МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 1999
MATHEMATICS AND EDUCATION IN MATHEMATICS, 1999
Proceedings of Twenty Eighth Spring Conference of
the Union of Bulgarian Mathematicians
Montana, April 5–8, 1999

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

Lubomir Petrov Stanchev

The work introduces a tool for rapid design and development of information system (IS) [7]. 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 [3] 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. With 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 ([1], [4] and [6]), it differs from them in some ways. The major characteristics of VOOSIS are:

- It has visual view for browsing the object and class hierarchy.
- Supports quantitative objects, i.e. objects that have quantitative property.

*This paper is partially supported by the Ministry of Education, Science and Technology of Bulgaria under grant I-811/1998.

Some files associated with this paper can be found at `copern.bas.bg/lubomir/voosis.zip`

- Allows the merging of two quantitative objects into one and the split of one quantitative object into two objects.

- 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.

- 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.

- 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 propose a new tool for creating a special class of IS and describe its design. In part 2 the main theoretical concepts behind the proposed tool will be outlined. Part 3 of the paper summarized the reached in the work results.

2. 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 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 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.

2.1 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.

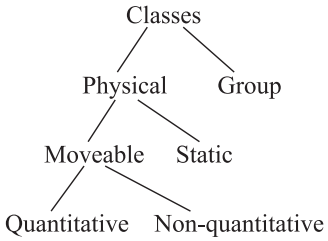
The characteristics of an object are the class to which it belongs, the coordinates of the object relative to the object that contains it and the values of the object's class local attributes. If a given local attribute value is not entered during an object's definition the default value for that attribute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

2.2. 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 ([2] and [5]) 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.

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.

2.3. 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. It remains to look at the most important element of VOOSIS – its data query language.

2.4 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 or 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 which 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 attri bute 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 snapshots 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.

This concludes 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.

3. Conclusion. In this paper 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.

REFERENCES

[1] Building an Object-Oriented Database System. (Eds. B. Francois, C. Delobel, P. Kanellakis) The Story of O2, Mortgan Kaufmann Publishers, 1992.

[2] . Booch. Object-Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Company, Inc., 1994.

[3] DELPHI USER'S GUIDE, BORLAND Publishing, 1997.

[4] R. G. G. Cattell. Object Data Management, Revised Edition. Object-Oriented and Extended Relational Database Systems, Addison-Wesley Publishing Company, 1994.

[5] M. Ellis, B. Stroustrup. The Annotated C++ Reference Manual, Addison-Wesley Publishing, 1990.

[6] W. Kim. Introduction to Object Oriented Databases, The MIT Press, 1990.

[7] R. M. Stair. Principles of Informational Systems. A Managerial Approach, Boyd & Fraser publishing company, 1992.

Central Laboratory for Parallel Processing
Bulgarian Academy of Science

ВИЗУАЛНА ИНТЕГРИРАНА СРЕДА (ШЕЛ) ЗА ОБЕКТНО ОРИЕНТИРАНО СЪЗДАВАНЕ И ИЗПОЛЗВАНЕ НА ОПРЕДЕЛЕН КЛАС ИНФОРМАЦИОННИ СИСТЕМИ

Любомир Станчев

Статията е свързана с проектирането и частичната реализация на софтуерния продукт "Визуален Обектно Ориентиран Шел на Информационна Система" (ВООШИС), който позволява създаването и експлоатацията на определен клас информационни системи (ИС). Кръгът на разглежданите ИС е ограничен до ИС следящи променящите се характеристики на обекти в системата, както и тяхното местоположение спрямо съдържащите ги обекти. Интерфейсът към ВООШИС е графичен, като въвеждането на обекти в системата може да става чрез тяхното изчертаване, а преместването им с помощта на мишката и механизма "влачене и пускане" (drag and drop).