The use of scripts in a CAD/CAM database

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

Abstract – **Definition and the execution of combining profiles in profile systems represent one of the key problems in a specific CAD/CAM application for design and manufacture of facade carpentry support. General rules which are common for all the systems are executed within the application. Main mechanism of accomplishing specific rules of combining is comprised of executing scripts affiliated to elements on the lowest hierarchy level of profile system within the database. The script language is specially created for this use and concepts enable us to relatively easy model the rules of combining of some new system with all the specifics it carries with it.**

Keywords – **Scripts , OODB, CAD/CAM, Object-oriented computing, facade carpentry**

I. INTRODUCTION

Specialized CAD/CAM programs usually serve as a solution for only a narrow specter of problems. To justify its existence, they must be maximally adjusted to the user and optimized for the problem they are used for. On the other hand, narrowing its area of usage does not necessarily mean that such programs do not need general enough solutions so that they can be applied to all other/future uses from that area.

Throughout this work, we will illustrate one object-oriented database in a CAD/CAM software package for the project support and the manufacture of the facade carpentry.

During the process of the facade carpentry creation a cluster of profiles and fillings are used, together with the associated parts for their connection and assembly. Profiles must be made of different kinds of materials such as aluminum, PVC, wood, iron or their combination (aluminum-wood or PVC-aluminum). There are lots of profile manufacturers and each of them has a number of profile systems that are used and manufactured. All the profile systems must follow certain general rules and standards, but, on the other hand, each one of them has some specific characteristics which make them unique. This is the reason why software packages specialized for the support in that area, beside beyond doubt narrowed field of application, must have enough general solutions to acquire and process all of the specifics of a profile system. Special difficulty lies in the fact that it is almost impossible to foresee all the specific solutions in profile systems which can appear in the future and which must certainly be supported by these software packages.

[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 PRINCIPLE OF ONE APPLICATION – MULTIPLE BASE

By analyzing qualities of the elements and rules of combining, certain laws might be noted which govern all the profile systems. However, each of the profile systems has certain specifics which make it different compared to a competition's product. Systems basically never differ more than 5% to 20% depending on the material, area of use and the manufacturer.

In order for a CAD/CAM application to be applied to a design process and the manufacture of windows and facades, all the elements of the system must be described, and all the rules for their combining must be carried out. The storage and organization of that data is the first problem that needs to be solved.

Being unable to generalize all the rules within the profile system presents the main problem. That problem has been solved in multiple ways. One way is the existence of multiple application versions – one application for each profile system; or that the general rules are executed within the application itself and all the specifics of certain systems are being accomplished through external additional modules. Both suggested solutions posses obvious flaws – firstly in terms of application maintenance, ease of new profile system data input and the crucial one – defining new rules of profile combining.

In our case. general rules are executed within the application itself as well, but all the certain profile system specifics are recorded in the database. This is mainly accomplished by the strict hierarchy in the element organization, by introducing attributes for the element description and by defining and executing scripts (they are being kept in the database and executed within the application itself). So, during description of a new profile system only data within the database is being altered while the main application remains unchanged for all the systems, which leads to the principle of one application – multiple base.

III. DESCRIBING A SYSTEM USING ATTRIBUTES

All the data for the description of elements, as well as rules, are kept in a standard relational database. We have chosen an open-source database – Firebird to satisfy the needs in terms of availability, price, ease of installing and maintenance. Unfortunately, this kind of choice has certain limits such as: fixed number of fields inside tables, data types defined in advance and there are also limits in speed of reading the data [1], [2]. The solution we have applied in this case is based on creating a separate cluster of objects inside the application itself, which enables us the flexibility in terms of number and types of data. The base of this approach is the use of so called

attributes i.e. the base class tAttribute along with its subsequent class clusters. Each entity is described with a random number of attributes and each attribute has its type, current value from the list of possible values [3], [4]. This approach brought a number of benefits regarding ease of execution of small, but also large enough differences in profile system description, Multilanguage terms support within the database, ease of upgrading the existing database version, automatic data copy from the database into the memory or disk structures and vice versa, quick and easy writing and reading of data about projects on/from the disk, possibility of execution of automatic written project data upgrade in accordance with data changes in value, as well as in number and type, accomplishment of all advantages which object-oriented data access brings (data abstraction, inheritance, overload of attributes) over data located inside the database, memory or on disk.

All the actions concerning writing, editing and reading attributes inside the database are executed inside tAttribues class [5].

IV. HIERARCHY SYSTEM DESCRIPTION

It has already been mentioned that each profile system, besides its elements, is defined by the rules of combining those elements. All general rules of combining are executed in the main application code while the other, specific rules of combining are executed within the database itself. In order to model the profile system rules as efficiently as possible, all the system elements data is organized in a hierarchal tree in multiple fixed levels (see Fig. 1.).

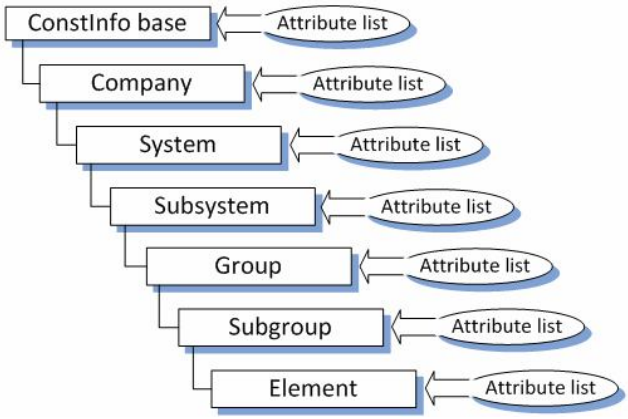

Fig. 1. Schematic display of hierarchal system description

The exception are elements of the lowest hierarchy level which can have their "children" i.e. elements that are added when adding parent-element itself. Process of adding "children"-elements is done by executing scripts which are allocated to each "child"-element and/or using conditional "child" adding system (see Fig. 2.).

V. DEFINING THE RULES OF COMBINING

Main mechanism of accomplishing specific rules of combining is comprised of executing scripts affiliated to

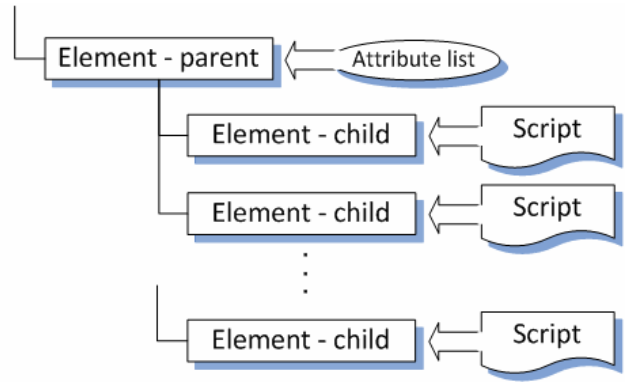

Fig. 2. Parent – child relation

elements on the lowest hierarchy level of profile system within the database. As mentioned before, one parent-element can have unlimited number of child-elements which are conditionally added after the parent-element is added first.

Process of adding one parent-element begins with forming an instance of tItem class inside the memory along with the list of attributes written in the tAttributeList class. After that, the list of children of that element is ran through and for each child-element a process of adding a child is executed. The whole process is done inside the script-executing module which takes an appropriate script from the database and acquires its input parameters (see Fig. 3.).

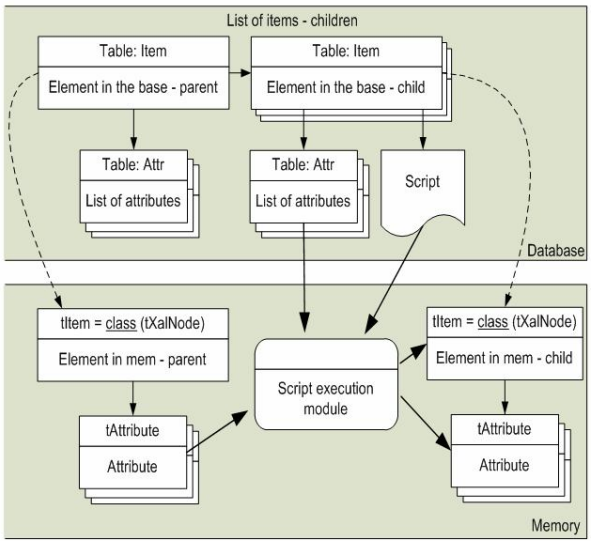

Fig. 3. Generating a new child-element by executing a script

Typically, script input parameters are two lists of attributes. The first one is a list of parents, and the second is a child-element attribute list. Inside the module, script commands are executed in order to get a list of attributes which are allocated to the newly formed instance of the tItem class for the child-element as a result (see Fig. 3.).

The script language (see Fig. 4.) alone has the following features: it follows the Pascal programming language syntax, there is a list of input and output parameters, beside basic types, types defined within the main application can be used (tItem, tAttribute, tAttributeList), possibility of defining local variables, possibility of defining local procedures and functions, access and editing child and parent attributes, access to data structures from the main application during

script execution, syntax script analysis during defining and the possibility of compiling scripts before writing to the database.

```
script  =
 script-heading  block "."
script =
 script ident. "(" ident.-list ")" ","
block =
 declaration-part  statement-part
declaration-part =
 [ type-definition-part ]
 [ variable-declaration-part ]
 proc-and-funct-declaration
proc-and-funct-declaration =
 {(proc-declarat|funct-declarat)";"}
proc-declarat =
 procedure-heading ";" body |
 procedure-heading ";" directive |
 procedure-identification ";" body
funct-declarat =
 function-heading ";" body |
 function-heading ";" directive |
 function-identification ";" body
procedure-heading =
 procedure ident.[formal-parameter-list]
function-heading =
 function ident.[formal-parameter-list]
":" result-type
statement-part =
 begin statement-sequence end
statement-sequence =
 statement { ";" statement }
assignment-statement =
 (variable|function-ident)":="expression
procedure-statement =
 procedure-ident.[actual-parameter-list]
while-statement =
 while expression do statement
repeat-statement =
 repeat statement-seq. until expression
for-statement =
 for variable-ident. ":=" initial-
expression (to | downto) final-
expression do statement
if-statement =
 if expression then statement [ else
statement ]
```

Fig. 4. Part of EBNF definition the script language

VI. CONDITIONAL ELEMENT ADDING

Conditional element adding represents yet another execution mechanism of specific combining rules. Its function represents the expansion of previously described child-element adding mechanism by executing scripts. Namely, one parent-element can contain the list of pairs criteria=values. Each criterion must be registered and corresponds to a certain attribute from the parent-element attribute list. List of criteria contains the list of child-elements with their scripts. When adding a parent-element a special module from the main application is activated which, based on parent-elements attribute values and the values of criteria from the lists, decides if and which list of children will be passed over to the script executing module (see Fig. 5.).

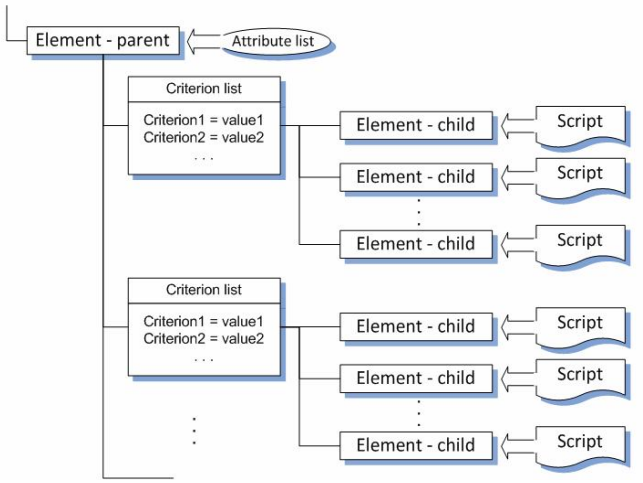

Fig. 5. Parent – child relation

Using this mechanism we emulate the complex if-then command within the very database. Its use could have been accomplished using special conditional commands within the scripts of each child-element as well, but the one we chose is much more efficient and clear especially concerning adding of child-elements.

VII. ONE EXAPMLE

Mechanisms described in previous section are illustrated in this section. We will used PVC profile system in this example. PVC profile is not strong enough and needs to insert Fe profiles inside PVC profile.

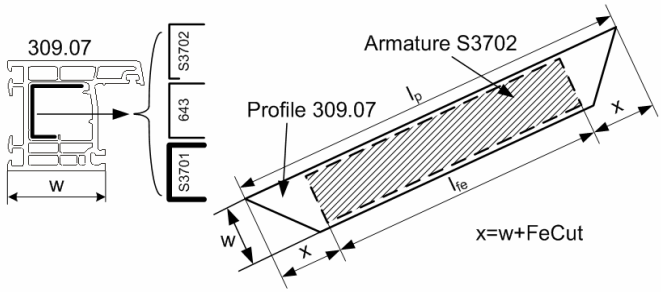

Fig. 6. One PVC profile and Fe armature profile

In one PVC profile can be inserted different Fe profiles depending of length of PVC profile, intensity of wind in this area, atc. Typically, angle of cutting Fe profile is 90° - 90° and his length must be less then length of parent PVC profiles (see Fig. 6).

Assumed that one instance of tItem class and his appropriate attribute list was made for PVC profile with catalogue name 309.07. Values of some typical attribute for this profile was show on Fig. 8 (left side).

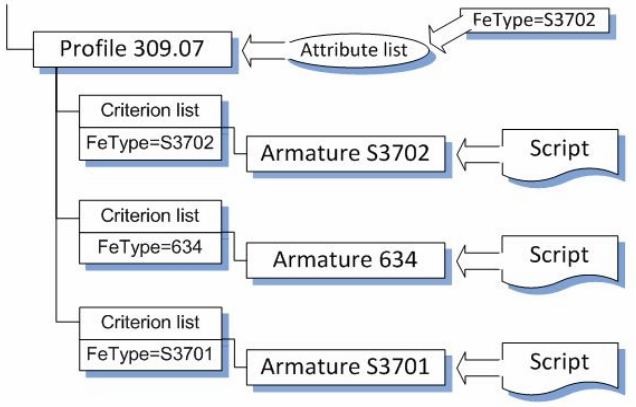

Fig. 7. PCV profile 309.07 (parent) with his "children"

In next step, conditional element adding mechanism is activated and some element - children of PVC profile will be adding.

```
CatName  = "309.07"
Width    = 88.0
Len      = 1206.00
U1       = 135.00
U2       = 45.00
Qty      = 4
FeType   = "S3702"
FeCut    = 5.0
minFeLen = 600
     . . .
```

```
CatName = " S3702"
Len     = 0.00
U1      = 0.00
U2      = 0.00
Qty     = 0
     . . .
```

Fig. 8. Attribute value of PVC profile (left) and Fe profile (right)

In this case, three diferrent Fe profile (S3702, 634 and S33701) can be added depending of value of FeType attribute in attribute list of PVC profile 309.07 (see Fig. 7.). S3702 Fe profile was selected and his attribute list will be loading from hierarchal system description data structure with default attribute value (see Fig. 1.).

```
script AddFe (p:tItem; var c : tItem)
begin
  c.Len := p.Len-(p.w+p.FeCut)*2;
  c.U1 := 90.00;
  c.U2 := 90.00;

  if p.Len > p.minFeLen
  then c.Qty := p.Qty
  else c.Qty := 0
end.
```

Fig. 9. Script for Fe profile adding

After that, appropriate script will be assign to S3702 Fe profile (see Fig. 7.). Attribute list of PVC profile 309.07 (as parent), attribute list of Fe profile S3702 (as child) and appropriate script will be sent to script-execution module (see Fig. 3.). Value of some children attributes will be changed during script executions and lenght and cutting angle of Fe profile will be define. In most of PVC profile systems govern follow rule: if lenght of PVC profile is less then some critical value appropriate Fe profile not need to add. This rule is modeling with if-then-else command on the end of script (see Fig. 9.). On the end of this process, instance of tItem class

```
CatName = "S3702"
Len = 1020.00
U1  = 90.00
U2  = 90.00
Qty = 4
     . . .
```

Fig. 10. Final value of typical attribute of new Fe profile S3702

who represents S33702 Fe profile will be create if value of attribute c.qty is great then 0. Final value of typical attribute of new Fe profile S3702 is shown on Fig. 10.

VIII. CONCLUSION

Definition and the execution of combining profiles in profile systems represent one of the key problems in a specific CAD/CAM application for design and manufacture of facade carpentry support. General rules which are common for all the systems are executed within the application, and all the specifics that certain systems possess are executed within the database. When describing new profile system only information in the database is altered while the main application remains the same for all systems, which brings us to one application – multiple base principle. Information on system elements has hierarchal organization in multiple levels, where lowest level elements can have “children”. Process of child-element adding is accomplished by executing scripts allocated to each child-element and/or with the conditional child adding system. The script language is specially created for this use, follows the Pascal programming language syntax, acquires the list of input and output parameters; beside basic types, types defined within the main application can be used. There is a possibility of defining local variables, procedures and functions, access to and editing of parent/child attributes as well as access to data structures from the main application during the script execution. Before writing the script into the database, a syntax script analysis is performed and there is a possibility of compiling scripts which significantly increases their execution.

All these concepts enable us to relatively easy and within a short period of time model the rules of combining of some new system with all the specifics it carries with it.

REFERENCES

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

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

[4] Y. Yang, "An enhanced data model for CAD/CAM database systems", Proceedings of the 25th ACM/IEEE conference on Design automation, pp. 263 - 268, 1988.
P. Buneman, M. Atkinson, "Inheritance and persistence in database programming languages", Proceedings of the 1986 ACM SIGMOD international conference on Management of data, Pages: 4 – 15, 1986.