STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

A FORMAL SUPPORT SYSTEM FOR THE OBJECT ORIENTED ANALYSIS OF AN APPLICATION DOMAIN

S. BERAR, M. VANCEA, AND A. VANCEA

Abstract. The paper introduces a formal system aimed to assist the programmer in the object oriented analysis of an arbitrary application domain. The object oriented models shift the modelling semantics towards a conceptual modelling in the application's domain. Being given a graphical representation of the application's domain structure, we further generate a correct and consistent specification in an intermediary code, independent of the programming language in which the application will be developed.

1. Preliminaries

In approaching complex applications domains, classical data models show some limitations, comming mainly from the impossibility of modelling implicit knowledge, the limits imposed by the capacity of abstraction and the difficulties that arise when trying to access some composite parts of the involved objects. Object oriented models (OOM) shift the modelling semantics towards a conceptual modelling in the application's domain [2] ignoring the intensive use of notions like pointers or trees. OOM reduces the gap between reality and the model.

The paper presents the design of a system (partially implemented) which assists the programmer in object oriented analysis of an arbitrary application domain.

It is generally simpler to specify graphically the structure of an application domain. Starting from such a representation, one tries to generate a correct and consistent specification, in an intermediary code, independently from the programming language in which the application will be programmed. Such a specification can be further processed for obtaining an object oriented source code.

The system has to fulfil the following functions:

a): Offering a friendly user interface for the graphical specification of the involved classes and the relationships between them; based on the graphical

1991 *Mathematics Subject Classification.* 68M15, 68Q60, 68N05.

1991 *CR Categories and Descriptors.* D.1.5 [Programming Techniques]: Object-oriented Programming; D.2.1 [Software Engineering]: Requirements and Specification – languages, methodologies; D.2.4 [Software Engineering]: Program Verification – reliability, validation.

specification, the algorithm presented in section 3 will generate an adequate model for the chosen application;

b): Verifying the corectness and the consistence of the user defined model, signaling the possible occuring errors;

c): Offering the possibility of the dynamic modification of the model and of completion of an existing one;

d): Generating a specification file based on the graphical representation developed by the user.

2. The formal description of the problem

We will further consider the *object* as introduced by Booch [5]: an object is an entity which has a *state*, *behaviour* and *identity*, the structure and behaviour of similar objects being defined in their common class.

An *object oriented model* is a tuple $(C, IS-A, O)$, where C is a type system, $IS-A$ represents the structural and behavioural inheritance relationship (which introduces a partial ordering relation in the type system) and O is a type that generalizes all others types, being the root of the type hierarchy [4].

The set C can be divided in three disjoint sets:

- C_A - the type set identified in the application domain;
- C_{AUX} - the auxiliary type set from the solution domain;
- C_B - the basic type set.

Considering that the C_A types are completed with the implementation component, we can approach the classes identified in the application domain. Our system intends to support the user in the specification process of these classes.

We will assume that the inheritance relation $IS-A$ is represented as the tuple set

$$SI = \{< C_i, C_j > \in C \times C | (C_i IS\text{-}A\ C_j)\}.$$

We consider also that a class C_i is internally represented in our system by a data structure of the form

```
class_name: string;
structure : set_of_attributes;
interface : set_of_methods;
```

We will denote by an *attribute* a tuple of the form:

$$< attr\_name{:}string,\ attr\_domain : C_A \cup C_B > .$$

A *method* is declared by a tuple of the form:

$$< method\_name{:}string,\ method\_domain{:}tuple\_of(C_B \cup C_A),\ method\_result : C_A \cup C_B > .$$

We have further to define the following elements:

- a basic set G of parameterized graphical symbols with which the description will be made;

- a couple of functions $(T_C : G \rightarrow C_A, T_I : G \rightarrow SI)$ which express the correspondence between the graphical symbols instantiated by the user and classes, and the inheritance relationship between classes, respectively;
- a set of restrictions $SC_I : SI -> BOOL$, which expresses the validity of the defined inheritance relationships;
- a set of restrictions $SC_C : C_A -> BOOL$, which expresses the structural and behavioural consistence and correctness requirements of the classes.

3. The basic algorithm of our system

The functionning of the support system is defined by the following sequence:

Step 1: Initialization of the C_A and SI sets with the empty set (for the cases in which we start a new model description) or with C_{A0} and SI_0 (when we start from a partially existent model);

Step 2: Starting from C_A and SI we build the set of graphical instantiated symbols, D_0, which describes the initial state of the system. We initialize the work set $D = D_0$.

Step 3: The user will instantiate a sequence of graphic symbols $g_i \in G$ by giving values to their parameters. It is also possible the elimination of some elements $ge_j \in D$. The set $D = D \cup \{g_i | \forall i\} \backslash \{ge_j | \forall j\}$ is updated accordingly.

Step 4: T_C and T_I are applied on D, resulting the new forms of C_A and SI.

Step 5: The validity of the generated model is evaluated with the formula $V = CV \cap IV$, where

$$CV = \cap CC_i(c_j), \text{ for all } CC_i \in SC_C, c_j \in C_A,$$

$$IV = \cap CI_i(s_j), \text{ for all } CI_i \in SC_I, s_j \in SI.$$

Step 6: If V evaluates to false, the algorithm goes on from step 3. Otherwise, a procedure for building and printing the specification is started, based on C_A and SI. The algorithm of this operation is simple and depends on the system's internal implementation.

The algorithm can be restarted at the user's request until obtaining a satisfactory specification, adequate to the considered application domain. The output of the algorithm is a text file which contains the description (following the BNF syntax presented below in section 4) of the classes generated at step 6.

Our implementation was programmed in C++ under Windows, for having graphic facilities regarding the development of the graphical procedures.

4. The syntax of the specification language

The syntax of the generated specification language given in BNF form is adapted from [1] and follows below:

<class>::=**class** <name>
superclass <list_domains>
structure <list_attributes>
interface <list_methods¿
end <name>.
<list_domains> ::= <domain> | <list_domains>;<domain>
<list_attributes> ::= <attribute>;<list_attributes> | ε
<list_methods> ::= <method>; <list_methods> | ε
<domain> ::= <name> | <base_type>
<base_type> ::= **int** | **real** | **string** | **char** | **boolean**
<attribute> ::= <name>:<domain>
<method> ::= <input> → <output>
<input> ::= ε | <domain> | <domain>x<input>
<output> ::= <domain>
<name> ::= **identifier**

Steps 5 and 6 of the algorithm presented in section 3 assures the validation and the correct generation (meaning to follow the restrictions imposed at step 5) of the specification language described above. We have thus that the generated specification will follow the defined restrictions set.

5. The set of graphical symbols

The structure of a graphical symbol is formed by a tuple of parameters $(p_1, p_2, \ldots p_n)$, n varying upon the type of that symbol.

We will use the following graphical symbols: **class (a)**, **inherit (b)**, **aggregate one (c)**, **aggregate many (d)**, based on the OMT notation [6]. These assure the expressiveness of the classes specification and of the orthogonal hierarchies *IS- A* and PART-OF. For specifying the methods of a class one will use also a graphical symbol, namely **method (e)**.

A common parameter to all those symbols is *shape*, which refers to the graphical aspect presented to the user. In figure 1 it is presented the value of this parameter for every symbol used.

6. Model validation

Model validation means verifying the restrictions set $SC_C : C_A -> BOOL$, which expresses the correctness and structural and behavioural consistency requests of the classes, and respectively of the restrictions set $SC_I : SI -> BOOL$, which corresponds to the inheritance relationships [2].

The *classes validation* is accomplished at the moment at which all the classes were described. This validation is composed of:

- verifying if the classes names (class_name) are unique and if the attributes and methods names are unique in the frame of the same class;

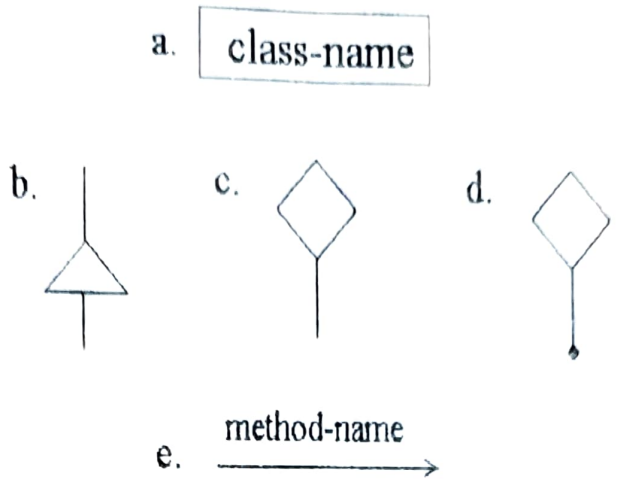

FIGURE 1. The values of the *shape* parameter for the graphical symbols from G

- verifying the existence of the attributes definition domains and also of the domains on which the methods are defined; these could be implicit domains or user defined classes.

The *relationships validation* refers particulary to multiple inheritance restrictions verification. Multiple inheritance is allowed when there are no two distinct descendant paths [4]. But, considering that in the model of an arbitrary application domain all the classes emerge from an abstract class called ROOT, it follows that any multiple inheritance relationship leads us to multiple paths. From this reason, in the program, multiple inheritance will be signaled as a validation error.

7. Conclusions

In the paper a theoretical model for designing, and afterwards for syntactic and semantic validation of the object oriented applications is introduced and developed. These actions take place in the context proposed in section 4.

The main advantage of using this specification and validation method resides in simplifying the design and generation of the object oriented applications. Our implemented system has mainly a didactic goal and helps the user in the correct design and reliability of his object oriented applications.

References

[1] B. Stroustrup, *The C++ Programming Language*, Addison-Wesley Series in Computer Science, 1986.

[2] B. Eckel, *Using C++*, Osborne/McGraw-Hill, 1990.

[3] * * *, *BORLAND C++: Programmer's Guide*, 1990.

[4] I. Salomie, *Object Oriented Programming Techniques*, Ed. Microinformatica, Cluj-Napoca, 1995 (in Romanian).

[5] G. Booch, *Object Oriented Design with Applications*, The Benjamin/Cummings Publishing Company Inc., 1991.

[6] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modelling and Design*, Prentice Hall International, 1991.

Babeş-Bolyai University, Faculty of Economics, RO 3400 Cluj-Napoca, str. Kogălniceanu 1, Romania

E-mail address: {sanda,vancea}@econ.ubbcluj.ro

Babeş-Bolyai University, Faculty of Mathematics and Informatics, RO 3400 Cluj-Napoca, str. Kogălniceanu 1, Romania

E-mail address: vancea@cs.ubbcluj.ro