN the next sub-point the actions, events and messages in the system will be examined.

3.5. 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 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.

3.6 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 those functions. For queries on the present state of the DB of an IS the functions that are used have the following signature:

function CLASS(result_attributes: **list of attribute names and types**, name_of_class: **string**): **table**;

function CLASSR(result_attributes: **list of attribute names and types**, name_of_class: **string**,
used_objects: **table**): **table**;
function OBJECTS(function_type: **enumerative**, result_attributes: **list of attribute names and types,**
used_objects: **table**): **table**.

The data types in the SQL are: number (marked with N), currency (marked with $), string (marked with A), Boolean (Marked with B), date (marked with D) and time (marked with T). The result of each function is a set of object represented in a table (each row in the table corresponds to an object), where the attributes of the objects that should be returned are given by the parameter *result_attributes*. Except for the attributes specified in that parameter the result tables contain also the parameters *ID* and *class*, to represent the unique identifier of the object and the class to which the object belongs.

The function *CLASS* is used to find some of the objects belonging directly of indirectly to the class with name: *name_of_class*. If that class is a physical class, some of its objects are returned in the result table. If the class is a group class, then the objects belonging to its physical sub-classes (direct or indirect are returned). The way to filter out which objects belonging to the class to be returned is by using the second parameter in the function – *result_attributes*. If the attributes listed in that parameter are part of the definition of the object's class and have the types listed in the parameter, then the object is included in the result set. Let us look at an example to clarify things: "The group class fruits have the physical subclasses "cherries", which has one attribute named quantity of type integer". In this example the function CLASS({quantity(A)},fruits) will return a table in which there will be no objects from the class "cherries", because that class does not have an attribute with name quantity and type string. The reason *VOOSIS SQL* is type sensitive is to distinguish between objects having attributes with the same name, but different types.

The function ***CLASSR*** is similar to the function *CLASS*, but it has one parameter – *used_objects*. This parameter is a table, which has an attribute named *ID* in it. This table corresponds to the set of objects, from witch the result set of objects will be chosen with the constraints coming from the *result_attributes* and *name_of_class* attributes.

The function ***OBJECTS*** is similar to the function *CLASSR*, but is does not select the objects belonging to a given class, but applies the function *function_type* on the objects specified by the parameter *used_objects* to get the result set. The parameter *function_type* can accept the following values: {LEAVES, ALL, THIS, IN}. If the value of the parameter is *LEVEVES*, then for each object from *used_objects*, all leaves of the object are found in the object hierarchy and the *result_attributes* parameter is used to filter the end result of the query. If the value of the function_type is ALL, IN or THIS, then respectively all sub-objects including the objects themselves, only the sub-objects, or only the objects themselves from the *used_objects* parameter are used in calculating the result set of objects (a sub-object of a super-object in the object hierarchy is an object that is part of the tree with root the super-object). Let us look at this example *VOOSIS SQL* query:

```
SELECT sum(price)
FROM CLASSR({price[$]}, fruits,
      (OBJECTS(LEAVES, {price[$]},
          (SELECT ID
           FROM CLASS({owner[A]},basket) as T3
```

```
        WHERE T3.owner = 'Ivanka')
    ) AS T2
) AS T1);
```

The query gives reposes to the question: "What is the total value of the sold by the cashier with login Ivanka fruits".
The query gives the required result by using compound queries. The innermost query selects Ivanka's basket, where all sales made by her are stored in the model. The next outer query selects all objects from her basket that have the attribute price. The next outer query selects from those objects only the fruits and the most outer query sums the price of the fruits.
Let us now look at the ways the object of the class on which the method calling a query is executed can be used as a parameter in a query. This is done by using the primitives ***$CLASS*** and ***$OBJECT*** which return respectively the class or the object on which the query is executed. This ends our review of the queries on the present state of an IS. Let us now look examine how queries on past states of an IS can be retrieved.
The functions on past states of an IS, that can be used in VOOSIS SQL, are similar to those of the present state, but are changed to allow for entering the time factor to be considered. The functions are:

function TCLASS(result_attributes: **list of attribute names and types**, name_of_class: **string**, from: **date**, to: **date**): **table**;
function TCLASSR(result_attributes: **list of attribute names and types**, name_of_class: **string**,
used_objects: **table**, from: **date**, to: **date**): **table**;
function TOBJECTS(function_type: **enumerative**, result_attributes: **list of attribute names and types,**
used_objects: **table**, from: **date**, to: **date**): **table**.

The way the functions work is that they find all snap-shots for the particular IS in the specified by the parameters from and to period. For each snapshot the function is executed, and to the result set an attribute with name data_of_snapshot is added. After this is done for all snapshots in the period, the result sets from each snapshot are united to give the end result. Let us look at the following queries:

```
(1) SELECT SUM (price)
   FROM TCLASS ({price[$]}, tomatoes, '1/6/98', '30/6/98');

(2) SELECT price, data_of_snapshot
        FROM T1
        WHERE price = (
          SELECT MAX (price)
   FROM TCLASS({price[$]}, tomatoes, '1/6/98', '30/6/98') AS T1 );
```

The first query gives answer to the question: "What is the income from tomatoes during June, 98 for some superstore?" and the second to the question: "At what day during that month the pick price for tomatoes was reached and what was that price?".

With those two examples we finished our overview of the system VOOSIS. 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.

4.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 VOOSIS. An overview of the main principles behind VOOSIS was presented. Because of the limited space for this paper the created prototype of VOOSIS was not described in it. However the source, the help files and the executable applications of the prototype can be found at the stated at the first page of the paper web address.

The areas for future work could be: (1) extending the prototype of VOOSIS and making it a commercial shell for IS and (2) extending the system to OODBMS by adding to it popular for such systems features as reference attributes, composite attributes, concurrency control mechanisms, etc. Both directions could lead to commercialization of the described in the paper system and to its transformation from a theoretical model to a commercial tool.

LITERATURE

[BaDeKa92] 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).

[Booc94] Booch,Grady, Object-Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Company, Inc., ISBN 0-8053-5340-2 (1994).

[Catt94] 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).

[ElSh94] Elmasri Ramez and Shamkant B. Navathe, Fundamentals of Database Systems, The Benjamin/Cummings Publishing Company, ISBN 0-8053-17553-8 (1994).

[FrMaGo94] Freytag, Johann C, David Maier and Gottfried Vossen (editors), Query processing for advanced database systems, Morgan Kaufmann Publishers, ISBN 1-55860-271-2 (1994).

[Kim90] Kim, Won, Introduction to Object Oriented Databases, The MIT Press, ISBN 0-262-11124-1 (1990).

[Stai92] Stair, Ralph M., Principles of Informational Systems. A Managerial Approach, Boyd & Fraser publishing company, ISBN 0-87835-789-0 (1992).