The realization of the OO database in the specific CAD/CAM applications

Dejan S. Aleksić[1], Dragan S. Janković[2]

Abstract – **This paper describes the realization of the object-oriented database in the specific CAD/CAM software package supporting the design and manufacturing of the facade carpentry. The application of this software package creates a number of limitations and problems for whose solution a class group Attr API was created. Attr API class allow us to generating the various application variants, starting from the usual desktop application, to the client/server and Internet variant application.**

Keywords – **CAD/CAM database, object-relational mapping, OODB, design and manufacturing of the facade carpentry.**

I. INTRODUCTION

Specialized CAD/CAM programs commonly solve a small number of problems. They are justified as long as they meet the user's needs and solve the problem they were created for. On the other hand, narrowing of the application range does not necessarily lead to decreasing the number of solutions general enough to be used in all other/future applications within the specific area.

This paper illustrates the realization of the object-oriented database in the CAD/CAM software package supporting the design and manufacturing of the facade carpentry [1].

Tha facade carpentry is made out of a number of profiles and items together with the corresponding parts needed to join and assemble them. Profiles may be made of various materials such as aluminium, PVC, wood, iron, or their combinations (aluminium-wood, PVC-aluminium). There is a great number of profile manufacturers, each of them producing lots of profile systems. All profile systems are supposed to abide to the general rules and standards, and yet, each one of them has its own specific characteristics which make it unique. It is precisely because of this that the software packages specialized in giving support in that area should, beside the obviously narrowed application area, offer the solutions general enough to accept and process all the specific characteristics of a profile system. A particular difficulty arises from the fact that it is almost impossible to foresee all the specific solutions in the profile systems which are likely to appear in the future and which must undoubtedly be supported by such software paskages.

[1]Dejan S. Aleksić is with the Faculty of Sciences and Mathematics, University of Nis, Visegradska 3, 18000 Nis, Serbia, e-mail: dejan_aleksic@yahoo.com.

[2]Dragan S. Janković is with the Faculty of Electronics, University of Nis, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: dragan.jankovic@elfak.ni.ac.yu.

II. THE CHARACTERISTICS OF APPLICATION

The application of this kind of CAD/CAM as well as its users creates quite a number of characteristics and limitations [2]. We will deal with the most characteristic ones in this paper.

The information concerning the profile systems must undoubtedly be kept in the database; yet, writing in the base occurs rarely whereas reading of the same data is quite intensive, so that the base is Read Only DB in most of the cases. Besides the need for intensive reading of the relatively small amount of data, the critical factor is the reading speed which directly affects the speed at which the main evaluation module works. The factor of the data access speed to the profile system is one of the most significant ones which should be considered with the utmost care in the realization of such an application [3].

It has already been mentioned that there are a great number of profile systems having a lot of common characteristics and rules which will certainly be valid for all future profile systems. Apart from having the common characteristics, each profile system possesses some specific traits which make it different/unique compared to other systems. It is almost impossible to foresee which characteristics some future profile system will have. The second important factor is the possibility to modify the existing application so that it could accept the specific characteristics of the new systems.

Besides the profile systems data, the manipulation with the particular projects data should be attained. The project data may be kept either in the database or on the disk as an independent file [4]. The project data are usually stored in the base in the server/client software variant, but in case of physically separate representatives or operations, the variant of storing the projects into the data bank is used. Each project contains some data concerning the profile system which are practically the copy versions of these data stored in the description base of the profile systems. It is precisely this doubling of the data in the base and projects with the aim of maintaining them (the data changes by number and type, the content changes or the version improvement) and the synchronization after the changes that represent the problem we encounter next.

We are rather inhibited concerning the database choice by the fact that these applications are mainly used by small firms with an insufficient knowledge regarding the database and system maintainance, and by the fact that those applications are supposed to be of a reasonable price.

In accordance with what has been mentioned previously, it is clear that the description of the profile systems and the rules within them will be performed (with rare exception) not by the users themselves, but that the application will have to be

delivered with the already 'filled' bases for the particular profile systems. Such a 'centralized' maintainance of the profile system description and rules database requires a very sophisticated version improvement system of both the database and the application itself.

The multilingual support of the UI application characterizes every resourceful software package and there are quite a number of 'ready-made' solutions for its realization. The problem we encounter arises from the fact that the data stored in the profile system description base have to be translated. The substantial complexity of the system descriptions, their great number and the necessity to support a rather large number of languages make the simple creation and maintainance of the separate base for each profile system and each language impractical and almost impossible.

Concerning the great number of systems, the complexity of the project data processing and the complex report system, it is clear that a hierarchical organization of the data within both the system description base and the project itself is needed.

III. The System Description using Attr API

Regarding the mentioned facts, we chose the open source relation database FireBird. Since this database is available to lots of OS platforms the easy migration from the single user variant to the client/server variant was created. Thus all the requirements concerning the availability, price, easy installation and maintainance were fulfilled.

Unfortunately, such a choice brought about some limitations especially those regarding the fixed number of the fields within the tables and the beforehand defined data types, as well as the data reading speed. Our soulution was based on the creation of a separate group of objects within the application itself that gave us flexibility concerning the number and type of the entity descriptions data, but with the realized mechanism for the data type check. The basis of this approach is the usage of the so called attribute, i.e. the base class tAttribute with the group of its descending classes. Each entity can be described using the arbitrary number of attributes, and each attribute has its type and the momentary value from the list of possible values. Such an approach created a number of advantages, such as the easy realization of some small and yet big enough differences in the profile system description, the multilingual concept support within the base, the easy improvement of the existing database version, the automatic copying of the database into the memory structures and the disk and vice versa [5], the quick and easy project data recording and input onto/from the disk, the automatic data improvement in the base, the possibility to realize the automatic improvement of the recorded project data in accordance with the data changes in the base concerning their value, number and type, the realization of all the advantages created by the object-oriented data approach (data abstraction, inheriting, attribute overburdening) over the data stored in the base, memory or disk.

All the manipulations concerning the input, changes and reading of the attributes were realized within the tAttributes class and tAttributeList.

IV. The Attr API Realization

The *Attr API* itself contains a number of classes but its total functionality can be presented through the three main classes - *tAttribute*, *tAttributeList* and *tXalNode* with their descending classes. The basic functionality when working with attributes was realized in the base class *tAttribut*e. Precisely speaking, class *tAttribute*, being the base class, does not contain much of the functionality but it only defines the basic variables and methods (virtual and abstract) that will be copied and realized in one of the descending classes *tAttrString*, *tAttrInteger*, *tAttrReal* and *tAttrBoolean* (see Fig. 1.). Initially, the support for the four basic data types was realized, although it is possible to realize the support for some arbitrary data type through the new hereditary *tAttribute* class.

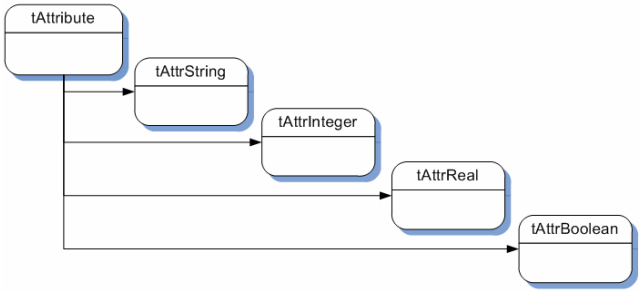

Fig. 1. Inherites tree of base class tAttribute

The second basic class is *tAttributeList* by which the keeping and manipulation of the attribute list was realized, more precisely of the instances of the base class *tAttribute,* i.e., its descending classes. The number of attributes, their type and value may be completely arbitrary.

The third crucial class of *Attr API* is *tXalNode* as the hereditary class descending from the class *tList* (see. Fig. 2.). Its primary task is the formation and manipulation of the basic hierarchical structures. Each instance of this class represents the knot of the hierarchical structure. Each knot of that structural tree has its own list of attributes which describe its state [see Fig. 2.].

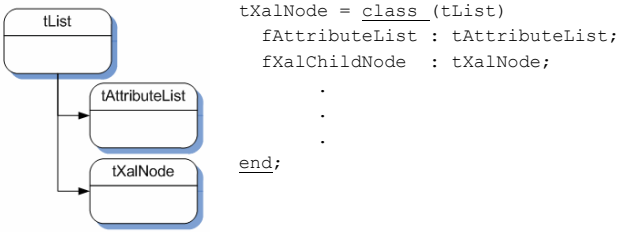

Fig. 2. *tXalNode* - crucial class of *Attr API*

The *Attr API* classes make possible the hierarchical organization and processing of the data stored in the memory as a group of objects – instances of those classes. The data access is quite quick in that case, but the need for a permanent keeping of those data still remains. The input/read process of the hierarchical structure of the *Attr API* objects in/from the base was completely realized within the basic classes, which is completely transparent for the class' users. All the data, together with their hierarchichal structure, no matter how complex it may be or what type or number of attributes are stored in it, are put in only three tables within the relation

database (see Fig. 3.). Furthermore, if lots of hierarchical structures are defined in one application (using *Attr API* classes), then all of them are placed in the same database in the already mentioned three tables.

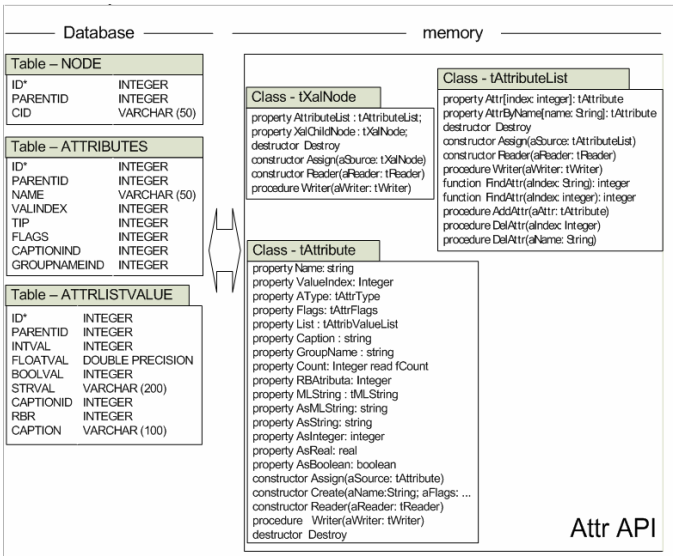

Fig. 3. Relationship between data in relation database and memory (in class of *Attr API*)

It is important to emphasize the possibility of the easy creation of the complex hierarchical structures by using the classes from *Attr API*. The user creates the new classes by commonly inheriting the class *tXalNode*. The additional functionality is realized by copying the methods from the parent class and by creating the new methods. The data in those newly defined classes 'are kept' in the attribute list (class *tAttributeList*) which makes the input/read process of the data in/from the base completely transparent for the user because it has already been realized in the base classes. On the other hand, the user has an absolute freedom in its own class while creating the attributes concerning both their number and their type or values.

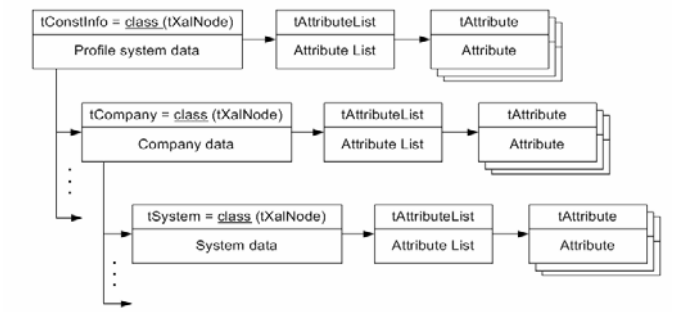

Fig. 4. The profile system description data

The Fig. 4. shows the hierarchical structure created for the manipulation with the profile system description data. The data from this structure are typically edited with a special program (*ConstInfo Manager*) and are put into the database. The main application only reads the data from that structure so that they are quite often cashed because of the data reading speed.

The second example is the hierarchical structure which manipulates the project descripition data (see. Fig. 5.).

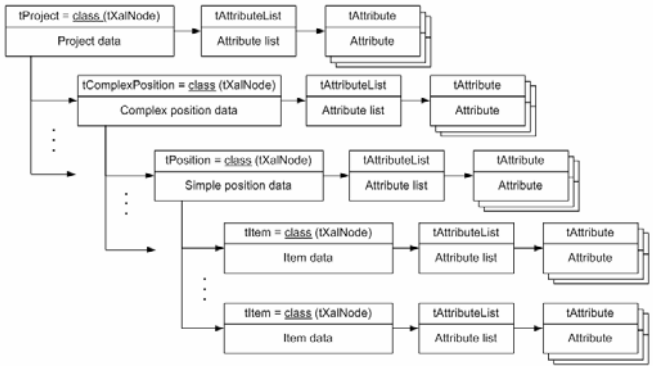

Fig. 5. The hierarchical structure for the project descripition data

Depending on the configuration of the main application, this structure is placed in the file on a disk (common/classical application) or in the database (client/user application). The total functionality has already been realized in both cases in the base classes of *Attr API,* while the user himself has to decide where the data will be stored.

Generally speaking, the hierarchical structures of the Attr API objects can be recorded/input into the various forms and formats. The first of them is the usage of some of the relation databases, the choice of database not depending on the functionality. It is possible to read from one relation database within the same application and to write the processed data into the relation database of some other manufacturer. These operations are completely transparent for the Attr API class users. The working speed makes possible the updating of the data of a particular part of the hierarchical structure from the memory into the database without writing in all the data of that structure. Using the file on a disk for storing the object hierarchical structures is another way of permanently keeping the data from the Attr API objects.

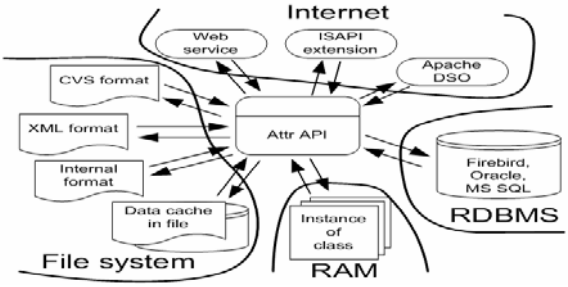

Fig. 6. Attr API objects can be recorded/input into the various forms

Several write-in data formats in the file on a disk are supported (see Fig. 6.) depending on the users' needs. Each one of the formats has its own advantages and disadvantages concerning the crucial parameters, such as the speed of write/read (internal format), the uncomplicated exchange of data with other general programs (CVS format) or in self-desc format (XML). A possibility of the automatic data copying creation from the database into the file on a disk should be particularly emphasized, since it is frequently used for cashing of data on the client side.

The data availability on the Internet with the maximum possible protection is realized by the special modules so that client can exchange data with the server side. The premise of the transparency of the way of keeping and accumulating data

is completely maintained and it is realized without any additional actions on the part of the users.

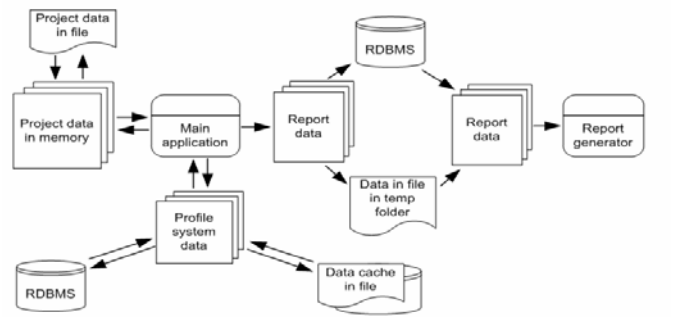

Fig. 7. One common application

This fact allows us the easy realization of the various configuration applications. The Fig. 7. illustrates one common application which uses *Attr API* for keeping and manipulating with the data. There are three different hierarchical structures – the project data, the profile systems data and the report generating data.

In this case, the project data are stored in the file on a disk while the profile systems data (read only) are usually automatically cashed from the database into the file on a disk.

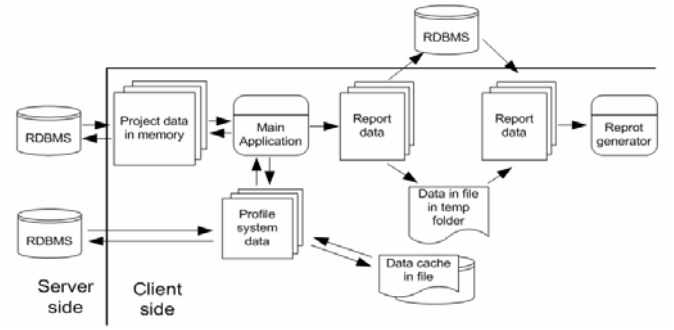

Fig. 8. The client/server variant of application

The data needed for the report generating are stored in either temporary file on a disk or the database itself, depending on the fact whether the user wants to have the reports available later or not. The possibility of the module realization for the report generating not depending on the main application version appears to be an additional advantage. Since the data format is not fixed, and concerning the fact that it is possible to detect the existance of the particular attribute in the hierarchical structure by using the program, then the report generating of various forms becomes quite easy.

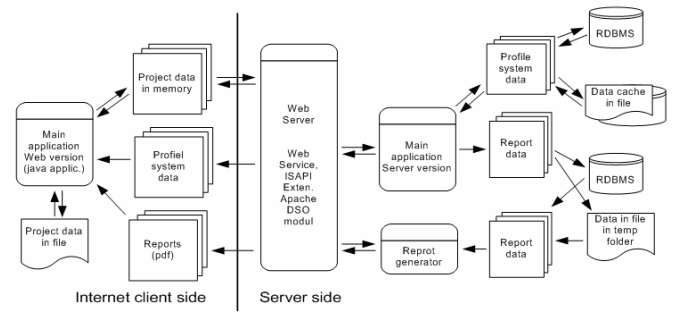

Fig. 9. The web client/server variant of the application

The advantages of using Attr API are particularly evident if we want to realize the client/server variant of our application (see Fig. 8). All we need to do is to define the place and way of storing the data in the configuration Attr API. It is important to point out that it is not necessary to change the remaining part of the application which manipulates the hierarchichally organized data. Following the given logic, the realization of the web client/server variant of the application is not too complicated. The server page has to undergo minimal changes in the logic, whereas the situation is quite similar on the client page, apart from the fact that the main application in the web client variant and Attr API are realized in the Java program language.

V. CONCLUSION

This paper illustrates the realization of the object-oriented database in the CAD/CAM software package for the support of the design and manufacturing of the facade carpentry. The application of this software package creates a number of limitations and problems for whose solution a class group *Attr API* was created. It gives us a necessary flexibility concerning the entity description data number and type with the possibility of checking the data types. Each entity may be described using an arbitrary number of attributes which have a defined type and value from the list of the possible values. This attribute concept offers a number of advantages concerning the modelling of small but considerable differences in the description of profile systems. The classes of Attr *API* allow for the hierarchical organization of the attributes for the profile systems or project description. The process of write/read hierarchical structure of the Attr API objects in/from the database is completely transparent for the *Attr API* users. Generally speaking, the hierarchical structures of Attr API objects can be save/load in various forms and formats (RDBMS, memory, file system, Internet, ...).

All these facts allow for a relatively easy generating of the various application variants, starting from the usual desktop application, to the client/server and Internet variant application.

BIBLIOGRAPHY

[1] Q. Zhang, "Object-oriented database systems in manufacturing: selection and applications", Industrial Management & Data Systems, vol. 101, no. 3, pp. 97-105, 2001.

[2] M.L. Brodie, B. Blanstein, U. Dayal, F. Manola, A. Rosenthal, "CAD/CAM Database Management", IEEE Database Engineering, Vol.7, No.2, pp. 12-20, June 1984.

[3] D. Maier, "Making database systems fast enough for CAD applications", Object-oriented concepts, databases, and applications, ACM Press, New York, NY, 1989.

[4] Ying-Kuei Yang, "An enhanced data model for CAD/CAM database systems", Proceedings of the 25th ACM/IEEE conference on Design automation, pp. 263 - 268, 1988.

[5] M. Blaha, W. Premerlani, H. Shen, "Converting OO Models Into RDBMS Schema" IEEE Software, vol. 11, no. 3, pp. 28-39, May/June 1994.